

Tracking Moving Objects with Few Handovers

David Eppstein, Michael T. Goodrich, and Maarten Löffler

Dept. of Computer Science, Univ. of California, Irvine

Abstract. We study the online problem of assigning a moving point to a base-station region that contains it. For instance, the moving object could represent a cellular phone and the base station could represent the coverage zones of cell towers.

Our goal is to minimize the number of *handovers* that occur when the point moves outside its assigned region and must be assigned to a new one. We study this problem in terms of a competitive analysis measured as a function of Δ , the *ply* of the system of regions, that is, the maximum number of regions that cover any single point.

In the offline version of this problem, when object motions are known in advance, a simple greedy strategy suffices to determine an optimal assignment of objects to base stations, with as few handovers as possible. For the online version of this problem for moving points in one dimension, we present a deterministic algorithm that achieves a competitive ratio of $O(\log \Delta)$ with respect to the optimal algorithm, and we show that no better ratio is possible. For two or more dimensions, we present a randomized online algorithm that achieves a competitive ratio of $O(\log \Delta)$ with respect to the optimal algorithm, and a deterministic algorithm that achieves a competitive ratio of $O(\Delta)$; again, we show that no better ratio is possible.

1 Introduction

A common problem in wireless sensor networks involves the online tracking of moving objects [2, 6, 11, 14, 20, 26, 27]. Whenever a moving object leaves a region corresponding to its tracking sensor, a nearby sensor must take over the job of tracking the object. Similar *handovers* are also used in cellular phone services to track moving customers [22]. In both the sensor tracking and cellular phone applications, handovers involve considerable overhead [11, 14, 20, 22, 27], so we would like to minimize their number.

Geometrically, we can abstract the problem in terms of a set of n closed regions in \mathbb{R}^d , for a constant d , which represent the sensors or cell towers. We assume that any pair of regions intersects at most a constant number of times, as would be the case, say, if they were unit disks (a common geometric approximation used for wireless sensors [2, 6, 11, 14, 20, 27]). We also have one or more moving entities, which are represented as points traveling along 1-dimensional curves (which we do not assume to be smooth, algebraic, or otherwise well-behaved, and which may not be known or predictable by our algorithms) with a time stamp associated to each point on the curve (Figure 1).

We need to track the entities via regions that respectively contain them; hence, for each moment in time, we must assign one of the regions to each entity, p , with the requirement that p is inside its assigned

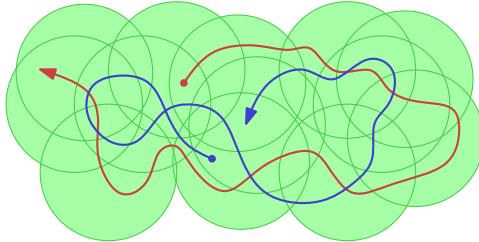


Fig. 1. Example input

region at each moment in time. Finally, we want to minimize the number of times that we must change the assignment of the region tracking an entity, so as to minimize the number of handovers.

We also consider a generalized version of this problem, where each point p is required to be assigned to c regions at each moment in time. This generalization is motivated by the need for *trilateration* in cellular networks and wireless sensor networks [15, 24], where directional information from three or more sensors is used to identify the coordinates of a moving point.

1.1 Related Work

There has been considerable previous work in the wireless sensor literature on mobile object tracking. So, rather than providing a complete review of this area, let us simply highlight some of the most relevant work from the wireless sensor literature.

Cao *et al.* [6] study the problem of modeling an object moving along a straight-line trajectory among uniformly-distributed unit-disk sensors. Their analysis involves a probabilistic study of when tracking is feasible if sensors enter their tracking states independently at random. Alaybeyoglu *et al.* [2] also study the object tracking problem using uniformly-distributed unit disks to model sensors, with their focus on the problem of identifying the tracking sensor with strongest signal in each case.

Zhou *et al.* [27] introduce the idea of using handovers to reduce energy in mobile object tracking problems among wireless sensor networks. Pattem *et al.* [20] study energy-quality trade-offs for various strategies of mobile object tracking, including one with explicit handovers. He and Hou [14] likewise study mobile object tracking with respect to handover minimization, deriving probabilistic upper and lower bounds based on distribution assumptions about the moving objects and wireless sensors. Ghica *et al.* [11] study the problem of tracking an object among sensors modeled as unit disks so as to minimize handovers, using probabilistic assumptions about the object's future location while simplifying the tracking requirements to discrete epochs of time.

The analysis tool with which we characterize the performance of our algorithms comes from research in *online algorithms*, where problems are defined in terms of a sequence of decisions that must be made one at a time, before knowing the sequence of future requests. Sleator and Tarjan [21], introduce *competitive analysis*, which has been used for a host of subsequent online algorithms (e.g., see [5]). In competitive analysis, one analyzes an online algorithm by comparing its performance against that of an idealized adversary, who can operate in an offline fashion, making his choices after seeing the entire sequence of items.

We are not aware of any previous work that applies competitive analysis to the problem of handover minimization. Nevertheless, this problem can be viewed from a computational geometry perspective as an instantiation of the *observer-builder* framework of Cho *et al.* [7], which itself is related to the *incremental motion* model of Mount *et al.* [18], the *observer-tracker* model of Yi and Zhang [26], and the well-studied *kinetic data structures* framework [1, 4, 12, 13]. In terms of the observer-builder model, our problem has an *observer* who watches the motion of the point(s) we wish to track and a *builder* who maintains the assignment of tracking region(s) to the point(s). This assignment would define a set of Boolean *certificates*, which become *violated* when a point leaves its currently-assigned tracking region. The observer would notify the builder of any violation, and the builder would use information about the current and past states of the point(s) to make a new assignment (and define an associated certificate). The goal, as in the previous work by Cho *et al.* [7], would be to minimize the number of interactions between the observer and builder, as measured using competitive analysis. Whereas Cho *et al.* apply their model to the maintenance of net trees for moving points, in our case the interactions to be minimized correspond to handovers, and our results supply the algorithms that would be needed to implement a builder for handover minimization. Yi and Zhang [26] study a general online tracking problem, but with a different objective function than ours: when applied to mobile object tracking, rather than optimizing the number of handovers, their scheme would aim to minimize the distance between objects and the base-station region to which they are each assigned.

