Multi-Dimensional Online Tracking[*]

Ke Yi   Qin Zhang

Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Hong Kong, China
{yike,qinzhang}@cse.ust.hk

Abstract

We propose and study a new class of online problems, which we call *online tracking*. Suppose an *observer*, say Alice, observes a multi-valued function $f : \mathbb{Z}^+ \to \mathbb{Z}^d$ over time in an online fashion, i.e., she only sees $f(t)$ for $t \leq t_{now}$ where t_{now} is the current time. She would like to keep a *tracker*, say Bob, informed of the current value of f at all times. Under this setting, Alice could send new values of f to Bob from time to time, so that the current value of f is always within a distance of Δ to the last value received by Bob. We give competitive online algorithms whose communication costs are compared with the optimal offline algorithm that knows the entire f in advance. We also consider variations of the problem where Alice is allowed to send "predictions" to Bob, to further reduce communication for well-behaved functions. These online tracking problems have a variety of application ranging from sensor monitoring, location-based services, to publish/subscribe systems.

1 Introduction

Let Alice be an *observer* who observes a function $f(t)$ in an online fashion over time. She would like to keep a *tracker*, Bob, informed of the current function value within some predefined error. What is the best strategy that Alice could adopt so that the total communication is minimized? This is the general problem that we study in this paper. For concreteness, consider the simplest case where the function takes integer values at each time step[1] $f : \mathbb{Z}^+ \to \mathbb{Z}$, and we require an absolute error of at most Δ. The natural solution to the problem is to first communicate $f(0)$ to Bob; then every time $f(t)$ has changed by more than Δ since the last communication, Alice updates Bob with the current $f(t)$. Interestingly, this natural algorithm for this seemingly simple problem has an unbounded competitive ratio compared with the optimal. Consider the case where $f(t)$ starts at $f(0) = 0$ and then oscillates between 0 and 2Δ. Then this algorithm will communicate an infinite number of times while the optimal solution only needs two messages: $f(0) = 0$ and $f(t') = \Delta$ where t' is the first time f reaches Δ.

The example above shows that even in its simplest instantiation, the *online tracking* problem will require some nontrivial solutions. Indeed, in Section 2 we give an $O(\log \Delta)$-competitive algorithm for the above problem. The competitive ratio is also tight. Formally, we define the general problem considered in this paper as follows. Let $f : \mathbb{Z}^+ \to \mathbb{Z}^d$ be a function observed by Alice over time. At the current time t_{now}, Alice only sees all function values for $f(t), t \leq t_{now}$. Then she decides if she wants to communicate to Bob, and if so, a pair $(t_{now}, g(t_{now}))$ to be sent. Note that $g(t_{now})$ is not necessarily equal to $f(t_{now})$. The only constraint is that at any t_{now}, if Alice does not communicate, then we must have $\|f(t_{now}) - g(t_{last})\| \leq \Delta$, where t_{last} is the last time Bob got informed, and $\Delta > 0$ is some predefined error parameter. Unless stated otherwise, $\| \cdot \|$ denotes the ℓ_2 norm throughout the paper. We are mostly interested in the total communication (also referred to as the *cost*) incurred throughout time, i.e., the total number of messages sent by the algorithm, and will analyze the performance of an algorithm in terms of its competitive ratio, i.e., the worst-case ratio between the cost of the online algorithm and the cost of the best offline algorithm that knows the entire f in advance.

Motivations. Online tracking problems naturally arise in a variety of applications whenever the observer and the tracker are separate entities and the communication between them is expensive. For example, wireless sensors [23] are now widely deployed to collect many different kinds of measurements in the physical world, e.g., temperature, humidity, and oxygen level. These small and

[*]Ke Yi is supported in part by Hong Kong Direct Allocation Grant (DAG07/08). Qin Zhang is supported by Hong Kong CERG Grant 613507.

[1]We use $\mathbb{Z}^+$ to denote the domain of all non-negative integers in this paper.

cheap devices can be easily deployed, but face strict power constraints. It is often costly or even impossible to replace them, so it is essential to develop energy-efficient algorithms for their prolonged functioning. It is well known that among all the factors, wireless transmission of data is the biggest source of battery drain [19]. Therefore, it is very important to minimize the amount of communication back to the tracker, while guaranteeing that the tracker always maintains an approximate measurement monitored by the sensor. This directly corresponds to the one-dimensional version of our problem mentioned at the beginning of the paper.

Our study is also motivated by the increasing popularity of *location-based services* [21]. Nowadays, many mobile devices, such as cell phones and PDAs, are equipped with GPS. It is common for the service provider to keep track of the user's (approximate) location, and provide many location-based services, for instance finding the nearest business (ATMs or restaurants), receiving traffic alerts, etc. This case corresponds to the two-dimensional version of our problem. Here approximation is often necessary not just for reducing communication, but also due to privacy concerns [2]. For carriers, location-based services provide added value by enabling dynamic resource tracking (e.g., tracking taxis and service people). Similar to sensors, power consumption is the biggest concern for these mobile devices, and both the user the service provider have incentives to reduce communication while being able to track the locations dynamically.

Finally, our problem also finds applications in the so called *publish/subscribe* systems [4, 11]. Traditionally, users *poll* data from service providers for information; but this has been considered to be very communication-inefficient. In a pub/sub system, users register their queries at the server, and the server *pushes* updated results to the users according to their registered queries as new data arrives. Unlike the two applications above, here we have one observer (the server) and many trackers (the users). Although energy is not a concern here, bad decisions by the online tracking algorithm still have severe consequences, since the messages need to be forwarded to potentially a larger number of users, consuming a lot of network bandwidth. Depending on the nature of the query, the function being tracked could take values from a high-dimensional space. For instance, a set of items from a universe U corresponds to a $\{0, 1\}$-vector in $|U|$ dimensions.

Related work. Although our problem is easily stated and finds many applications, to the best of our knowledge it has not been studied before in the theory community. Some related models include online algorithms, communication complexity, data streams, and the distributed tracking model.

Our problem generally falls in the realm of online algorithms, and as with all online algorithms, we analyze the performance of our algorithms in terms of competitive ratios.

In communication complexity [22], Alice has x and Bob has y, and the goal is to compute some function $f(x, y)$ by communicating the minimum number of bits between them. There are two major differences between communication complexity and online tracking: First, in online tracking, only Alice sees the input, Bob just wants to keep track of it. Secondly, in communication complexity both inputs x and y are given in advance, and the goal is to study the worst-case communication between x and y; while in online tracking, the inputs arrive in an online fashion, and we focus on the competitive ratio. It is easy to see that the worst-case (total) communication bounds for our problems are mostly trivial.

