Competitive Query Strategies for Minimising the Ply of the Potential Locations of Moving Points

William Evans David Kirkpatrick
Department of Computer Science
University of British Columbia, Canada
will@cs.ubc.ca kirk@cs.ubc.ca

Maarten Löffler Frank Staals
Department of Information and Computing Sciences
Utrecht University, the Netherlands
m.loffler@uu.nl f.staals@uu.nl

ABSTRACT

We study the problem of maintaining the locations of a collection of n entities that are moving with some fixed upper bound on their speed. We assume a setting where we may query the current location of entities, but handling this query takes a certain unit of time, during which we cannot query any other entities. In this model, we can never know the exact locations of all entities at any one time. Instead, we maintain a representation of the potential locations of all entities. We measure the quality of this representation by its *ply*: the maximum number of entities that could potentially be at the same location.

Since the ply could be large for every query strategy, we analyse the performance of our algorithms in a competitive framework: we consider the worst case ratio of the ply achieved by our algorithms to the *intrinsic ply* (the smallest ply achievable by any algorithm, even one that knows in advance the full trajectories of all entities). We show that, if our goal is to mimimise the ply at some number τ of time units in the future, an $O(1)$-competitive algorithm exists, provided τ is sufficiently large. If τ is small and the n entities move in any constant dimension d, our algorithm is $O\left((\frac{\ell}{\tau}+1)^{d-\frac{d}{d+1}}\right)$-competitive, where ℓ is the average time since the last query over all entities. We also provide matching lower bounds, and we show that computing the intrinsic ply exactly is NP-hard, even when the trajectories are known in advance.

SoCG'13, June 17-20, 2013, Rio de Janeiro, Brazil.

1. INTRODUCTION

Due to the rapid growth in availability of cheap location-aware mobile devices, *data in motion* is increasingly becoming a topic of interest to researchers in various application fields, such as GIS, sensor networks, social networks, robotics, etc. [2, 8, 19, 24]. Although these applications are very different, they share some important characteristics: the motion they deal with is often *unpredictable*, and data must be processed in *real-time*. These two aspects make it hard to apply traditional geometric algorithms.

The first challenge arises from the unpredictable nature of the data. In computational geometry, there is an extensive history on data in motion: kinetic data models [1, 3, 12, 13] can be used to study the complexity and structural changes of geometric objects in motion. However, they were developed to deal with moving data in a controlled environment and rely on the possibility to predict, at least locally, the trajectory a moving point will follow. More recently, several isolated efforts have been made to deal with data in much less restricted models of motion (though some assumptions are still necessary, such as an upper bound on the speed of moving points) [22, 6, 7, 9, 11, 18, 23].

The second challenge is obtaining and processing the data in real-time. Traditional algorithms work on the assumption that data is first collected and stored in a central location, and then analysed, preferably as quickly as possible (i.e., minimising computation time). The *online algorithms* framework lifts this assumption, and allows data to be processed as it arrives, but ignores the cost of obtaining the data and does not permit the algorithm to influence the order in which the data arrives. In modern applications, a significant cost may be associated with obtaining the locations of the entities being studied, and computation time may not be the restricting factor. On the other hand, in the *update complexity* model, one is given an incomplete data set on which some structure (for example, the Delaunay triangulation) has to be computed using a minimum number of "updates"[4, 10, 14, 15, 20, 21]. Here, an update typically returns additional information (precision) about the location of a specified data point. However, this has mostly been considered in a static context.

Model and contribution.

When dealing with real-time data in motion that takes a certain amount of time to obtain, minimising the number of required updates may not be the true goal. Instead, we should try to maximise the effect of the updates that we can

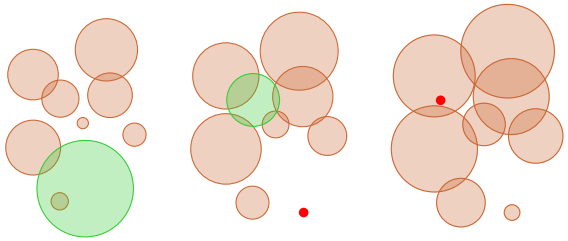

Figure 1: Illustrating the model at three subsequent time steps. In each step, one disk (green) is queried, resulting in a point (red) at the next step. All other disks grow in diameter by 1.

make, knowing that each update of one entity takes time during which all entities may move.

There is a natural connection between data imprecision and data in unpredictable motion. When one knows the location of a moving entity at a certain time, but does not constantly refresh this information, the entity can be anywhere in a region of potential locations, typically a ball (stale data). A natural question is whether it is possible to keep track of imprecise and moving data using queries, that take unit time, to individual data items.

In this paper, we consider one fundamental interpretation of this question. Given a set of entities moving unpredictably with a bounded speed, we wish to maintain a representation of their potential locations that can distinguish them as well as possible. At any point in time, we measure this by the *ply* [17] of the set of balls: the maximum number of balls that contain any given point in $\mathbb{R}^d$. In many applications involving imprecise numbers or points, a low ply implies more efficient algorithms [5, 16].