Several previous papers study overlap and connectivity problems for geometric regions, often in terms of their *ply*, the maximum number of regions that cover any point. Guibas *et al.* [12] study the maintenance

of connectivity information among moving unit disks in the kinetic data structure framework. Miller *et al.* [17] introduce the concept of ply and show how sets of disks with low ply possess small geometric separators. Eppstein *et al.* [9, 10] study road network properties and algorithms using a model based on sets of disks with low ply after outliers are removed. Van Leeuwen [23] studies the minimum vertex cover problem for disk graphs, providing an asymptotic FPTAS for this problem on disk graphs of bounded ply. Alon and Smorodinsky [3] likewise study coloring problems for sets of disks with low ply.

Our problem can also be modeled as a *metrical task system* in which the sensor regions are represented as states of the system, the cost of changing from state to state is uniform, and the cost of serving a request is zero for a region that contains the request point and two for other regions. Known randomized online algorithms for metrical task systems [16] would give a competitive ratio of $O(\log n)$ for our problem, not as good as our $O(\log \Delta)$ result, and known lower bounds for metrical task systems would not necessarily apply to our problem.

1.2 New Results

In this paper, we study the problem of assigning moving points in the plane to containing base station regions in an online setting and use the competitive analysis to characterize the performance of our algorithms. Our optimization goal in these algorithms is to minimize the number of *handovers* that occur when an object moves outside the range of its currently-assigned base station and must be assigned to a new base station. We measure the competitive ratio of our algorithms as a function of Δ , the *ply* of the system of base station regions, that is, the maximum number of such regions that cover any single point. When object motions are known in advance, as in the offline version of the object tracking problem, a simple greedy strategy suffices to determine an optimal assignment of objects to base stations, with as few handovers as possible. For the online problem, on the other hand, for moving points in one dimension, we present a deterministic online algorithm that achieves a competitive ratio of $O(\log \Delta)$, with respect to the offline optimal algorithm, and we show that no better ratio is possible. For two or more dimensions, we present a randomized algorithm that achieves a competitive ratio of $O(\log \Delta)$, and a deterministic algorithm that achieves a competitive ratio of $O(\Delta)$; again, we show that no better ratio is possible.

2 Problem Statement and Notation

Let \mathcal{D} be a set of n regions in \mathbb{R}^d . These regions represent the areas that can be covered by a single sensor. We assume that each region is a closed, connected subset of \mathbb{R}^d and that the boundaries of any two regions intersect $O(1)$ times – for instance, this is true when each region is bounded by a piecewise algebraic curve in \mathbb{R}^2 with bounded degree and a bounded number of pieces. With these assumptions, the arrangement of the pieces has polynomial complexity $O(n^d)$. The *ply* of \mathcal{D} is defined to be the maximum over \mathbb{R}^d of the number of regions covering any point. We always assume that \mathcal{D} is fixed and known in advance.

Let T be the trajectory of a moving point in \mathbb{R}^d . We assume that T is represented as a continuous and piecewise algebraic function from $[0, \infty)$ to \mathbb{R}^d , with a finite but possibly large number of pieces. We also assume that each piece of T crosses each region boundary $O(1)$ times and that it is possible to compute these crossing points efficiently. We also assume that $T([0, \infty)) \subset \cup \mathcal{D}$; that is, that the moving point is always within range of at least one sensor; this assumption is not realistic, and we make it only for convenience of exposition. Allowing the point to leave and re-enter $\cup \mathcal{D}$ would not change our results since the handovers caused by these events would be the same for any online algorithm and therefore cannot affect the competitive ratio.

As output, we wish to report a *tracking sequence* S : a sequence of pairs (τ_i, D_i) of a time τ_i on the trajectory (with $\tau_0 = 0$) and a region $D_i \in \mathcal{D}$ that covers the portion of the trajectory from time τ_i to τ_{i+1} . We require that for all i , $\tau_i < \tau_{i+1}$. In addition, for all i , it must be the case that $T([\tau_i, \tau_{i+1}]) \subseteq D_i$, and there should be no $\tau' > \tau_{i+1}$ for which $T([\tau_i, \tau']) \subset D_i$; in other words, once a sensor begins tracking the moving point, it

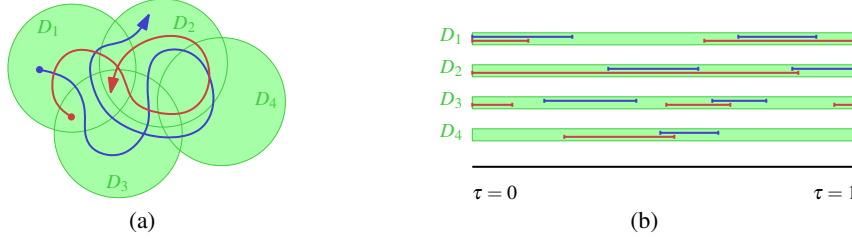


Fig. 2. An example of of four sensors and two trajectories, in the original setting (a) and the corresponding interval representation (b).

continues tracking that point until it moves out of range and another sensor must take over. Our goal is to minimize $|S|$, the number of pairs in the tracking sequence. We call this number of pairs the *cost* of S ; we are interested in finding tracking sequences of small cost.

Our algorithm may not know the trajectory T completely in advance. In the *offline tracking problem*, T is given as input, and we must find the tracking sequence S that minimizes $|S|$; as we show, a simple greedy algorithm accomplishes this task. In the *online tracking problem*, T is given as a sequence of *updates*, each of which specifies a single piece in a piecewise algebraic decomposition of the trajectory T . The algorithm must maintain a tracking sequence S that covers the portion of T that is known so far, and after each update it must extend S by adding additional pairs to it, without changing the pairs that have already been included. As has become standard for situations such as this one in which an online algorithm must make decisions without knowledge of the future, we measure the quality of an algorithm by its *competitive ratio*. Specifically, if a deterministic online algorithm A produces tracking sequence $S_A(T)$ from trajectory T , and the optimal tracking sequence is $S^*(T)$, then the competitive ratio of A (for a given fixed set \mathcal{D} of regions) is

$$\sup_T \frac{|S_A(T)|}{|S^*(T)|}.$$

In the case of a randomized online algorithm, we measure the competitive ratio similarly, using the expected cost of the tracking sequence it generates. In this case, the competitive ratio is

$$\sup_T \frac{E[|S_A(T)|]}{|S^*(T)|}.$$

As a variation of this problem, stemming from trilateration problems in cellular phone network and sensor network coverage, we also consider the problem of finding tracking sequences with *coverage* c . In this setting, we need to report a set of c tracking sequences S_1, S_2, \dots, S_c for T that are *mutually disjoint* at any point in time: if a region D appears for a time interval $[\tau_i, \tau_{i+1}]$ in one sequence S_k and a time interval $[\sigma_j, \sigma_{j+1}]$ in some other sequence S_l , we require that the intervals $[\tau_i, \tau_{i+1}]$ and $[\sigma_j, \sigma_{j+1}]$ are disjoint. We wish to minimize the total cost $\sum_{i=1}^c |S_i|$ of a set of tracking sequences with coverage c , and in both the offline and online versions of the problem.