In data streams [1], the inputs arrive online, and the goal is to track some function over the inputs received so far. In this aspect it is similar to our problem. However, the focus in streaming algorithms is to minimize the space used by the algorithm, not communication. The memory contents could change rapidly, so simply sending out the memory contents could lead to high communication costs.

In distributed tracking [6–9, 13, 17], the inputs are distributed among multiple *sites* and arrive online. There is a coordinator who wants to keep track of some function over the union of the inputs received by all sites up until t_{now}. So in some sense our problem is the special version of distributed tracking where there is only one site. However, most works in this area are heuristic-based with two only exceptions to the best of our knowledge. Cormode et al. [8] consider monotone functions and study worst-case costs. But when the function is not monotone, the worst-case bounds are trivial. In this paper, we allow functions to change arbitrarily and use competitive analysis to avoid meaningless worst-case bounds. Davis et al. [9] propose online algorithms for distributed tracking functions at multiple sites where site i is allowed an error of Δ_i, and the total error $\sum_i \Delta_i$ is fixed. However, they compare with the offline algorithm that can only send an updated value when the error Δ_i allocated to the site is reached, and the site can only send in the current value of the function, i.e., exactly what the naive algorithm that we described at the beginning is doing. As such, the problem is only meaningful for two or more sites since when there is only one site, the offline algorithm itself is already an online algorithm. In our problem, we compare with the offline algorithm that is allowed to send in *any* function value and at *any* time, as long as the error bound Δ is satisfied.

	$\beta = 1$		$\beta = 1 + \epsilon$	
problem	competitive ratio	running time	competitive ratio	running time
1-dim	$O(\log \Delta)$	$O(1)$	/	/
d-dim	$O(d^2 \log(d\Delta))$	$\text{poly}(d, \log \Delta)$	$O(d \log(d/\epsilon))$	$\text{poly}(d, \log(1/\epsilon))$
1-dim with prediction	$O(\log(\Delta T))$	$\text{poly}(\Delta, T)$	/	/

Table 1: Summary of results for online tracking. T is the length of the tracking period.

Finally, it should be noted that similar problems have been studied in the database community [10, 13, 15, 17]. However, all the techniques proposed there are based on heuristics with no theoretical guarantees.

Our results. In Section 2 we first give an $O(\log \Delta)$-competitive algorithm for tracking an integer-valued function. We show that the algorithm is optimal by proving a matching lower bound on the competitive ratio. Our lower bound argument also implies that any real-valued function cannot be tracked with a bounded competitive ratio. This justifies our study being confined with integer-valued functions. In Section 3 we extend our algorithms to d dimensions for arbitrary d. Here we consider the more general (α, β)-competitive algorithms. An online algorithm is (α, β)-*competitive* if its cost is $\alpha \cdot \texttt{OPT}$ while allowing an error of $\beta \cdot \Delta$, where $\texttt{OPT}$ is the cost of the optimal offline algorithm allowing error Δ. We first give a simple algorithm using the *Tukey median* of a set of points, and then propose improved algorithms based on *volume-cutting*, a technique used in many convex optimization algorithms. This results in algorithms with a competitive ratio of $O(d^2 \log(d\Delta))$ for $\beta = 1$ and $O(d \log(d/\epsilon))$ for $\beta = 1 + \epsilon$, respectively. The algorithms also have running times polynomial in d.

In Section 4 we further extend our model by considering tracking with predictions. More precisely, Alice tries to predict the future trend of the function f based on history, and then sends the "prediction" to Bob, for example a linearly increasing trend. If the actual function values do not deviate from the prediction by more than Δ, no communication is necessary. The previous tracking problem can be seen as a special case of this more general framework, in which we always "predict" $f(t)$ to be $g(t_{last})$. In general, we could use a family $\mathcal{F}$ of prediction functions (e.g., linear functions), which could greatly reduce the total communication when f can be approximated well by a small number of functions in $\mathcal{F}$ (note that the offline algorithm also uses $\mathcal{F}$ to approximate f). In this paper we only consider the most natural case of linear functions, but we believe that our technique can be extended to more general prediction functions (e.g. polynomial functions with bounded degrees). Our results are summarized in Table 1.

Finally, we comment that our study in this paper focuses only on the ℓ_2 metric; the online tracking problem could in general be posed in any metric space, which could potentially lead to other interesting techniques and results.

2 Online Tracking in One Dimension

In this section, we consider the online tracking problem for functions in the form of $f : \mathbb{Z}^+ \to \mathbb{Z}$. The algorithm for the one-dimensional case mainly serves an illustration purpose, which lays down the general framework for the more advanced algorithms in higher dimensions. For simplicity we assume for now that Δ is an integer; the assumption will be removed in later sections.

Algorithm 1: One round of 1D tracking

```
1 let $S = [f(t_{now}) - \Delta, f(t_{now}) + \Delta] \cap \mathbb{Z}$;
2 while $S \neq \emptyset$ do
3     let $g(t_{now})$ be the median of $S$;
4     send $g(t_{now})$ to Bob;
5     wait until $\|f(t_{now}) - g(t_{last})\| > \Delta$;
6     $S \leftarrow S \cap [f(t_{now}) - \Delta, f(t_{now}) + \Delta]$;
```

An $O(\log \Delta)$-competitive algorithm. Let $\texttt{OPT}$ be the cost of the optimal offline algorithm. The basic idea of the algorithm actually originates from the motivating example at the beginning of the paper: When f oscillates within a range of 2Δ, then $\texttt{OPT}$ is constant. Thus, our algorithm tries to guess what value the optimal algorithm has sent using a binary search. Our algorithm proceeds in rounds, and the procedure for each round is outlined in Algorithm 1.

Algorithm 1 is correct since at any t_{now}, if $f(t)$ deviates more than Δ from $g(t_{last})$, we always update S so that all elements in S are within Δ of $f(t_{now})$. It is also easy to see that Algorithm 1 can be implemented in $O(1)$ time per time step. Below we show that its competitive ratio is $O(\log \Delta)$.

We will proceed by showing that in each round, the offline optimal algorithm $\mathcal{A}_{\texttt{OPT}}$ must send at least one message, while Algorithm 1 sends $O(\log \Delta)$ messages, which will lead to the claimed competitive ratio. The latter simply follows from the fact the cardinality of S reduces by at least half in each iteration in the while loop, so we only argue for the former. For convenience, we define a round to include its starting time (when S

is initialized) and ending time (when $S = \emptyset$). Thus, a message sent at a joint point will be counted twice, but that will not affect the competitive ratio by more than a factor of 2. Suppose the last function value sent by $\mathcal{A}_{\texttt{OPT}}$ in the previous round is y. Note that if $\mathcal{A}_{\texttt{OPT}}$ has not sent any message by t_{now}, then we must have $y \in S$ at that time, since S is a superset of $\bigcap_t [f(t) - \Delta, f(t) + \Delta] \cap \mathbb{Z}$, where the intersection is taken over all t up to t_{now} in this round. In the end, $S = \emptyset$, so $\mathcal{A}_{\texttt{OPT}}$ must have sent a new function value other than y.