1.1 Problem statement

We consider the following setting. Let $\mathcal{E}$ be a set $\{e_1, e_2, \ldots, e_n\}$ of point entities in $\mathbb{R}^d$, where every entity e_i has an associated (unknown) *trajectory* f_i, which is a continuous function from time ($\mathbb{R}$) to position ($\mathbb{R}^d$, for some dimension d). An entity can move in any direction with a maximum speed of $1/2$ per unit of time. We wish to maintain a representation of the locations of these entities. To do this, we are allowed to *query* the current location of a given entity, which takes an amount of time which we assume to be 1.[1] As a result, at any point t in time, each entity e_i can be represented by a ball $B_i(t)$ of *possible* locations, which we refer to as the *uncertainty region for* e_i. This ball is centered at $f_i(t_i)$, the location of e_i at its *most recent query time* t_i, and has diameter $d_i(t) = t - t_i$. Figure 1 illustrates the model in $\mathbb{R}^2$.

Let $\mathcal{E}' = \{e'_1, e'_2, \ldots, e'_m\}$ be any subset of the entities in $\mathcal{E}$. We say that the collection $\mathcal{B}' = \{B_1, B_2, \ldots B_m\}$ of balls, forms an *uncertainty realisation* of $\mathcal{E}'$ at time t if there exists a sequence of distinct query times $t_1, t_2, \ldots t_m$, all less than t, such that B_i has center $f_i(t_i)$ and diameter $d_i(t)$. We refer to the minimum, over all uncertainty realisations $\mathcal{B}'$ of $\mathcal{E}'$, of the ply of $\mathcal{B}'$, as the *intrinsic ply* of the set $\mathcal{E}'$.

Now, the question is: what is a good query strategy? Clearly, we might as well make a new query at every unit of time. Also clearly, we might not be able to ensure a good ply with any strategy, if the entities move too close together. Therefore, we apply a form of competitive analysis. Since the set of uncertainty regions associated with any query strategy at some fixed time t provides an uncertainty realisation of the set of entities $\mathcal{E}'$ at time t, the intrinsic ply of $\mathcal{E}'$ provides a lower bound on the ply that could be achieved by *any* query strategy that attempts to minimise ply at time t, even one that has full knowledge of the trajectories of the entities involved.

Single shot and recurrent ply minimisation.

A common situation is that, at current time t, we are interested in the uncertainty regions associated with a set of entities at some future time $t^* = t + \tau$. When t^* is understood, we refer to these as their *projected* uncertainty regions. The projected uncertainty region $B_i^* = B_i(t^*)$ of an entity e_i has diameter $d_i(t) + \tau$. Note that this means that the entity queried at time t, when the remaining time is τ, has diameter τ.

We begin by considering the *single shot* problem in which we start at a time t^0 and the goal is to minimise the ply of the uncertainty regions at one fixed time $t^* = t^0 + \tau$ which lies $\tau > 0$ time units in the future.

Figure 2(a) illustrates a collection of trajectories and their associated uncertainty regions for $d = 1$, as time progresses. In the single shot problem we are only interested in the uncertainty regions at time t^*. That is, the intersection of the brown cones with the horizontal line at t^*.

We also consider a *recurrent* version of ply minimisation in which the goal is to minimise the ply at regular checkpoints, spaced τ units apart.

1.2 Overview of results

In this paper we present competitive algorithms to query a set of moving entities in $\mathbb{R}^d$ to minimise the ply of their uncertainty regions at a given time τ time units in the future. We show that if τ is large enough, at least twice as large as the number of entities n, then our algorithm can achieve a competitive ratio in $O(1)$. When we have less than $2n$ remaining time, the competitive ratio depends on the average time ℓ since the last query, and hence the average initial size of the uncertainty regions. More precisely, our algorithms obtain a competitive ratio of $O\left((\ell/\tau + 1)^{d - \frac{d}{d+1}}\right)$, for any constant d. In Section 3 we provide a lower bound on the competitive ratio, which shows that our algorithms are within a constant factor of the best achievable competitive ratio. We show that computing the intrinsic ply of the entities exactly is NP-hard in Section 4, and we briefly discuss the recurrent version of the problem in Section 5.

2. COMPETITIVE ALGORITHMS FOR THE SINGLE SHOT PROBLEM

Recall that our input is a set $\mathcal{E}$ of n entities $\{e_1, e_2, \ldots, e_n\}$ where every entity e_i has an associated (unknown) trajectory f_i and a *most recent query time* $t_i \in \mathbb{Z}$. We start at time t^0, and our goal is to minimise the ply at given time $t^* = t^0 + \tau$, with $\tau > 0$. We will denote the intrinsic ply of $\mathcal{E}$ at time t^* by Δ. We start with some basic observations and lemmas that form the core of our algorithms.