3 Offline Tracking

Even though we focus on the case where the trajectories of the entities are not known in advance, we also study the offline tracking problem. We will use some of the observations and algorithms from the offline problem in our analysis of algorithms for the online problem.

As mentioned in Section 4, we may view the input as a sequence of events that describe when an entity enters or leaves the region belonging to a sensor. In other words, we can translate the offline tracking problem into a problem with one continuous dimension (time), where for each sensor we represent the set

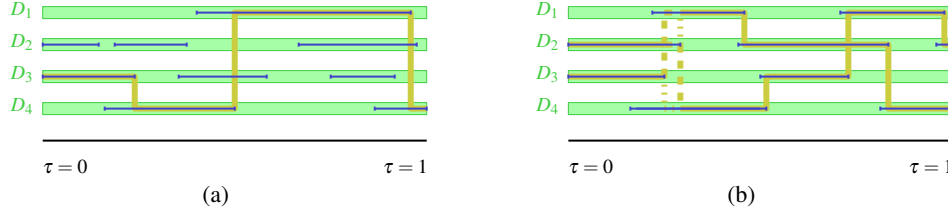


Fig. 3. Greed is good!

of times during which the entity is inside or outside that sensor as a set of intervals in this time dimension. Figure 2 shows an example of two different trajectories travelling in the same set of regions \mathcal{D} , and the corresponding interval representations of these trajectories.

3.1 Greedy Algorithm for Offline c -coverage Tracking

We describe a greedy algorithm for the offline problem where T needs to be covered by c disjoint sensors (trilateration) at any time. The original problem is a special case where $c = 1$.

In the greedy algorithm, We start with the c longest available segments at the start. Now, whenever we reach the end of an interval at time τ , we consider the set of available intervals (intervals that contain τ , and are not currently in use by one of the other $c - 1$ trackers), and always switch to the one that continues for the longest time into the future. Figure 3(a) shows a simple example for $c = 1$.

Theorem 1. *The greedy algorithm solves the offline tracking problem optimally, in polynomial time.*

Proof. Consider an arbitrary solution. First, we may assume whenever a path enters an interval, it stays there until the end of the interval. The only reason why it would leave is to make room for another path. But in that case, we could switch the roles of these two paths, leading to a better solution. Similarly, if a path reaches the end of an interval and does not jump to the longest available interval, this must be because another path uses that interval in the future. But then we can just select it anyway, and when the other path wants to select it, we send it instead to the place where our first path would have been at that time. See Figure 3(b). This gives another solution that is at least as good.

Assuming the input is given as a sequence of events, the greedy algorithm can be trivially implemented in time quadratic in the length of this sequence (this can likely be improved). Since we assume that the arrangement of the regions in \mathcal{D} has polynomial complexity and each piece of the trajectory has a constant number of intersections with the regions, the length of the event sequence is polynomial in n , the number of regions in \mathcal{D} , and m , the number of pieces of T . \square

4 Online Tracking

We now move on to the dynamic setting. We assume that we are given the start locations of the trajectory, and receive a sequence of updates extending the trajectory. From these updates we can easily generate a sequence of *events* caused when the trajectory crosses into or out of a region. We will describe three algorithms for different settings, which are all based on the following observations.

Let T be the (unknown) trajectory of our moving entity, and recall that $T(\tau)$ denotes the point in space that the entity occupies at time τ . Let τ_0 be the starting time. We will define a sequence of times τ_i as follows. For any i , let $p_i = T(\tau_i)$ be the location of the entity at time τ_i , and let $\mathcal{D}_i \subset \mathcal{D}$ be the set of

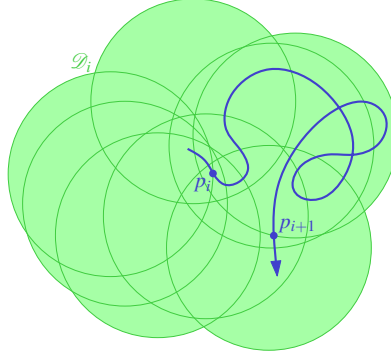


Fig. 4. The set \mathcal{D}_i of disks containing p_i , and the point p_{i+1} where the trajectory leaves the last disk of \mathcal{D}_i .

regions that contain p_i . For each $D_{ij} \in \mathcal{D}_i$, let τ'_{ij} be first the time after τ_i that the entity leaves D_{ij} . Now, let $\tau_{i+1} = \max_j \tau'_{ij}$ be the moment that the entity leaves the last of the regions in \mathcal{D}_i (note that it may have re-entered some of the regions). Figure 4 shows an example. Let τ_k be the last assigned time (that is, the entity does not leave all disks \mathcal{D}_k before the the end of its trajectory).

Observation 2 *Any tracking sequence S for trajectory T must have length at least k .*

Proof. For any i , a solution must have a region of \mathcal{D}_i at time τ_i . However, since by construction there is no region that spans the entire time interval $[\tau_i, \tau_{i+1} + \varepsilon]$ (for any $\varepsilon > 0$), there must be at least one handover during this time, resulting in at least $k - 1$ handovers, and at least k regions. \square

4.1 Randomized Tracking with Logarithmic Competitive Ratio

With this terminology in place, we are now ready to describe our randomized algorithm. We begin by computing τ_0 , p_0 and \mathcal{D}_0 at the start of T . We will keep track of a set of candidate regions \mathcal{C} , which we initialize to $\mathcal{C} = \mathcal{D}_0$, and select a random element from the candidate set as the first region to track the entity. Whenever the trajectory leaves its currently assigned region, we compute the subset $\mathcal{C} \subset \mathcal{D}_i$ of all regions that contain the whole trajectory from p_i to the event point, and if \mathcal{C} is not empty we select a new region randomly from \mathcal{C} . When \mathcal{C} becomes empty, we have found the next point p_{i+1} , giving us a new nonempty candidate set \mathcal{C} . Intuitively, for each point p_i , if the set of candidate regions containing p_i is ordered by their exit times, the selected regions form a random increasing subsequence of this ordering, which has expected length $O(\log \Delta)$, whereas the optimal algorithm incurs a cost of one for each point p_i . Refer to Algorithm 1 for a more formal description of the algorithm.