THEOREM 2.1. *There is an $O(\log \Delta)$-competitive online algorithm to track any function $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}$.*

Lower bound on the competitive ratio. We now show that the $O(\log \Delta)$ competitive ratio is optimal. We will construct an adversary under which any deterministic online algorithm $\mathcal{A}_{\texttt{SOL}}$ has to send at least $\Omega(\log \Delta \cdot \texttt{OPT})$ messages, while the optimal offline algorithm $\mathcal{A}_{\texttt{OPT}}$ only needs to send $\texttt{OPT}$ messages.

The adversary (call her Carole) also divides the whole tracking period into rounds. We will show that in each round, Carole could manipulate the value of f so that $\mathcal{A}_{\texttt{SOL}}$ has to communicate $\Omega(\log \Delta)$ times, while $\mathcal{A}_{\texttt{OPT}}$ just needs one message. During each round, Carole maintains a set S of possible values so that for any $y \in S$, if $\mathcal{A}_{\texttt{OPT}}$ communicates y at the beginning of this round, it does not need any further communication in this round. The round terminates when S contains less than 3 elements. More precisely, S is initialized to $[y - \Delta, y + \Delta] \cap \mathbb{Z}$ where y is some function value at a distance of at least $2\Delta + 1$ from any function value used in the previous round; as time goes on, S is maintained as $\bigcap_t [f(t) - \Delta, f(t) + \Delta] \cap \mathbb{Z}$, where the intersection is taken over all time up to t_{now}. Carole uses the following strategy to change the value of f. If $\mathcal{A}_{\texttt{SOL}}$ announces some function value greater than the median of S, decrease f until $\mathcal{A}_{\texttt{SOL}}$ sends out the next message; otherwise increase f until $\mathcal{A}_{\texttt{SOL}}$ sends out the next message. Let n_i be the number of elements left in S after the i-th triggering of $\mathcal{A}_{\texttt{SOL}}$, and initially, $n_0 = 2\Delta + 1$. It is not difficult to see that $n_{i+1} \geq \lceil (n_i - 3)/2 \rceil$, so it takes $\Omega(\log \Delta)$ iterations for $|S|$ to be a constant. When S contains less than 3 elements, Carole terminates the round and starts a new one. By the definition of S, $\mathcal{A}_{\texttt{OPT}}$ could send an element in S at the beginning of the round, which is a valid approximation for all function values in this round.

THEOREM 2.2. *To track a function $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}$, any online algorithm has to send $\Omega(\log \Delta \cdot \texttt{OPT})$ messages in the worst case, where $\texttt{OPT}$ is the number of messages needed by the optimal offline algorithm.*

Remark. The argument above also implies that, if f takes values from the domain of reals (or any dense set), the competitive ratio is unbounded, since S always contains infinitely many elements.

3 Online Tracking in d Dimensions

In this section we extend our algorithm to higher dimensions, i.e., tracking functions $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}^d$ for arbitrary d. From now on we will consider the more general (α, β)-competitive algorithms. Our algorithm actually follows the same framework as in the one-dimensional case. We still divide the whole tracking period into rounds, and show that $\mathcal{A}_{\texttt{OPT}}$ must communicate once in each round, while our algorithm communicates at most, say, k times, and then the competitive ratio would be bounded by k. The algorithm for each round is also similar to Algorithm 1. At the beginning of each round (say at time $t = t_{start}$), we initialize a set $S = S_0$ containing all the possible points that might be sent by $\mathcal{A}_{\texttt{OPT}}$ in its last communication. In each iteration in the while loop, we first pick a "median" from S and send it to Bob. When f deviates from $g(t_{last})$ by more than $\beta\Delta$, we cut S as $S \leftarrow S \cap \texttt{Ball}(f(t_{now}), \Delta)$ where $\texttt{Ball}(p, r)$ represents the closed ball centered at p with radius r in $\mathbb{R}^d$. This way, S is always a superset of $S_0 \cap \Big(\bigcap_{t_{start} \leq t \leq t_{now}} \texttt{Ball}(f(t), \Delta) \Big)$. When S becomes empty, we can terminate the round, knowing that $\mathcal{A}_{\texttt{OPT}}$ must have sent a new message. Thus, the only remaining issues are how to construct S_0 and how to choose the "median" so that S will become empty after a small number of cuts.

Note that for $\beta = 2$, the problem is trivial since $S_0 \subset \texttt{Ball}(f(t_{start}), \Delta)$. The algorithm simply needs to send $g(t_{start}) = f(t_{start})$, and then the first cut will use a ball centered at a distance of more than 2Δ away from $f(t_{start})$. So by the triangle inequality the round always terminates after just one cut, yielding an $(O(1), 2)$-competitive algorithm. Thus in the remaining of the paper, we are only interested in $\beta = 1$ or $\beta = 1 + \epsilon$ for any small $\epsilon > 0$.

3.1 Algorithms by Tukey medians

In this section we consider the case $\beta = 1$. We start by fixing the set S_0. Let C_l $(2 \leq l \leq d + 1)$ be the collection of centers of the smallest enclosing balls of every l points in $\texttt{Ball}(f(t_{start}), 2\Delta) \cap \mathbb{Z}^d$. At the beginning of the current round, we initialize S_0 to be $S_0 = C_2 \cup C_3 \ldots \cup C_{d+1}$. The following lemma justifies that S_0 is sufficient for our purpose.

LEMMA 3.1. *If S becomes empty at some time step, then the optimal offline algorithm must have communicated once in the current round.*

Proof: Suppose that the optimal offline algorithm $\mathcal{A}_{\texttt{OPT}}$ does not send any message in the current round when S

becomes empty. Let s be the point sent by $\mathcal{A}_{\texttt{OPT}}$ in its last communication and $q_1, q_2, \ldots, q_m$ be all the distinct points taken by the function f in the current round. It is easy to see that if $\mathcal{A}_{\texttt{OPT}}$ keeps silent in the current round, we have $\|s - q_i\| \le \Delta$ for all $1 \le i \le m$. If $m = 1$, then $\mathcal{A}_{\texttt{SOL}}$ just communicates at most once, so we assume that $m \ge 2$.