[1] Note that setting both the speed limit and sampling time does not restrict the generality of the problem. We chose $1/2$ because this will turn out to be convenient later.

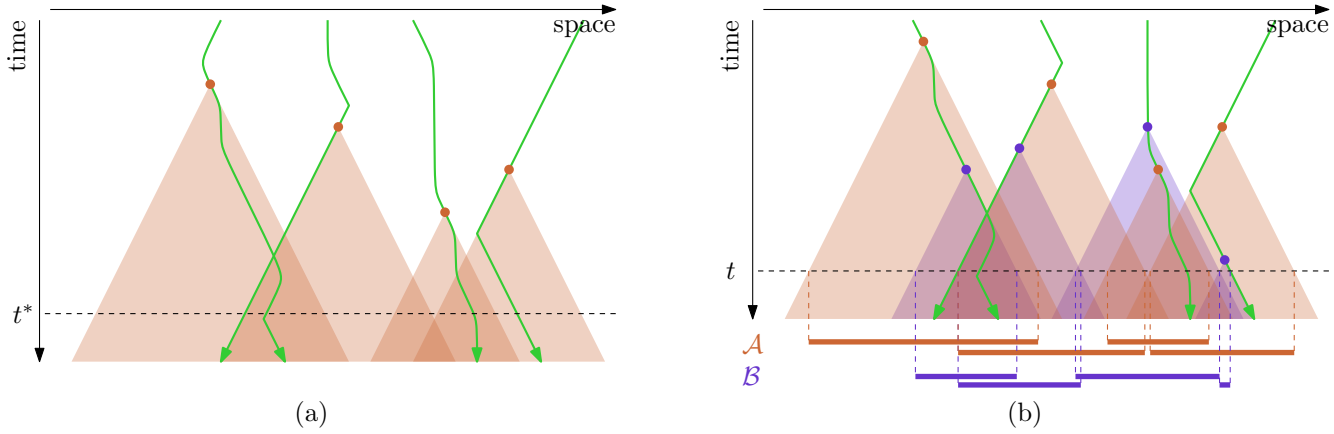

Figure 2: (a) The single shot problem for $d = 1$. The true trajectory of each entity is a y-monotone curve with a slope bounded by $1/2$, and is drawn in green. (b) Showing two sets of intervals $\mathcal{A}$ and $\mathcal{B}$, coming from different samplings of the same entities.

OBSERVATION 2.1. *Suppose that $\mathcal{A} = \{A_1, .., A_n\}$ and $\mathcal{B} = \{B_1, .., B_n\}$ are uncertainty realisations of the entities in $\mathcal{E}$ at time t. Then for all $1 \le i \le n$, $A_i \subseteq B_i$ or $B_i \subseteq A_i$.*

Proof: Suppose w.l.o.g. that B_i has diameter d_i and that the diameter of A_i is $d_i + u$, for some non-negative integer u. Then the center of B_i (the location of e_i at time $t - d_i$), must lie in the ball of diameter u centered at the center of A_i (the location of e_i at time $t - d_i - u$). It follows that $B_i \subseteq A_i$. □

Let $\nu_d = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$ be the volume coefficient of a d-dimensional ball. For a set of balls $\mathcal{B}$, let the *span* of $\mathcal{B}$ denote the d-dimensional volume of their union.

LEMMA 2.2. *Let $\mathcal{B}$ be any uncertainty realisation of a set of entities $\mathcal{E}$ at time t, whose intrinsic ply at the same time is Δ. Then the span of $\mathcal{B}$ is at least $C_d|\mathcal{E}|^{d+1}/\Delta$ where $C_d = \frac{\nu_d}{(d+1)4^d}$ for $d \ge 1$.*

Proof: Let $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$ be the uncertainty realisation of $\mathcal{E}$, and let $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ be any uncertainty realisation of $\mathcal{E}$ that realises the intrinsic ply Δ of $\mathcal{E}$ at time t. The span of $\mathcal{B}$ is $\mathrm{vol}\left(\bigcup_{i=1}^n B_i\right)$, which satisfies:

$$\mathrm{vol}\left(\bigcup_{i=1}^{n} B_i\right) \ge \mathrm{vol}\left(\bigcup_{i=1}^{n} A_i \cap B_i\right) \ge \frac{1}{\Delta}\sum_{i=1}^{n} \min\{\mathrm{vol}\,(A_i), \mathrm{vol}\,(B_i)\}$$

where the last inequality holds because either $A_i \subseteq B_i$ or $B_i \subseteq A_i$ for all i (Observation 2.1), and the regions $\{A_1 \cap B_1, A_2 \cap B_2, \ldots A_n \cap B_n\}$ have ply at most Δ. Since the smallest uncertainty region has diameter 1 and each set $\mathcal{A}$ and $\mathcal{B}$ contains regions with distinct integral diameters,

$$\sum_{i=1}^{n} \min\{\mathrm{vol}\,(A_i), \mathrm{vol}\,(B_i)\} \ge$$

$$\sum_{i=1}^{n} \min\{V_d(i/2), V_d((n-i+1)/2)\} = 2\sum_{i=1}^{\lceil n/2 \rceil} V_d(i/2).$$