Lemma 1. *Algorithm 1 produces a valid solution of expected length $O(k \log \Delta)$.*

Proof. By construction, we produce a new time stamp τ_i as soon as the entity leaves all available sensors that contain p_{i-1} . This corresponds exactly to the times we switch sensors in the greedy algorithm of Section 3, which by Theorem 1 yields an optimal solution.

Next, we prove that between τ_i and τ_{i+1} the expected number of new regions is $O(\log \Delta)$. Recall that, for each $D_{ij} \in \mathcal{D}_i$, we defined τ'_{ij} to be the first time after τ_i that the entity leaves D_{ij} . These numbers τ'_{ij} form a set of at most $|\mathcal{D}_i|$ numbers. At each call to `pick_sensor`, we select a random number from this set that is larger than any number we chose before. The expected length of such a sequence is $\ln |\mathcal{D}_i| + O(1) = O(\log \Delta)$. \square

Combining Observation 2 and Lemma 1, we see that Algorithm 1 has a competitive ratio of $O(\log \Delta)$.

Algorithm 1 Randomized online tracking algorithm.

We keep global variables i , \mathcal{C} , and S .

Initialization:

1. set $i = -1$
2. call $\text{next_step}(\tau_0)$
3. call $\text{pick_sensor}(\tau_0)$

Procedure $\text{next_step}(\tau)$:

1. increment i
2. set $\tau_i = \tau$
3. compute p_i and D_i (see Section 4.4 for efficiency considerations)
4. set $\mathcal{C} = D_i$

Procedure $\text{pick_sensor}(\tau)$:

1. take a random element $C \in \mathcal{C}$
2. append (τ, C) to S

Handle event (τ, D) :

1. if the event is a region-enter-event, ignore it
2. if $D \in \mathcal{C}$ then
 - (a) set $\mathcal{C} = \mathcal{C} \setminus \{D\}$.
 - (b) if \mathcal{C} now is empty, then call $\text{next_step}(\tau)$
 - (c) if D is equal to the last region in S , then call $\text{pick_sensor}(\tau)$

When there are no more events, output S .

4.2 Deterministic Tracking with Linear Competitive Ratio

We now describe a deterministic variant of Algorithm 1. The only thing we change is that, instead of selecting a random member of the set \mathcal{C} of candidate regions, we select an arbitrary element of this set. Here we assume that \mathcal{C} is represented in some deterministic way that we make no further assumptions about. For example, if the elements in \mathcal{D} are unit disks we might store them as a sorted list by the x -coordinate of their center points. Algorithm 2 shows the pseudocode for the changed procedure.

Algorithm 2 Deterministic online tracking algorithm.

Procedure $\text{pick_sensor}(\tau)$:

1. let C be the first element in \mathcal{C}
 2. append (τ, C) to S
-

This strategy may seem rather naïve, and indeed produces a competitive ratio that is exponentially larger than that of the randomized strategy of the previous section. But we will see in Section 5 that this is unavoidable, even for the specific case of unit disks.

Lemma 2. *Algorithm 2 produces a valid solution of length $O(k\Delta)$.*

Proof. Since the change in the algorithm does not influence the validity of the solution, the correctness of the algorithm follows directly from Lemma 1. The length of a solution is clearly no more than $k\Delta$, since there are k steps and at each step there are at most Δ disks in \mathcal{D}_i . \square

As before, combining Observation 2 and Lemma 2, we see that Algorithm 2 has a competitive ratio of $O(\Delta)$.

4.3 Deterministic Tracking in One Dimension

In the 1-dimensional case, a better deterministic algorithm is possible. In this case, the regions of \mathcal{D} can only be connected intervals, due to our assumptions that they are closed connected subsets of \mathbb{R} .



Fig. 5. A set of 8 intervals covering the current location of the entity (blue dot). A good interval is highlighted; this interval has $\ell_i = 3 \leq 8/2$ and $r_i = 2 \leq 8/2$.

Now, when we want to pick a new sensor, we have to choose between $c = |\mathcal{C}|$ intervals that all contain the current position of the entity. For each interval C_i , let ℓ_i be the number of intervals in $\mathcal{C} \setminus \{C_i\}$ that contain the left endpoint of C_i , and let r_i be the number of intervals in $\mathcal{C} \setminus \{C_i\}$ that contain the right endpoint of C_i . We say that an interval C_i is *good* if $\max(\ell_i, r_i) \leq c/2$. Our deterministic algorithm simply chooses a good sensor at each step. Figure 5 illustrates this.

Algorithm 3 Deterministic online tracking algorithm for $d = 1$.

Procedure `pick_sensor`(τ):

1. let L be the sequence of left end points of the intervals in \mathcal{C} , sorted from left to right
 2. let R be the sequence of right end points of the intervals in \mathcal{C} , sorted from right to left
 3. for each $C \in \mathcal{C}$, let i_C be the highest index of C in either L or R
 4. let C^* be the sensor that has the lowest i_C
 5. append (τ, C^*) to S
-

The new algorithm is described in Algorithm 3. As with our deterministic algorithm in higher dimensions, the only change from Algorithm 1 is the implementation of the `pick_sensor` procedure.

Lemma 3. *Algorithm 3 produces a valid solution of length $O(k \log \Delta)$.*

Proof. There always exists a good interval, by the pigeonhole principle, because there are at most $(c-1)/2$ intervals that are not good due to ℓ_i being too high and at most $(c-1)/2$ intervals that are not good due to r_i being too high. Therefore the algorithm always succeeds in finding a good interval to choose. As in the previous section, the change in the algorithm does not influence the validity of the solution, so the correctness of the algorithm follows directly from Lemma 1.