Let B be the smallest enclosing ball with center o containing all the q_i $(1 \le i \le m)$. By definition, we have $o \in S_0$. Since $\|s - q_i\| \le \Delta$ for all $1 \le i \le m$, and B is the smallest enclosing ball containing all q_i $(1 \le i \le m)$ with center o, we have $\|o - q_i\| \le \Delta$ for all $1 \le i \le m$. Thus o must still survive at the current time step, which means that S is not empty, a contradiction. □

The rest of our task is to choose a good median so that the cardinality of S would decrease by some fraction after each triggering of communication. Before proceeding, we need the following concepts.

DEFINITION 1. (Location depth) *Let S be a set of points in $\mathbb{R}^d$. The* location depth *of a point $q \in \mathbb{R}^d$ with respect to P is the minimum number of points of S lying in a closed halfspace containing q.*

The following observation is a direct consequence of Helly's Theorem [16].

OBSERVATION 1. *Given a set S in $\mathbb{R}^d$, there always exists a point $q \in \mathbb{R}^d$ having location depth at least $|S|/(d+1)$ with respect to S. The point with maximum depth is usually called the* Tukey median.

The algorithm for the $\mathbb{R}^d$ case maintains rounds similarly as the one dimensional case. We just pick the Tukey median to send in each triggering of communication. Since whenever $\|f(t_{now}) - g(t_{now})\| > \Delta$, $\texttt{Ball}(f(t_{now}), \Delta)$ is strictly contained in a halfspace bounded by a hyperplane passing through $g(t_{last})$, the cardinality of S decreases by a factor of at least $1/(d+1)$. Thus, the algorithm sends $\log_{1+\frac{1}{d}} |S_0| = O(d \log |S_0|)$ messages in each round.

To put things together, notice that initially, the set S_0 contains at most

$$\sum_{l=0}^{d} \binom{(\lfloor 4\Delta \rfloor + 1)^d}{l+1} = O\left(d \left(\frac{e(\lfloor 4\Delta \rfloor + 1)^d}{d+1} \right)^{d+1} \right)$$

points, since $\left|\texttt{Ball}(v, 2\Delta) \cap \mathbb{Z}^d\right| \le (\lfloor 4\Delta \rfloor + 1)^d$. Therefore, we have

THEOREM 3.1. *There is an $O(d^3 \log \Delta)$-competitive online algorithm that tracks any function $f : \mathbb{Z}^+ \to \mathbb{Z}^d$.*

Running time. However, to find the Tukey median exactly requires S to be explicitly maintained, which has size exponential in d. Clarkson et al. [5] proposed fast algorithms to compute an approximate Tukey median (a point with location depth $\Omega(n/d^2)$) via random sampling, but it seems difficult to sample from S when S is only implicitly maintained. We get around this problem with a new approach presented in the next subsection, which also improves the competitive ratio by roughly a d factor.

3.2 Algorithms by volume-cutting

In this section, we first consider $\beta = 1 + \epsilon$, and then show that we can set ϵ small enough to obtain an algorithm for the $\beta = 1$ case. Before proceeding to the new algorithm, note that an $O(d^2 \log(d/\epsilon), 1+\epsilon)$-competitive algorithm can be obtained by slightly modifying the algorithm in the previous section. However, as discussed earlier, this algorithm has running time exponential in d. In this section we propose algorithms with polynomial running time (w.r.t. both d and Δ) and also improved competitive ratios. These new algorithms use a volume-cutting technique, which shares similar spirits as many convex optimization algorithms.

3.2.1 The case with $\beta = 1 + \epsilon$

DEFINITION 2. (Directional width) *For a set P of points in $\mathbb{R}^d$, and a direction $\mu \in \mathbb{S}^{d-1}$, the* directional width *of P in direction μ is $\omega_\mu(P) = \max_{p \in P} \langle \mu, p \rangle - \min_{p \in P} \langle \mu, p \rangle$, where $\langle \mu, p \rangle$ is the standard inner product.*

Algorithm 2: One round of d-dimensional tracking via volume-cutting

```
1 let $P = \texttt{Ball}(f(t_{now}), \beta\Delta)$;
2 while ($\omega_{\max}(P) \ge \epsilon\Delta$) do
3     let $g(t_{now})$ be the centroid of $P$;
4     send $g(t_{now})$ to Bob;
5     wait until $\|f(t_{now}) - g(t_{last})\| > \beta\Delta$;
6     $P \leftarrow P \cap \texttt{Ball}(f(t_{now}), \beta\Delta)$;
```

Let $\omega_{\max}(P), \omega_{\min}(P)$ be the maximum and minimum directional width of P, respectively. Our volume-cutting algorithm also proceeds in rounds, and the procedure for each round is outlined in Algorithm 2. There are two differences between Algorithm 1 and 2. First, we now do not maintain the set S, instead we maintain P as the intersection of a collection of balls. Note that P could be maintained efficiently since the number of intersecting balls is polynomial in d and $\log \Delta$ as we will show later. Second, instead of sending the median of $P \cap S$, we send the centroid of P to Bob. The correctness of the algorithm is obvious since any point in P is within a distance of $\beta\Delta$ to $f(t_{now})$. As for the competitive ratio, it is easy to see that P always contains S. Thus when P contains no point