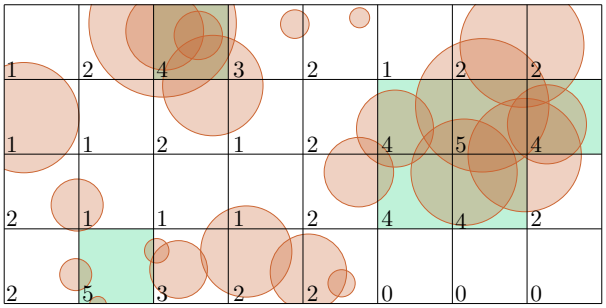

Figure 3: A set of uncertainty regions $\mathcal{B}$ and a partition Ξ of $\mathbb{R}^2$ into cells. The heaviness of each cell is given; cell of heaviness at least 4 are shaded.

where $V_d(r) = \nu_d r^d$ is the volume of a d-dimensional ball with radius r. The result follows by bounding the sum by an integral. □

For a positive number w, we define a *w-partition* of $\mathbb{R}^d$ to be a regular grid with *cells* of width w; that is, the set of hypercubes of the form $\Xi = \{[\mathbf{i}_1 w, (\mathbf{i}_1 + 1)w] \times [\mathbf{i}_2 w, (\mathbf{i}_2 + 1)w] \times \ldots \times [\mathbf{i}_d w, (\mathbf{i}_d + 1)w] \mid \mathbf{i} \in \mathbb{Z}^d\}$. We say a cell $\xi \in \Xi$ is *h-heavy* with respect to a set of d-dimensional balls $\mathcal{B}$, if at least h balls of $\mathcal{B}$ intersect ξ. These definitions are illustrated for $d = 2$ in Figure 3.

LEMMA 2.3. *Let $\mathcal{B}$ be any collection of uncertainty regions at time t of maximum diameter w, whose associated entities have intrinsic ply Δ at time t. Then at most $6w\left(\frac{\Delta}{C_d h}\right)^{1/d}$ regions in $\mathcal{B}$ intersect an h-heavy cell in any w-partition of $\mathbb{R}^d$.*

Proof: Suppose that z regions intersect heavy cells. These z regions must have span at least $C_d z^{d+1}/\Delta$ by Lemma 2.2. Thus they intersect at least $C_d z^{d+1}/(\Delta w^d)$ cells of size w, in total. Some of these cells may be *light* (non-h-heavy), but every light cell must be adjacent to at least one heavy cell since each cell is intersected by an uncertainty region with diameter at most w that intersects a heavy cell. Thus at least one of every 3^d of these cells is heavy and consequently

there are at least $C_d z^{d+1}/(\Delta(3w)^d)$ heavy cells. These heavy cells are each intersected by at least h regions, so there are a total of $hC_d z^{d+1}/(\Delta(3w)^d)$ region intersections with heavy cells. This means that $2^d z \geq hC_d z^{d+1}/(\Delta(3w)^d)$, since each region intersects at most 2^d cells. Thus, $z \leq 6w\left(\frac{\Delta}{C_d h}\right)^{1/d}$. □

2.1 A competitive algorithm for $\tau \geq 2n$

In the case that we start with sufficient time relative to the number of entities, we can show that the following simple algorithm produces a solution of ply $O(\Delta)$, that is, we can guarantee a constant competitive ratio in any fixed dimension. The algorithm is as follows:

Algorithm PlentyOfTime($\mathcal{E}, \tau$)
Input. A set of n entities $\mathcal{E}$, and the remaining time τ, with $\tau \geq 2n$.
1. **if** $\tau > 0$ **then**
2. Wait (query nothing) for $\tau - 2n$ time steps.
3. Query all entities in $\mathcal{E}$ once.
4. Let $\mathcal{B} = \{B_1, .., B_n\}$ be the projected uncertainty regions associated with entities in $\mathcal{E}$.
▷ *The max. diameter of the regions in* $\mathcal{B}$ *is* $2n$. ◁
5. Form a $2n$-partition of $\mathbb{R}^d$, and choose the smallest h so that at most $n/2$ regions in $\mathcal{B}$ intersect h-heavy cells.
6. Let $\mathcal{E}' \subseteq \mathcal{E}$ be the entities whose projected uncertainty regions intersect h-heavy cells.
7. Recursively call PlentyOfTime($\mathcal{E}', n$).