Each time Algorithm 3 performs a handover, it must be the case the trajectory has just crossed either the left endpoint or the right endpoint of the interval it most recently selected. Therefore, within the time interval from τ_i to τ_{i+1} , the number of intervals in \mathcal{C} goes down by at least a factor of two at each handover, and it begins as at most Δ . Therefore, the number of handovers within this interval is at most $\log_2 \Delta$ and the total cost of the solution is at most $k \log_2 \Delta$. \square

Combining Observation 2 and Lemma 3, we conclude that Algorithm 3 also has a competitive ratio of $O(\log \Delta)$.

4.4 Summary of Algorithms

Our input assumptions ensure that any trajectory can be transformed in polynomial time into a sequence of events: trivially, for each piece in the piecewise description of the trajectory, we can determine the events involving that piece in time $O(n)$ (where $n = |\mathcal{D}|$) and sort them in time $O(n \log n)$.

Once this sequence is known, it is straightforward to maintain both the set of regions containing the current endpoint of the trajectory, and the set \mathcal{C} of candidate regions, in constant time per event. Additionally, each event may cause our algorithms to select a new region, which may in each case be performed given the

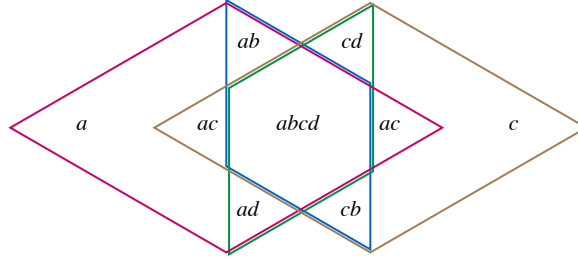


Fig. 6. Four similar rhombi form a set of regions for which no stateless algorithm can be competitive.

set \mathcal{C} in time $O(|\mathcal{C}|) = O(\Delta)$. Therefore, if there are m events in the sequence, the running time of our algorithms (once the event sequence is known) is at most $O(m\Delta)$.

Additionally, geometric data structures (such as those for point location among fat objects [19]) may be of use in more quickly finding the sequence of events, or for more quickly selecting a region from \mathcal{C} ; we have not carefully analyzed these possibilities, as our focus is primarily on the competitive ratio of our algorithms rather than on their running times.

We summarize these results in the following theorem:

Theorem 3. *Given a set \mathcal{D} of n connected regions in \mathbb{R}^d , and a trajectory T ,*

- *there is a randomized strategy for the online tracking problem that achieves a competitive ratio of $O(\log \Delta)$; and*
- *there are deterministic strategies for the online tracking problem that achieve a competitive ratio of $O(\log \Delta)$ when $d = 1$ or $O(\Delta)$ when $d > 1$.*

Each of these strategies may be implemented in polynomial time.

5 Lower Bounds

We now provide several lower bounds on the best competitive ratio that any deterministic or randomized algorithm can hope to achieve. Our lower bounds use only very simple regions in \mathcal{D} : similar rhombi, in one case, unit disks in \mathbb{R}^d in a second case, and unit intervals in \mathbb{R} in the third case. These bounds show that our algorithms are optimal, even with strong additional assumptions about the shapes of the regions.

5.1 Lower Bounds on Stateless Algorithms

An algorithm is *stateless* if the next sensor that covers the moving point, when it moves out of range of its current sensor, is a function only of its location and not of its previous state or its history of motion. Because they do not need to store and retrieve as much information, stateless algorithms provide a very enticing possibility for the solution of the online tracking problem, but as we show in this section, they cannot provide a competitive solution.

Theorem 4. *There exists a set \mathcal{D} of four similar rhombi in \mathbb{R}^2 , such that any stateless algorithm for the online tracking problem has unbounded competitive ratio.*

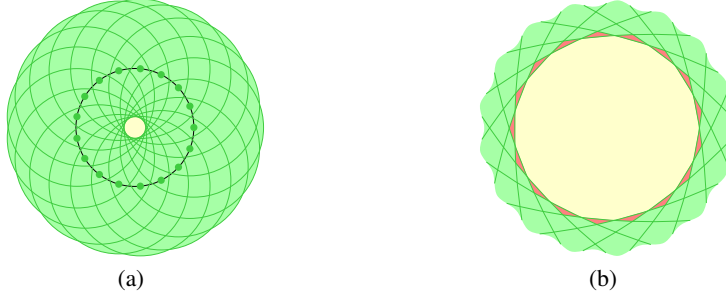


Fig. 7. (a) A set of Δ disks whose centers are equally spaced on a circle. (b) The heart of the construction, zoomed in. The yellow cell is inside all disks; the red cells are inside all but one disk.

Proof. The set \mathcal{D} is shown in Figure 6. It consists of four rhombi a , b , c , and d ; these rhombi partition the plane into regions (labeled in the figure by the rhombi containing them) such that the common intersection $abcd$ of the rhombi is directly adjacent to regions labeled ab , ac , ad , bc , and cd .

Let G be a graph that has the four rhombi as its vertices, and the five pairs ab , ac , ad , bc , and cd as its edges. Let A be a stateless algorithm for \mathcal{D} , and orient the edge xy of G from x to y if it is possible for algorithm A to choose region y when it performs a handover for a trajectory that moves from region $abcd$ to region xy . If different trajectories would cause A to choose either x or y , orient edge xy arbitrarily.

Because G has four vertices and five edges, by the pigeonhole principle there must be some vertex x with two outward-oriented edges xy and xz . There exists a trajectory T that repeatedly passes from region $abcd$ to xy , back to $abcd$, to xz , and back to $abcd$, such that on each repetition algorithm A performs two handovers, from z to y and back to z . However, the optimal strategy for trajectory T is to cover the entire trajectory with region x , performing no handovers. Therefore, algorithm A has unbounded competitive ratio. \square

5.2 Lower Bounds on Deterministic Algorithms

Next, we show that any deterministic algorithm in two or more dimensions must have a competitive ratio of Δ or larger, matching our deterministic upper bound and exponentially worse than our randomized upper bound. The lower bound construction consists of a set of Δ unit disks with their centers on a circle, all containing a common point (Figure 7). The idea is that if the trajectory starts at this common point, it can exit from any single disk, in particular, the one that a deterministic algorithm previously chose.