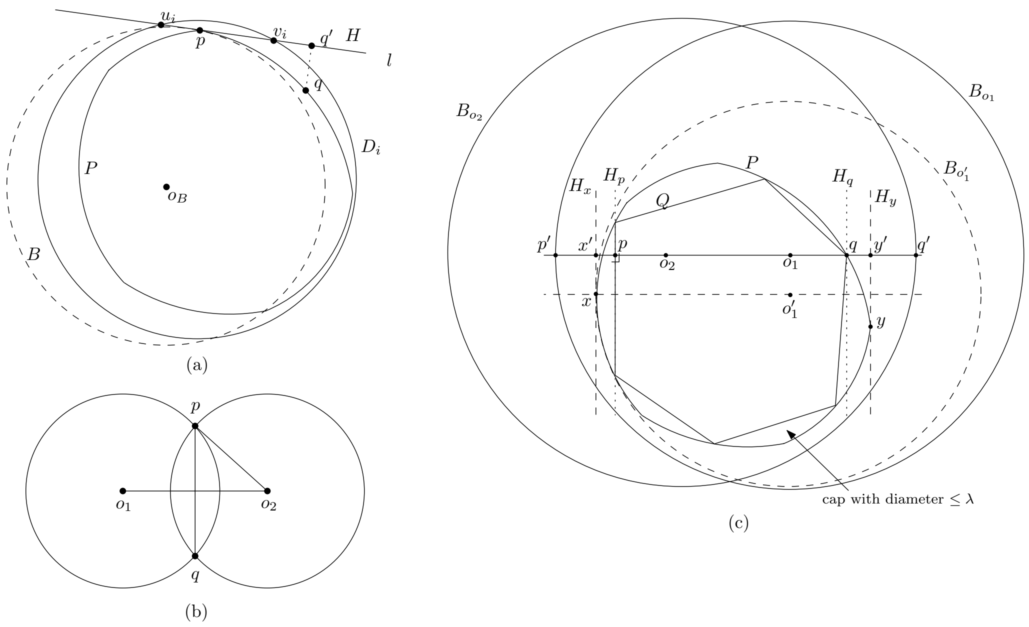

Figure 1: The relation between $\omega_{\min}(P)$ and $\omega_{\max}(P)$

in S_0, we can safely terminate the round, knowing that $\mathcal{A}_{\texttt{OPT}}$ must have sent a message. However, we cannot simply repeat the algorithm and wait till P is empty, since it may never be. Instead we will stop the round when the maximum width of P is small enough (we will show later how to conduct this test efficiently), and then argue that when this happens, S must be empty.

We need the following result proved by Grunbaum [12].

LEMMA 3.2. ([12]) *For a convex set P in $\mathbb{R}^d$, any halfspace that contains the centroid of P also contains at least $1/e$ of the volume of P.*

Since $\texttt{Ball}(f(t_{now}), \beta\Delta)$ is contained in a halfspace not containing the centroid of P, every time a communication is triggered in Algorithm 2, we have

$$\frac{\texttt{vol}(P \cap \texttt{Ball}(f(t_{now}), \beta\Delta))}{\texttt{vol}(P)} < 1 - \frac{1}{e},$$

that is, the volume of the convex set P containing S will be decreased by a constant factor.

The rest of our task is to bound the iterations in each round. At the first glance, the number of iterations could be endless, since P might be cut into thinner and thinner slices. Fortunately, we can show that such a situation will not happen, by making use of the fact that we are cutting P using a series of balls with radii not too large.

LEMMA 3.3. *If $\omega_{\max}(P) \geq \epsilon\Delta$, then $\omega_{\min}(P) = \Omega(\epsilon^2\Delta)$.*

Before proving Lemma 3.3, we first show the following facts.

LEMMA 3.4. *Let H be any supporting hyperplane of P at $p \in \partial P$, that is, H contains p and P is contained in one of the two halfspaces bounded by H. Then there is a ball B with radius $\beta\Delta$ such that H is tangent to B at p and B contains P.*

Proof: It is easy to see that there is a unique ball B with center o_B and radius $\beta\Delta$ such that H is tangent to B at p and B is on the same side of H as P. We show that P must be contained in B. Suppose not, there must be a point $q \in P$ such that $q \notin B$. Let J be the two-dimensional plane spanned by o_B, p and q, intersecting H at line l; Figure 1(a) shows the situation on J. Suppose P is the intersection of $B_i, i = 1, \ldots, m$. Clearly, the intersection between J and any B_i is a disk D_i containing p and q. Simple planar geometry shows that ∂D_i must intersect l at two points, since the radius of D_i is no more than $\beta\Delta$ and $\|o_B q\| > \beta\Delta$. Let u_i, v_i be the two intersection points between ∂D_i and l, and p' be the projection of p on line l. It is also easy to see that one of u_i, v_i, say v_i, is different than p and lies at the same side of p as q'. Therefore, the intersection of all such disks D_i $(1 \leq i \leq m)$ must contain a segment $\overline{pv_j}$ where v_j is the closest point to p among all the points $v_i, i = 1, \ldots, m$, which means that P, the intersection of all the B_i, must lie on both sides of H, a contradiction. $\square$

LEMMA 3.5. *Let M be the intersection of two balls of radius r in R^d. If $\omega_{\max}(M) \geq \epsilon r$, then $\omega_{\min}(M) = \Omega(\epsilon^2 r)$.*

Proof: Let B_1, B_2 be two balls whose intersection is M and let o_1, o_2 be their centers, respectively. Let S_1, S_2 be the boundary of B_1 and B_2. It is clear that the intersection of S_1 and S_2 is a $(d-2)$-dimensional sphere S. Let p be an arbitrary point on S, and let J be the two dimensional plane passing through p, o_1, o_2 and intersecting S at another point q. It is easy to see that $\|pq\|$ is equal to the maximum width of M, and $\omega_{\min}(M)$ is equal to $2(r - \sqrt{r^2 - (\|pq\|/2)^2})$; see Figure 1(b). Thus if $\|pq\| \geq \epsilon r$, then $\omega_{\min}(M) = \Omega(\epsilon^2 r)$. $\square$

Proof: (Lemma 3.3) Let Q be a polytope inscribed in P such that the diameter of every cap formed by the intersection of P and a halfspace bounded by the hyperplane containing a face of ∂Q is no more than λ; see Figure 1(c). Let μ be the direction in which the directional width of Q is minimized. Let H_p and H_q be the two parallel supporting hyperplanes of Q orthogonal to μ. Let p, q be two points on $Q \cap H_p$ and $Q \cap H_q$ respectively so that $\overline{pq}$ is in the direction of μ. Such two points must exist since Q is a polytope. Let H_x and H_y be the two hyperplanes parallel to H_p and H_q and support P at x and y, respectively. Suppose the line $\overline{pq}$ intersects H_x and H_y at x' and y', respectively.

From Lemma 3.4 we know that there is a ball $B_{o'_1}$ centered at o'_1 with radius $\beta\Delta$ containing P and tangent to H_x. Pick o_1 on the line $\overline{pq}$ between p and q such that $\|o_1 x'\| = \|o'_1 x\|$. By triangle inequality it is easy to see that the ball B_{o_1} centered at o_1 with radius $(\beta\Delta + \lambda)$ must contain $B_{o'_1}$ and thereby contain the convex set P. Similarly, there is another ball B_{o_2} entered at o_2 ($o_2 \in \overline{pq}$) with radius $(\beta\Delta + \lambda)$ containing P if we consider y, y' instead of x, x'. Let $p' = \partial B_{o_1} \cap \overline{pq}$ and $q' = \partial B_{o_2} \cap \overline{pq}$. We have

$$\|pq\| \leq \omega_{\min}(P) \leq \|p'q'\|.$$

Note that $\|p'q'\|$ is the minimum width of $M = B_{o_1} \cap B_{o_2}$. By lemma 3.5, we know that $\omega_{\max}(M) = O(\|p'q'\|/\epsilon)$ provided that $\omega_{\max}(M) \geq \epsilon\Delta$. Finally, if we choose λ sufficiently small, we have $\omega_{\min}(P) \geq \Omega(\|p'q'\|) \geq \Omega(\epsilon \cdot \omega_{\max}(M)) \geq \Omega(\epsilon \cdot \omega_{\max}(P))$. $\square$