Lemma 2.4. *Algorithm* PlentyOfTime *with n entities that have intrinsic ply Δ at time $t^* = t^0 + \tau$, with $\tau \geq 2n$, yields uncertainty regions at time t^* with ply at most $(24)^d\Delta/C_d$.*

Proof: Consider any call to PlentyOfTime with input $\hat{\mathcal{E}} \subseteq \mathcal{E}$ and remaining time $\hat{\tau} \leq \tau$. Note that the assumptions on the input are satisfied in any recursive call since $|\mathcal{E}'| \leq n/2$. From Lemma 2.3 with $t = t^*$ it follows that the algorithm will choose $h \leq (24)^d\hat{\Delta}/C_d$, where $\hat{\Delta}$ is the intrinsic ply of the set $\hat{\mathcal{E}}$. Since $\hat{\mathcal{E}} \subseteq \mathcal{E}$, the intrinsic ply Δ of $\mathcal{E}$ is at least $\hat{\Delta}$. Hence, $h \leq (24)^d\Delta/C_d$.

The entities that we set aside before each recursive call to PlentyOfTime intersect only non-h-heavy cells, so their projected uncertainty regions cover a space that has ply less than h, no matter how the other entities are queried. Eventually, all entities are set aside and the result of the algorithm's queries is a set of uncertainty regions at time t^* with ply less than $(24)^d\Delta/C_d$. □

2.2 A competitive algorithm for $\tau = n$

When $\tau \leq 2n$, we can no longer afford to start by querying all entities at least once. We investigate how to determine a subset of entities to focus on without querying the others ever, and how this influences the competitive ratio. The competitive ratio now not only depends on the intrinsic ply Δ, but also on the average time since the last query to an entity. We focus first on the case $\tau = n$, and then describe the general algorithm later.

The main idea is to set aside entities until we are in the situation where we have twice as much time as entities again, so we can apply PlentyOfTime. The key observation is that the ply of the projected uncertainty regions associated with the entities that we set aside is not too large.

We begin with n entities, $\tau = n$ remaining time, and ℓ the average time since the last query over all entities. With maximum speed 1/2, $\ell + \tau$ is the average diameter of the entities' projected uncertainty regions. Let $\lambda = \left(\frac{\ell+\tau}{\tau}\right)^d$.

Algorithm NoTimeToLose!($\mathcal{E}, \tau$)
Input. A set $\mathcal{E}$ of n entities and $\tau = n$ remaining time; ℓ is the average time since the last query over all entities.
▷ *The average diameter of the projected uncertainty regions is* $\ell + \tau = \lambda^{1/d}\tau$. ◁
1. $\mathcal{J} = \emptyset$
2. **until** $|\mathcal{J}| \geq 2\tau/\lambda^{1/(d+1)}$
3. ▷ *Call an entity* narrow *if its projected uncertainty region has diameter* $\leq 2(\ell + \tau)$. ◁
4. Choose a maximal disjoint set $\mathcal{I}$ of narrow entities in $\mathcal{E} \setminus \mathcal{J}$.
5. **if** $|\mathcal{I}| \leq 1$ **then** Query all entities in $\mathcal{E}$ once & stop.
6. **else** $\mathcal{J} = \mathcal{J} \cup \mathcal{I}$
7. Set aside the entities in $\mathcal{J}$ and query each of the remaining at most $\tau - 2\tau/\lambda^{1/(d+1)}$ entities once, in any order.
8. Form a τ-partition of $\mathbb{R}^d$, and choose the smallest h so that at most $\tau/\lambda^{1/(d+1)}$ of the remaining projected uncertainty regions intersect h-heavy cells. Set aside the entities whose projected uncertainty regions intersect only non-h-heavy cells.
9. Let $\mathcal{E}'$ be the at most $\tau/\lambda^{1/(d+1)}$ remaining entities, and let $\tau' = 2\tau/\lambda^{1/(d+1)}$ be the remaining time.
10. Call PlentyOfTime($\mathcal{E}', \tau'$).

Lemma 2.5. *The ply of the regions produced if* NoTimeToLose! *stops at line 5 is $O((n/\Delta)^{d/(d+1)})\Delta$.*

Proof: Let p be the ply of the uncertainty regions (at time t^*) produced at line 5 and let $\mathcal{B}$ be a subset of p of these regions that all contain a common point. Since every region in $\mathcal{B}$ has diameter at most n, the span of $\mathcal{B}$ is at most $(2n)^d$. The entities associated with the regions $\mathcal{B}$ have intrinsic ply at most Δ, so, by Lemma 2.2, their span is at least $C_d p^{d+1}/\Delta$. Together, these bounds imply that $p \leq \Delta^{1/(d+1)}(2n)^{d/(d+1)}/C_d^{1/(d+1)}$. □

Lemma 2.6. NoTimeToLose! *in line 7 sets aside regions that contribute at most $4\lambda^{d/(d+1)}\Delta(12)^d/C_d$ to the final ply.*