Theorem 5. *There exists a set \mathcal{D} of unit disks in \mathbb{R}^2 , such that any deterministic algorithm for the online tracking problem has competitive ratio at least $\Delta - 1$.*

Proof. Let \mathcal{D} be a set of Δ unit disks whose centers are equally spaced on a given circle C of slightly less than unit radius, as in Figure 7(a). Let the moving point to be tracked start at the point p_0 at the center of C , in the common interior of all disks. For each disk $D_i \in \mathcal{D}$, there exists a cell X_i in the arrangement that is interior to all disks in $\mathcal{D} \setminus \{D_i\}$, but outside D_i itself. Furthermore, this cell is directly adjacent to the center cell. See Figure 7(b) for an illustration.

Now, let A be any deterministic algorithm for the online tracking problem, and construct a sequence of updates to trajectory T as follows. Initially, T consists only of the single point p_0 . At each step, let algorithm A update its tracking sequence to cover the current trajectory, let D_i be the final region in the tracking sequence constructed by algorithm A , and then update the trajectory to include a path to X_i and back to the center.

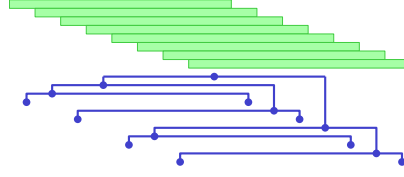


Fig. 8. A set of $\Delta = 8$ intervals, and a tree of 8 different trajectories in \mathbb{R}^1 (horizontal dimension).

Since X_i is not covered by D_i , algorithm A must increase the cost of its tracking sequence by at least one after every update. That is, $|S_A(T)| \geq |T|$. However, in the optimal tracking sequence, every $\Delta - 1$ consecutive updates can be covered by a single region D_i , so $S^*(T) \leq |T|/(\Delta - 1)$. Therefore, the competitive ratio of A is at least $\Delta - 1$. \square

This construction generalizes to any $d > 2$.

5.3 Lower Bounds on Randomized Algorithms

The above lower bound construction uses the fact that the algorithm to solve the problem is deterministic: an adversary constructs a tracking sequence by reacting to each decision made by the algorithm. For a randomized algorithm, this is not allowed. Instead, the adversary must select an entire input sequence, knowing the algorithm but not knowing the random choices to be made by the algorithm. Once this selection is made, we compare the quality of the solution produced by the randomized algorithm to the optimal solution. By Yao's principle [8, 25], finding a randomized lower bound in this model is equivalent to finding a random distribution R on the set of possible update sequences such that, for every possible deterministic algorithm A , the expected value of the competitive ratio of A on a sequence from R is high.

Our lower bound construction consists of Δ unit intervals that contain a common point, and a tree of Δ different possible paths for the moving object to take, each of which leaves the intervals in a different ordering, in a binary tree-like fashion. Half of the trajectories start by going to the left until they are outside the right half of the intervals, the others start towards the right until they are outside the left half of the intervals, and this recurses, as shown in Figure 8.

More formally, let us assume for simplicity that Δ is a power of 2. Let \mathcal{D} be a set of Δ distinct unit intervals in \mathbb{R} , containing a common point p_0 . For any $k \in [1, \Delta]$ we define point p_k to be a point outside the leftmost k intervals but in the interior of the rest, and p_{-k} to be a point outside the rightmost k intervals but in the interior of the rest.

Now, for each $j \in [1, \Delta]$, we construct a trajectory T_j with $h = \log \Delta$ steps, as follows. We define an index $\xi(j, i)$ for all $j \in [1, \Delta]$ and all $i \in [1, h]$ such that trajectory T_j is at point $p_{\xi(j, i)}$ at step i . At step 0, all trajectories start at $\xi(j, 0) = 0$. Then, at step i :

- all T_j with $j \bmod 2^{h-i} \leq 2^{h-i-1}$ move to the left to $\xi(j, i) = \min_{l < i} \xi(j, l) - 2^{h-i}$,
- all T_j with $j \bmod 2^{h-i} > 2^{h-i-1}$ move to the right to $\xi(j, i) = \max_{l < i} \xi(j, l) + 2^{h-i}$.

Figure 8 shows \mathcal{T} be the resulting set of these Δ trajectories in a tree representation.

Theorem 6. *There exists a set \mathcal{D} of unit intervals in \mathbb{R} , for which any randomized algorithm to solve the online tracking problem has competitive ratio $\Omega(\log \Delta)$.*

Proof. Let \mathcal{D} and the set of trajectories \mathcal{T} be as described above. Let R be a probability distribution over the set of all possible trajectories that has a probability of $1/\Delta$ to be any element of \mathcal{T} , and a probability of 0 elsewhere.

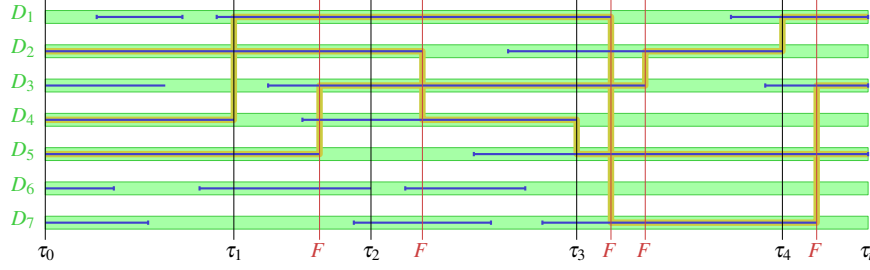


Fig. 9. Illustration of the two type of event times when covering the entity with $c = 3$ sensors at the same time. The yellow paths indicate the optimal greedy solution. The times marked with τ_i are the times at which we start a new step in the algorithm. The times marked with F are times when a fixed interval ends. Note that at time τ_2 none of the sensors in the greedy solution changes its interval.

Now, let A be any deterministic algorithm for the online tracking problem. At each level of the tree, each region D_i that algorithm A might have selected as the final region in its tracking sequence fails to cover one of the two points that the moving point could move to next, and each of these points is selected with probability $1/2$, so algorithm A must extend its tracking sequence with probability $1/2$, and its expected cost on that level is $1/2$. The number of levels is $\log_2 \Delta$, so the total expected cost of algorithm A is $1 + \frac{1}{2} \log_2 \Delta$, whereas the optimal cost on the same trajectory is 1. Therefore the competitive ratio of algorithm A on a random trajectory with distribution R is at least $1 + \frac{1}{2} \log_2 \Delta$.