The lower bound on the minimum width implies a lower bound on the volume of P. Formally, we have (proof omitted from this extended abstract):

LEMMA 3.6. *Let K be a convex set in $\mathbb{R}^d$. If $\omega_\mu(K) \geq r$ for all $\mu \in S^{d-1}$, then $\texttt{vol}(K) \geq r^d/d!$.*

By Lemma 3.3, we know that as long as $\omega_{\max}(P) \geq \epsilon\Delta$, the width of P in all directions is at least $c \cdot \epsilon^2\Delta$ for some constant c, which means that the volume of P is at least $(c \cdot \epsilon^2\Delta)^d/d!$ by Lemma 3.6. Then by Lemma 3.2, as well as the fact that at the beginning of the round, $\texttt{vol}(P) \leq (4\Delta)^d$, we know that after at most

$$\log \frac{(4\Delta)^d}{(c \cdot \epsilon^2\Delta)^d/d!} = O\left(d \log \frac{d}{\epsilon}\right) \tag{3.1}$$

triggerings of communication in Algorithm 2, $\omega_{\max}(P)$ will be less than $\epsilon\Delta$. At this moment, consider the P' obtained by replacing all the balls in Algorithm 2 by balls with radius Δ. By triangle inequality, we know that $P' = \emptyset$. Recall that $\mathcal{A}_{\texttt{OPT}}$ is only allowed an error of Δ. Therefore, $\mathcal{A}_{\texttt{OPT}}$ must have already sent a message since $P' = \emptyset$.

Running time. Generally, it is hard to compute the centroid of a convex body [20]. However, Bertsimas and Vempala [3] showed that there is a randomized algorithm that computes an approximate centroid of a convex body given by a separation oracle. Formally, they proved the following.

LEMMA 3.7. ([3]) *Let K be a convex body in R^d given by a separation oracle, and a point in a ball of radius Δ that contains K. If $\omega_{\min}(K) \geq r$, then there is a randomized algorithm with running time $\text{poly}(d, \log\left(\frac{\Delta}{r}\right))$ that computes, with high probability, the approximate centroid z of a convex set K such that any halfspaces that contains z also contains at least $1/3$ of the volume of K.*

In our case, since P is the intersection of $O(d \log \frac{d}{\epsilon})$ balls, we can simply implement the separation oracle by checking each of these balls one by one. Moreover, $f(t_{start})$ could be used as the starting point p required by Lemma 3.7. We set $r = c \cdot \epsilon^2\Delta$, thus computing approximate centroid could be done in time $\text{poly}(d, \log\left(\frac{1}{\epsilon}\right))$. If the algorithm of Lemma 3.7 fails, then with high probability, $\omega_{\max}(P) < \epsilon\Delta$. This fact together with the discussion after Lemma 3.6 provide us a way to avoid monitoring the maximum width of P at the beginning of each iteration in Algorithm 2, which is expensive. More precisely, we slightly modify Algorithm 2 as follows.

1. Line 2 $\rightarrow$ **while** the number of iterations in the current round is no more than (3.1) **do**

2. Line 3 $\rightarrow$ compute the approximate centroid of P using the algorithm of Lemma 3.7 and assign it to $g(t_{now})$; if the algorithm of Lemma 3.7 fails, terminate the current round;

THEOREM 3.2. *There is an $(O(d\log(d/\epsilon)), 1+\epsilon)$-competitive online algorithm to track any function $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}^d$. The algorithm runs in time $\text{poly}(d, \log\frac{1}{\epsilon})$ at every time step.*

3.2.2 The case with $\beta = 1$ Recall that in Section 3.1, we have shown that considering $S_0 = C_2 \cup C_3 \ldots \cup C_{d+1}$ is enough. Since S_0 is the collection of points that are centers of the smallest enclosing balls of at most $d+1$ points in $\mathbb{Z}^d$, the following fact can be established (proof omitted from this extended abstract).

LEMMA 3.8. *For any points $s = (x_1, \ldots, x_d)$ in S_0, x_i $(1 \le i \le d)$ are fractions in the form of $\frac{y}{z}$ where y, z are integers and $|z| \le d!(16\Delta^2 d)^d$.*

By this observation, we know that the distance between any two points in S_0 is at least $1 \Big/ \big(d!(16\Delta^2 d)^d\big)^2$. Therefore, by setting $\epsilon = 1 \Big/ 8\Delta\big(d!(16\Delta^2 d)^d\big)^2$, we know that once $\omega_{\max}(P) < \epsilon\Delta$, there is at most one point of S_0 in P. The rest of our job is to find such a point if it exists. Once the point is found, we just send it to Bob, and the round will terminate as soon as $f(t_{now})$ gets Δ away from this point. However, directly computing such a point might be expensive. Instead we use an indirect way to find the last surviving point.

We say a number x is *good* if $x = \frac{y}{z}$ with $y, z \in \mathbb{Z}$ and $|z| \le d!(16\Delta^2 d)^d$. A point s is *good* if all of its coordinates are good. The basic idea is that if we can successfully compute the centroid p of P, we can snap p to its nearest good point s. If there is a point $s' \in S_0$ inside P, then we must have $s' = s$. Thus if $s \notin P$, we simply terminate the current round; otherwise s must be the last point of S_0 in P. The difficulty is that $\omega_{\min}(P)$ could be very small, so that Lemma 3.7 cannot be applied directly. To avoid such a situation, we expand P slightly by increasing all the balls' radii from Δ to $(1+\epsilon)\Delta$. Denote by P' the intersection of these enlarged balls. The observation is that, by our choice of ϵ, if there is a point s' of S_0 in P, then s' is still the only point of S_0 in P'. Now we can apply the algorithm of Lemma 3.7 on P' with $r = c \cdot \epsilon^2 \Delta$. If the algorithm fails, we know that P must be empty. Otherwise we obtain a point $p \in P'$. Finally, we find s by rounding each coordinate of p to its nearest good number, and check if $s \in P$. The rounding could be done in polynomial time according to a theorem by Khintchine (cf. [14, Chapter 4]).