Proof: Since $\ell + \tau$ is the average diameter of the projected uncertainty regions, $n/2 = \tau/2$ of these (the *narrow* entities) have projected uncertainty regions of maximum diameter at most $2(\ell + \tau)$. These narrow entities have intrinsic ply at most Δ and, by Lemma 2.2, their associated projected uncertainty regions have span at least $\tau^{d+1}C_d/(2^{d+1}\Delta)$. If we choose any narrow region, that region and all the narrow regions that it intersects have span at most $6^d(\ell + \tau)^d$. So we can find a disjoint subset of narrow regions of size at least $\tau C_d/(2(12)^d\lambda\Delta)$, using a simple greedy algorithm that repeatedly chooses a narrow region that does not intersect any previously chosen regions. If we can only find one such region then $\tau = n \in O(\lambda\Delta)$ and we stop. Otherwise, if we remove this ply-one subset and repeat with the remaining narrow regions for at most $4\lambda^{d/(d+1)}\Delta(12)^d/C_d$ rounds, we

construct a set of at least $2\tau/\lambda^{1/(d+1)}$ regions whose collective ply is at most $4\lambda^{d/(d+1)}\Delta(12)^d/C_d$. □

Note that line 7 sets aside entities whose collective ply must be *added* to the ply we obtain by querying the remaining entities. This is in contrast to the entities we set aside in line 8, whose projected uncertainty regions cover a space that has ply less than h, no matter how the other entities are queries.

LEMMA 2.7. *Line 8 in algorithm* NOTIMETOLOSE! *can be implemented with a heaviness threshold h of at most $6^d\lambda^{d/(d+1)}\Delta/C_d$.*

Proof: After line 7 of the algorithm, all remaining entities have a projected uncertainty region with diameter at most τ. By Lemma 2.3, with $t = t^*$ and $h = 6^d\lambda^{d/(d+1)}\Delta/C_d$, at most $\tau/\lambda^{1/(d+1)}$ uncertainty regions intersect h-heavy cells. □

Thus, the regions set aside in line 8 cover a space that has final ply at most $6^d\lambda^{d/(d+1)}\Delta/C_d$ no matter how the remaining entities are queried. After line 8, we still have $2\tau/\lambda^{1/(d+1)}$ time and $\tau/\lambda^{1/(d+1)}$ entities remaining. Hence, we can use algorithm PLENTYOFTIME. The intrinsic ply of the remaining entities is at most Δ (since they form a subset of $\mathcal{E}$), so PLENTYOFTIME yields a set of uncertainty regions at time t^* with ply at most $(24)^d\Delta/C_d$. We conclude:

LEMMA 2.8. *Algorithm* NOTIMETOLOSE! *with n entities that have been last queried, on average, at time $t^0 - \ell$ and have intrinsic ply Δ at time $t^* = t^0 + \tau$ yields uncertainty regions at time t^* with ply $O(\min\{n/\Delta, \lambda\}^{d/(d+1)})\Delta$, where $\lambda = \left(\frac{\ell+\tau}{\tau}\right)^d$.*

2.3 A competitive algorithm for all cases

We now extend the algorithm to an arbitrary amount of time. The focus is on identifying a suitable subset of entities to query, such that (*i*) their projected uncertainty regions have a ply that is not too high, and (*ii*) we end up in a situation where we can use one of our previous algorithms.

LEMMA 2.9. *The set $\mathcal{J}$ of entities chosen in line 3 of* MINIMISEPLY *contribute at most Δ to the final ply.*

Proof: Let $\Delta_{\mathcal{J}}$ be the ply of the projected uncertainty regions of the entities in $\mathcal{J}$. Note that any uncertainty realisation of the entities at time t^* must include at least $n - \tau$ regions that coincide with initial projected uncertainty regions (since the associated entities were never queried). The ply of this collection must be at least $\Delta_{\mathcal{J}}$, by construction. But it must also be no more than Δ, the intrinsic ply of the full set of entities at time t^*. □

THEOREM 2.10. *Algorithm* MINIMISEPLY *with n entities that have been last queried, on average, at time $t^0 - \ell$ and have intrinsic ply Δ at time $t^* = t^0 + \tau$ yields uncertainty regions at time t^* with ply*

$$\begin{cases} O(1)\Delta, & \text{if } \tau \geq 2n, \\ O(\min\{n/\Delta, \lambda, \alpha^{d+1}\}^{d/(d+1)})\Delta, & \text{if } \tau = n + n/\alpha \text{ and } 1 < \alpha \leq n, \\ O(\min\{n/\Delta, \lambda\}^{d/(d+1)})\Delta, & \text{otherwise (i.e. if } \tau \leq n), \end{cases}$$

where $\lambda = \left(\frac{\ell+\tau}{\tau}\right)^d$.