It follows by Yao's principle that the same value $1 + \frac{1}{2} \log_2 \Delta$ is also a lower bound on the competitive ratio of any randomized online tracking algorithm. \square

Although the trajectories formed in this proof are short relative to the size of \mathcal{D} , this is not an essential feature of the proof: by concatenating multiple trajectories drawn from the same distribution, we can find a random distribution on arbitrarily long trajectories leading to the same $1 + \frac{1}{2} \log_2 \Delta$ lower bound. This construction generalizes to unit balls in any dimension $d > 1$ as well.

6 Trilateration

We now extend our online algorithms to the scenario where the moving entity needs to be covered by c different sensors at all times. Obviously, this means we need to strengthen our assumptions on \mathcal{D} and T slightly: every point of $T[0, \infty)$ should be inside at least c regions of \mathcal{D} for this to be possible.

We analyze the online version of the problem, and provide the competitive ratio as a function of Δ and c . As c tends to Δ , we may expect the competitive ratio to improve, since in the extreme case we simply need to use all available sensors and have no choice in the matter. Indeed, we show that a randomized algorithm exists that has competitive ratio $O(\log(\Delta - c))$, and that again no better ratio is possible in this case.

6.1 Randomized Algorithm

The randomized algorithm with expected competitive ratio of $\log \Delta$ for the simple version of the online tracking problem can be extended to the case in which we want to track the entity with c sensors. The reason is that the greedy algorithm still works for the offline problem, as we proved in Theorem 1. Interestingly, the competitive ratio gets better as c increases: the algorithm can be adapted to achieve a competitive ratio of $O(\log(\Delta - c))$, so when the required coverage is close to the ply this gives a constant competitive ratio. The description gets slightly more complicated. We will now prove this theorem.

Theorem 7. *There exists a randomized algorithm that solves the trilateration problem with a competitive ratio of $O(\log(\Delta - c))$.*

First, we describe the altered algorithm. Again, we construct a sequence of time stamps τ_i with corresponding location of the entity p_i and set of disks \mathcal{D}_i that contain p_i . Obviously, we need to assume $|\mathcal{D}_i| \geq c$. Now, as before, we maintain a set of candidates \mathcal{C} that is initialized to \mathcal{D}_0 , and whenever the trajectory exits a sensor region, we remove that region from \mathcal{C} and replace it with a new random element from \mathcal{C} that is not currently used by another sensor. If there are not enough regions left in \mathcal{C} (that is, if $|\mathcal{C}| = c - 1$ after removing the current sensor), we mark the current time as τ_{i+1} , and compute a new set of candidates (note that the $c - 1$ remaining old candidates will also be in the new set of candidates). This proceeds until we mark τ_k as the end of time.

Now, to analyze the performance of this algorithm, we will define an additional class of events. Just before we reach a new time τ_i , we mark all c sensors currently in use (which must correspond to the current candidate set \mathcal{C}) as *fixed*. Whenever a fixed sensor reaches the end of its interval, we say we have an F -event, and after the F -event we stop considering this sensor as being fixed. In particular, this means that each new τ_i corresponds to an F -event, but there could be more F -events. Figure 9 shows the time stamps and F -events for a small example. If we define F_i to be the number of F -events between τ_i and τ_{i+1} , including the former and excluding the latter, then $1 \leq F_i \leq c$ for all i . We also write $m = \sum_{i=0}^k F_i$.

Now we will show that any solution needs $\Omega(m)$ handovers, and that our algorithm produces a solution with an expected number of $O(m \log(\Delta - c))$ handovers.

Lemma 4. *The optimal solution to the tracking problem uses $\Theta(m)$ handovers.*

Proof. By Theorem 1 the greedy approach yields an optimal solution, so we only need to bound the number of handovers in that solution.

First, we argue that each handover in the greedy solution coincides with an F -event. At the start, the greedy algorithm assigns the c longest available intervals, which all become fixed before they run out. By construction, for any i there must be a set of exactly c sensors that cover the whole interval $[\tau_i, \tau_{i+1}]$. Whenever an F -event occurs in this interval and a new sensor has to be assigned, the greedy approach will pick the longest available one, and there must be at least one available interval that extends beyond τ_{i+1} , which means it will also become fixed before it runs out.

Not all F -events need to be used by the greedy solution however (as can be seen in Figure 9), so we will now prove that at least half of them are, by charging the unused events to used ones. Suppose an F -event is not used, and assume it occurs in the interval $[\tau_i, \tau_{i+1})$. The fact that the sensor became fixed means its interval started already at or before time τ_{i-1} . Suppose there are f unused F -events in the time interval $[\tau_i, \tau_{i+1})$. Since there are only c sensors that completely cover the time interval $[\tau_{i-1}, \tau_i]$, and f of them are not used, there must have been f handovers in the time interval $[\tau_{i-1}, \tau_i]$, which by the previous argument occurred during F -events. \square

Lemma 5. *Our randomized solution to the online tracking problem produces a valid solution of expected length $O(m \log(\Delta - c))$.*

Proof. There are F_i F -events during the time interval $[\tau_i, \tau_{i+1})$. When each of these events occurs, we pick a random element from the set of candidates \mathcal{D}_i , that is larger than any other random element we chose so far. We know that $c \leq |\mathcal{D}_i| \leq \Delta$. We also know that $c - F_i$ sensors stay on their intervals during this time, so the set of free candidates is in fact at most $\Delta - c + F_i$. This results in an expected number of $F_i + \log((\Delta - c + F_i)/(F_i))$ changes (because we first take F_i random elements, and then a random increasing sequence that starts larger than any of these numbers).

The expected total number of handovers is $\sum_{i=0}^k (F_i + \log((\Delta - c + F_i)/(F_i)))$, which in the worst case (occurring when $F_i = 1$ for all i) comes down to $m \log(\Delta - c)$. \square

These two lemmas together imply the competitive ratio we claimed.

6.2 Lower Bound for the Randomized Case

We can extend the construction of Section 5.3 to the current situation, although we now require the regions of \mathcal{D} to be intervals of two different lengths. Note that when $d > 1$, this is no real restriction since we can always position a set of unit disks in such a way that their intersections with a given line form intervals of two different lengths.

We define a set of intervals \mathcal{D} , consisting of $\Delta - c + 1$ unit intervals that form the construction depicted in Figure 8, and additionally $c - 1$ intervals of length 2 that cover all $\Delta - c + 1$ unit intervals.