By the choice of ϵ and Theorem 3.2, we obtain the following:

THEOREM 3.3. *There is an $O(d^2 \log(d\Delta))$-competitive online algorithm to track any function $f : \mathbb{Z}^+ \to \mathbb{Z}^d$. The algorithm runs in time $\mathrm{poly}(d, \log \Delta)$ at every time step.*

3.3 Online tracking a dynamic set One of the main applications of online tracking in high dimensions is tracking a dynamic set. Formally, we want to track the function $f : \mathbb{Z}^+ \to 2^U$, where U is a finite universe consisting of d items. We can represent each set $X \in 2^U$ as a {0,1}-vector in $\mathbb{R}^d$, and define the difference between two sets X and Y to be the l_2 distance between the corresponding vectors in $\mathbb{R}^d$ (note that the Hamming distance between two sets is just the square of their l_2 distance). Ideally, Alice should send out subsets of U to approximate $f(t_{now})$, but applying our previous algorithms would send out vectors with fractional coordinates. Unfortunately, if we insist that Alice always sends a set, that is, a {0,1}-vector to Bob, the competitive ratio would be exponentially large in Δ, even we allow a relatively larger β, as shown in the next theorem.

THEOREM 3.4. *Suppose that there is an (α, β)-competitive algorithm for online tracking $f : \mathbb{Z}^+ \to 2^U$ and $|U| > (\beta\Delta)^2$, if the algorithm can only send subsets of U, then $\alpha = 2^{\Omega(\Delta^2)}$ for any constant $\beta < 19/18$.*

Proof: Without lose of generality, let $H = \{0,1\}^d$, where d is chosen to be $(\beta\Delta)^2 + 1$. Similar to the proof of theorem 2.2, we just need to show that the adversary can manipulate $f(t)$ so that a round will have at least α iterations.

Let S_0 be the set of possible vertices sent by $\mathcal{A}_{\texttt{OPT}}$ in its last communication, that is, all the vertices within distance Δ from $f(t_{start})$. The cardinality of S_0 is

$$|S_0| = \sum_{k=1}^{\Delta^2} \binom{d}{k} = \Omega(2^{\Delta^2}).$$

The adversary Carole sets $S = S_0$ at the beginning of each round and then manipulates the value of the function f according to the online algorithm $\mathcal{A}_{\texttt{SOL}}$, as follows. Whenever $\mathcal{A}_{\texttt{SOL}}$ sends $\mathbf{v} \in H$, Carole changes f to $\mathbf{u} = \mathbf{1} - \mathbf{v}$, that is, flipping all the coordinates of $\mathbf{v}$. Since $\|\mathbf{v}, \mathbf{u}\| > \beta\Delta$, $\mathcal{A}_{\texttt{SOL}}$ has to communicate again. Every time Carole uses a value $\mathbf{u}$ for f, S is cut as $S \leftarrow S \cap \texttt{Ball}(\mathbf{u}, \Delta)$. So S loses at most (let $\epsilon = \beta - 1 < 1/18$)

$$|H - \texttt{Ball}(\mathbf{u}, \Delta)| = \sum_{k=\Delta^2+1}^{d} \binom{d}{k} \le \binom{2\Delta^2}{3\epsilon\Delta^2} \le (e/\epsilon)^{3\epsilon\Delta^2}$$

elements. Therefore, $\mathcal{A}_{\texttt{SOL}}$ will communicate at least

$$\Omega\left(\frac{2^{\Delta^2}}{(e/\epsilon)^{3\epsilon\Delta^2}}\right) = \Omega(c^{\Delta^2}) \quad (c > 1 \text{ when } \epsilon < 1/18)$$

times before S becomes empty. □

Therefore, to avoid an exponentially large competitive ratio, we have to allow the algorithm to send vectors with fractional coordinates. We can use the previously

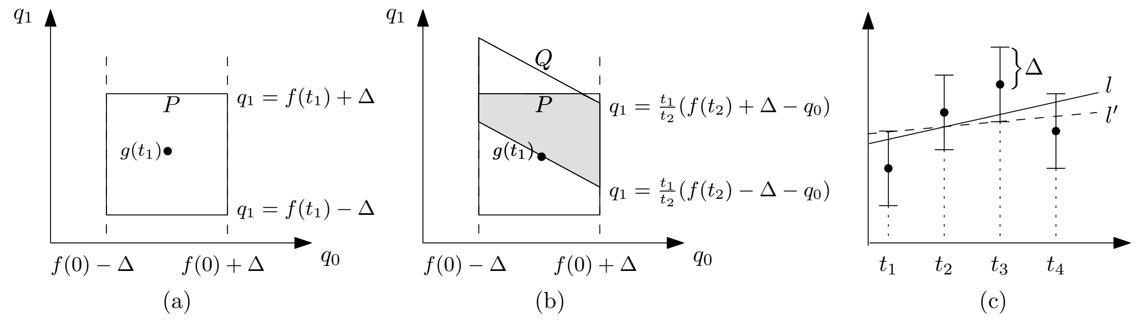

Figure 2: (a, b) Cutting in the parametric space. (c) Considering a small set of lines is enough.

developed algorithms to guarantee that the l_2 distance between $f(t_{now})$ and the fractional vector $g(t_{last})$ sent by our algorithm is no more than Δ. If in some applications it is unnatural to report to the client a vector with fractional values when the underlying function being tracked is a set, the tracker could convert the vector to a set Y by probabilistically rounding every coordinates of $g(t_{last})$. It can be easily shown that the expected distance between Y and $f(t_{now})$ is no more that Δ.

Finally, we notice that if f is always a $\{0, 1\}$-vector in $\mathbb{R}^d$, the points in S_0 must be in the form of $\{0, \frac{1}{2}, 1\}^d$, namely, the distance between any two points in S_0 is at least $1/2$. Applying this fact in the algorithm of Section 3.2.2 gives us the following:

THEOREM 3.5. *There is an $O(d \log(d))$-competitive online algorithm to track any dynamic set $f : \mathbb{Z}^+ \to 2^U$.*

4 Online Tracking with Predictions

In this section, we further generalize our model by considering "predictions" . We assume that Alice tries to predict the future trend of the function based on history, and then sends the prediction to Bob. If the actual function values do not deviate from the prediction by more than Δ, no communication is necessary. One can imagine that when f is "well behaved", using good predictions could greatly reduce communications incurred. Indeed, the same approach has been taken in many heuristics in practice [6, 7, 13]. In this paper we only consider the case where the algorithms (both the online and the offline) use linear functions as predictions, and for $d = 1$; the technique can be extended to more general prediction functions and high dimensions.

In one dimension, the offline problem is to approximate a function by a small number of straight line segments. O'Rourke [18] gave a linear-time algorithm to compute the optimal solution. His algorithm is "online" but in the sense that the algorithm scans f only once, and the partial solution computed so far is optimal for the portion of f that has been scanned. However, the partial solution could keep changing at each time step as f is observed. While in our problem, we need to make an immediate decision on what to communicate at each time step whenever f deviates more than Δ from the prediction previously sent.