Algorithm MINIMISEPLY($\mathcal{E}, \tau$)
Input. A set $\mathcal{E}$ of n entities and τ remaining time; ℓ is the average time since the last query over all entities.
▷ *The average diameter of the projected uncertainty regions is $\ell + \tau = \lambda^{1/d}\tau$.* ◁
1. **if** $\tau \geq 2n$ **then** Call PLENTYOFTIME($\mathcal{E}, \tau$).
2. **else if** $\tau \leq n$
3. Choose a set $\mathcal{J}$ of $n - \tau$ entities whose projected uncertainty regions have minimum collective ply. ▷ *In fact, a set with ply that is within a constant factor of the minimum ply will suffice.* ◁
4. Call NOTIMETOLOSE!($\mathcal{E} \setminus \mathcal{J}, \tau$).
5. **else**
6. Let $\alpha = n/(\tau - n)$. ▷ *Note: $1 < \alpha \leq n$ and $\tau = n + n/\alpha$.* ◁
7. **if** $\lambda < \alpha^{d+1}$ **then**
8. Query the $\tau - n$ entities with largest projected uncertainty regions. ▷ *Now we have n entities with remaining time $\tau' = n$ and with projected uncertainty regions of average diameter at most $\lambda^{1/d}\tau < 2\lambda^{1/d}\tau'$.* ◁
9. Call NOTIMETOLOSE!($\mathcal{E}, \tau'$).
10. **else**
11. Query all entities in $\mathcal{E}$ once. ▷ *Now the maximum diameter of the regions is τ.* ◁
12. Form a τ-partition of $\mathbb{R}^d$, and choose the smallest h so that at most $n/(2\alpha)$ regions intersect h-heavy cells.
13. Let $\mathcal{E}' \subseteq \mathcal{E}$ be the entities whose projected uncertainty regions intersect h-heavy cells.
14. Call PLENTYOFTIME($\mathcal{E}', n/\alpha$).

Proof: If $\tau \geq 2n$, the result follows from Lemma 2.4. If $\tau \leq n$, the result follows from Lemmas 2.8 and 2.9. Otherwise, suppose that $\tau = n + n/\alpha$, for some α, $1 < \alpha \leq n$. If α is relatively large ($\lambda < \alpha^{d+1}$), we reduce to the case when $\tau = n$ by querying the $\tau - n$ entities with the largest initial projected uncertainty regions, in any order, producing an instance of the original problem with n entities at time $\tau' = n$ whose projected uncertainty regions have an average diameter of at most $\lambda^{1/d}\tau < 2\lambda^{1/d}\tau'$. A direct application of algorithm NOTIMETOLOSE! from the previous section produces a solution with guaranteed competitive ratio $O(\min\{n/\Delta, \lambda\}^{d/(d+1)})$.

If α is relatively small ($\lambda \geq \alpha^{d+1}$), we reduce to the case when $\tau \geq 2n$. We query all entities once, leaving n/α time remaining, so that the maximum region diameter is $\tau = n + n/\alpha$. We chose $h = (24\alpha)^d\Delta/C_d$ and appeal to Lemma 2.3 to keep at most $n/(2\alpha)$ entities. The entities that we set aside have projected uncertainty regions that cover a space with ply at most h, no matter how the remaining entities are queried. □

OBSERVATION 2.11. *The bounds on competitive ratio hold when the entities move at any bounded speed.*

3. A LOWER BOUND ON THE COMPETITIVE RATIO

We now prove that the competitive ratios obtained by our algorithms are within a constant factor of the optimum. First, note that any query strategy with any set of entities with intrinsic ply Δ can only achieve final ply $\Omega(1)\Delta$, so we

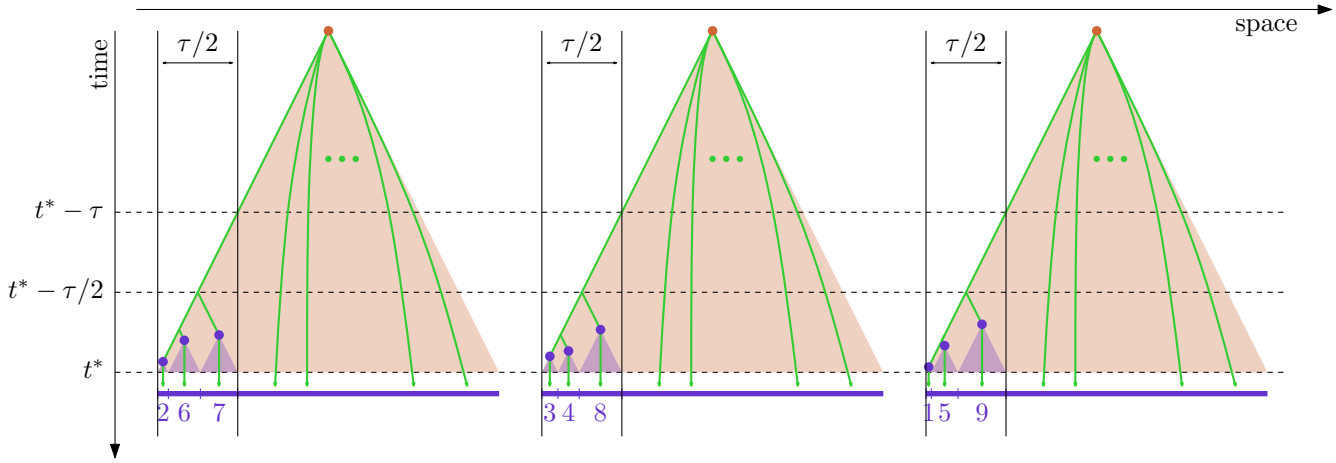

Figure 4: Illustrating the construction of Theorem 3.2, for $d = 1$, $s = 3$ and $m = 3$. The trajectories of the special entities coincide with the left edge of the triangles up until time $t = t^* - \tau/2$.