Theorem 8. *There exists a set \mathcal{D} of intervals in \mathbb{R} of two different lengths, for which any randomized algorithm to solve the online tracking problem has competitive ratio $\Omega(\log(\Delta - c))$.*

Proof. A solution of c disjoint tracking sequences must at any time use at least one of the $\Delta - c + 1$ unit intervals, since there are only $c - 1$ other intervals. We may assume that there is one single sequence that only uses unit intervals, since otherwise we could swap pieces of the sequences. But by Theorem 6, no algorithm can produce such a sequence with less than $\Omega(\log h)$ expected handovers, if there are h unit intervals in the construction. Since in our case $h = \Delta - c + 1$, the result follows. \square

Our other lower bounds can also be extended just as easily.

7 Conclusions

We studied the online problem of tracking a moving entity among sensors with a minimal number of handovers, combining the kinetic data and online algorithms paradigms. We provided several algorithms with optimal competitive ratios. Interestingly, randomized strategies are able to provably perform significantly better than deterministic strategies, and arbitrarily better than stateless strategies (which form a very natural and attractive class of algorithms in our application).

We are able to track multiple entities using the same algorithms, by simply treating them independently. As a future direction of research, it would be interesting to study the situation where each sensor has a maximum capacity C , and cannot track more than C different entities at the same time. Another possible direction of research is to analyze and optimize the running times of our strategies for particular classes of region shapes or trajectories, something we have made no attempt at.

References

1. P. K. Agarwal, J. Erickson, and L. J. Guibas. Kinetic BSPs for intersecting segments and disjoint triangles. In *Proc. 9th ACM-SIAM Sympos. Discrete Algorithms*, pages 107–116, 1998.
2. A. Alaybeyoglu, K. Erciyes, A. Kantarci, and O. Dagdeviren. Tracking Fast Moving Targets in Wireless Sensor Networks. *IETE Technical Review*, 27(1):46–53, 2010.
3. N. Alon and S. Smorodinsky. Conflict-free colorings of shallow discs. In *Proc. 22nd Symp. on Computational Geometry (SoCG)*, pages 41–43, New York, NY, USA, 2006. ACM.
4. J. Basch, L. J. Guibas, C. Silverstein, and L. Zhang. A practical evaluation of kinetic data structures. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 388–390, 1997.
5. A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.
6. Q. Cao, T. Yan, J. Stankovic, and T. Abdelzaher. Analysis of Target Detection Performance for Wireless Sensor Networks. In V. K. Prasanna, S. Iyengar, P. G. Spirakis, and M. Welsh, editors, *Distributed Computing in Sensor Systems (DCOSS)*, volume 3560 of *LNCS*, pages 276–292. Springer, 2005.
7. M. Cho, D. Mount, and E. Park. Maintaining Nets and Net Trees under Incremental Motion. In Y. Dong, D.-Z. Du, and O. Ibarra, editors, *Int. Symp. on Algorithms and Computation (ISAAC)*, volume 5878 of *LNCS*, pages 1134–1143. Springer, 2009.

8. M. Chrobak, L. L. Larmore, C. Lund, and N. Reingold. A better lower bound on the competitive ratio of the randomized 2-server problem. *Inform. Process. Lett.*, 63(2):79–83, 1997.
9. D. Eppstein and M. T. Goodrich. Studying (non-planar) road networks through an algorithmic lens. In *GIS '08: Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, pages 1–10, New York, NY, USA, 2008. ACM.
10. D. Eppstein, M. T. Goodrich, and L. Trott. Going off-road: transversal complexity in road networks. In *Proc. 17th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems (ACM GIS)*, pages 23–32, New York, NY, USA, 2009. ACM.
11. O. Ghica, G. Trajcevski, F. Zhou, R. Tamassia, and P. Scheuermann. Selecting Tracking Principals with Epoch Awareness. In *Proc. 18th ACM SIGSPATIAL Internat. Conf. on Advances in Geographic Information Systems (ACM GIS)*, 2010.
12. L. Guibas, J. Hershberger, S. Suri, and L. Zhang. Kinetic Connectivity for Unit Disks. *Discrete Comput. Geom.*, 25(4):591–610, 2001.
13. L. J. Guibas. Kinetic data structures — a state of the art report. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, *Proc. Workshop Algorithmic Found. Robot.*, pages 191–209. A. K. Peters, Wellesley, MA, 1998.
14. G. He and J. Hou. Tracking targets with quality in wireless sensor networks. In *13th IEEE Conf. on Network Protocols (ICNP)*, pages 1–12, 2005.
15. M. Hellebrandt, R. Mathar, and M. Scheibenbogen. Estimating position and velocity of mobiles in a cellular radio network. *IEEE Transactions on Vehicular Technology*, 46(1):65–71, 2002.
16. S. Irani and S. Seiden. Randomized algorithms for metrical task systems. *Theor. Comput. Sci.*, 194(1–2):163–182, 1998.
17. G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44:1–29, 1997.
18. D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. A computational framework for incremental motion. In *Proc. 20th Symp. on Computational Geometry (SoCG)*, pages 200–209, New York, NY, USA, 2004. ACM.
19. M. Overmars and F. van der Stappen. Range Searching and Point Location among Fat Objects. *J. Algorithms*, 21(3):629–656, 1996.
20. S. Pattem, S. Poduri, and B. Krishnamachari. Energy-Quality Tradeoffs for Target Tracking in Wireless Sensor Networks. In F. Zhao and L. Guibas, editors, *Information Processing in Sensor Networks*, volume 2634 of *LNCs*, pages 553–553. Springer, 2003.
21. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28:202–208, 1985.
22. S. Tekinay and B. Jabbari. Handover and channel assignment in mobile cellular networks. *IEEE Communications Magazine*, 29(11):42–46, 1991.
23. E. J. van Leeuwen. Better Approximation Schemes for Disk Graphs. In L. Arge and R. Freivalds, editors, *Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 4059 of *LNCs*, pages 316–327. Springer, 2006.
24. Z. Yang and Y. Liu. Quality of Trilateration: Confidence-Based Iterative Localization. *IEEE Trans. on Parallel and Distributed Systems*, 21(5):631–640, 2010.
25. A. Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.
26. K. Yi and Q. Zhang. Multi-dimensional online tracking. In *Proc. of the 20th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1098–1107. SIAM, 2009.
27. F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration. *IEEE Signal Processing Magazine*, 19(2):61–72, 2002.