Our algorithm with line predictions still follows the general framework outlined in Section 2. At the beginning of each round (assuming $t_{start} = 0$), we just send $f(0)$ to Bob, and predict f to be $f(0)$. Let t_1 be the time of the first triggering. We parameterize the lines by q_0, q_1, meaning that the line (q_0, q_1) passes through $(0, q_0)$ and (t_1, q_1). We call the (q_0, q_1)-space the parametric space, thus any line sent out by the algorithm is a point in the parametric space. Let P be the region in the parametric space consisting of all the points that are valid Δ-approximations of $f(0)$ and $f(t_1)$, which is a square (Figure 2(a)). We will pick a point $g(t_1)$ in P and send it to Bob. Suppose at time t_2, $g(t_1)$ fails to approximate $f(t_2)$. Let Q be the region in the parametric space consisting of all the valid Δ-approximations of $f(0)$ and $f(t_2)$, which can be shown to be a parallelogram (Figure 2(b)). We update $P \leftarrow P \cap Q$, and then iterate the procedure. It is easy to see that if $\mathcal{A}_{\texttt{OPT}}$ does not need any further communication in the current round, its last message must lie inside P.

The major task is to choose the initial set $S = S_0$ at the beginning of the round. After that, the algorithm is similar to that in Section 3.1, that is, at every triggering we update $S \leftarrow S \cap P$ and send the Tukey median of S. The analysis also follows the same line. Let $M = \{(t, y) \mid t \in [T], y \in \{\mathbb{Z}+\Delta\} \cup \{\mathbb{Z}-\Delta\}\}$, where $\{\mathbb{Z}+\Delta\}$ denotes the set $\{x \mid x = y + \Delta, y \in \mathbb{Z}\}$, and similarly $\{\mathbb{Z} - \Delta\}$. Let $\mathcal{L}$ be the collection of lines passing through two points in M. Let X be the collection of intersection points between line $t = 0$ and lines in $\mathcal{L}$, and Y be the collection of intersection points between line $t = t_1$ and lines in $\mathcal{L}$. We choose S_0 to be $\{(q_0, q_1) \mid q_0 \in X, q_1 \in Y\} \cap P$ (P is the first square we get).

We argue that only considering the points (lines) in S_0 is sufficient for our purpose. In particular, we can show that if $\mathcal{A}_{\texttt{OPT}}$ keeps silent in the current round, there must be some surviving point (line) in S_0. Consider the original function space (Figure 2(c)). Let l be the line chosen by $\mathcal{A}_{\texttt{OPT}}$ in its last communication. Suppose that $\mathcal{A}_{\texttt{OPT}}$ has not made any communication in the current round, l must intersect with all the line segments $((t, f(t) - \Delta), (t, f(t) + \Delta))$, for $t_{start} \leq t \leq t_{now}$. We can always rotate and translate l so that it passes through two points in M, and it still intersects with all lines segments (line l' in Figure 2(c)). Therefore, l' must still *survive* at the current time.

Finally we bound the cardinality of S_0 (proof omitted from this extended abstract).

LEMMA 4.1. $|S_0| = O(\Delta^2 T^6)$.

Therefore, S will become empty after at most $O(\log(\Delta T))$ iterations.

THEOREM 4.1. *There is an $O(\log(\Delta T))$-competitive online algorithm to track any function $f : \mathbb{Z}^+ \to \mathbb{Z}$ with line predictions, where T is the length of the tracking period.*

The algorithm above assumes that T is given in advance in order to initialize S_0. If T is not known, we can use the following squaring trick to keep the competitive ratio. We start with T being set to Δ. Whenever t_{now} reaches T and the current round has not finished yet, we restart the round with $T \leftarrow \Delta T^2$. It can be easily shown that the number of iterations in a round is still at most $O(\log(\Delta T))$.

5 Open Problems

As mentioned in the related work, the problem studied in this paper is a special case of the distributed tracking framework where there is only one site. It would be nice to generalize our techniques to multiple sites. Secondly, in the d-dimensional case, if we consider the number of bits (instead of number of messages) the algorithms has sent, the competitive ratios of our current algorithms will increase by roughly a factor of d. Thus we want to ask whether we can do better by a subset of the coordinates instead of a whole vector in $\mathbb{R}^d$. Finally, it is also interesting to consider online tracking problems in other metric spaces.

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999.

[2] A. Beresford and F. Stajano. Location privacy in pervasive computing. *Pervasive Computing, IEEE*, 2003.

[3] D. Bertsimas and S. Vempala. Solving convex programs by random walks. *Journal of the ACM*, 2004.

[4] B. Chandramouli, J. M. Phillips, and J. Yang. Valuebased notication conditions in largescale publish/subscribe systems. In *VLDB*, 2007.

[5] K. L. Clarkson, D. Eppstein, G. L. Miller, C. Sturtivant, and S. Teng. Approximating center points with iterated radon points. In *SoCG*, 1993.

[6] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, 2005.

[7] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *SIGMOD*, 2005.

[8] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed functional monitoring. In *SODA*, 2008.

[9] S. Davis, J. Edmonds, and R. Impagliazzo1. Online algorithms to minimize resource reallocations and network communication. In *APPROX and RANDOM*, 2006.

[10] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[11] Y. Diao, S. Rizvi, and M. J. Franklin. Towards an internet-scale xml dissemination service. In *VLDB*, 2004.

[12] B. Grunbaum. Partitions of mass-distributions and of convex bodies by hyperplanes. *Pacific J. Math*, 1960.

[13] R. Keralapura, G. Cormode, and J. Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *SIGMOD*, 2006.

[14] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer-Verlag, 4th edition, 2007.

[15] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, 2005.

[16] J. Matousek. *Lectures on Discrete Geometry*. Springer-Verlag New York, Inc., 2002.

[17] C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *SIGMOD*, 2001.

[18] J. O'Rourke. An on-line algorithm for fitting straight lines between data ranges. *Communications of the ACM*, 24(9), 1981.

[19] G. Pottie and W. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[20] L. A. Rademacher. Approximating the centroid is hard. In *SoCG*, 2007.

[21] J. H. Schiller and A. Voisard. *Location-based services*. Morgan Kaufmann Publishers, 2004.

[22] A. C. Yao. Some complexity questions related to distributive computing. In *STOC*, 1979.

[23] Y. Yao and J. Gehrke. Query processing for sensor networks. In *CIDR*, 2003.