will focus on the case $\tau < 2n$. Let $\tau = n + n/\alpha$ for $\alpha > 1$. (If $\tau < n$, we rely on the lower bound for $\tau = n$.) We describe a set of entities $\mathcal{E}$, all with projected uncertainty regions of diameter $\ell + \tau$, such that any query strategy can be forced, by relabeling entities with identical regions, to achieve final ply $\Omega(\kappa^{d/(d+1)})\Delta$ where $\kappa = \min\{n/\Delta, \lambda, (\alpha/16)^{d+1}\}$ and $\lambda = \left(\frac{\ell+\tau}{\tau}\right)^d$.

We construct a collection of $s = n/(\kappa\Delta)$ sets $E_1, \ldots, E_s$ (since $\kappa\Delta \leq n$, there is at least one), each consisting of $\kappa\Delta$ entities, where the entities in E_i all have the same projected uncertainty region R_i, of diameter $\ell + \tau$ (where $\ell + \tau = \lambda^{1/d}\tau \geq \kappa^{1/d}\tau$ by choice of κ), that is disjoint from all other regions R_j, $j \neq i$. We assign regions of diameters $1, 2, \ldots, n$ to the n entities, that realise the intrinsic ply Δ, so that each set is assigned $m = \Delta(\kappa/2)^{d/(d+1)}$ *small* regions. (A region is small if it has diameter at most $ms = n/(\kappa 2^d)^{1/(d+1)}$.) We pack the small regions into the "leftmost" $n/2 < \tau/2$-diameter portion of R_i. This is possible since each small region has volume at most $c(ms)^d$ (for some constant c), so the sum of the volumes of all the small regions assigned to E_i is at most $cm^{d+1}s^d$, which fit with ply Δ in a volume of at most $c'm^{d+1}s^d/\Delta = c'(n/2)^d$ (for some constant c'), which has diameter $n/2$.

In each set E_i, the trajectories followed by entities associated with small regions (*special* entities) are identical up until $\tau/2$ time remains. (In particular, they all contain the "leftmost" point in R_i. See Figure 4 for an illustration for the case $d = 1$.) Thus any of the first $\tau/2$ queries is unable to distinguish these special entities. Let $\mathcal{E} = \bigcup_{i=1}^{s} E_i$.

LEMMA 3.1. *Let A be any query strategy, and let p be the maximum ply, over relabeling of entities with identical projected uncertainty regions, achieved by executing A on $\mathcal{E}$. If $\kappa > 8^{d+1}$ then $p > \kappa^{d/(d+1)}\Delta/8$.*

Proof: Let a_i and b_i be the number of queries to distinct entities in the set E_i upto and after time $\tau/2$, respectively, made by a query strategy. Since the projected uncertainty regions associated with entities in set E_i are initially identical, we can force, by suitable relabeling of entities, the first m queries made to distinct entities in E_i to be to special entities. The final ply obtained by the strategy for set E_i is at least

$$\kappa\Delta - ((a_i \dot{-} m) + b_i),$$

where $x \dot{-} y \equiv \max\{x - y, 0\}$, since any query made to a special trajectory before time $\tau/2$ does not help to reduce the final ply at the "leftmost" point of R_i. Then

$$\sum_{i=1}^{s} \kappa\Delta - ((a_i \dot{-} m) + b_i) \leq ps.$$

This inequality is equivalent to

$$\sum_{i=1}^{s} (a_i \dot{-} m) \geq n - \sum_{i=1}^{s} b_i - ps. \tag{1}$$

The number of sets with $a_i \geq m$ is at least the number of sets with $b_i \leq \kappa\Delta - p - m$, otherwise the number of unqueried entities in a set exceeds the final ply p. Since the number of sets with $b_i > \kappa\Delta - p - m$ is less than $\frac{\tau}{2(\kappa\Delta-p-m)}$, the number of sets with $a_i \geq m$ is greater than

$$s - \frac{\tau}{2(\kappa\Delta - p - m)}.$$

Thus,

$$\sum_{i=1}^{s} (a_i \dot{-} m) < \sum_{i=1}^{s} a_i - m\left(s - \frac{\tau}{2(\kappa\Delta - p - m)}\right). \tag{2}$$

Since $\sum_{i=1}^{s}(a_i + b_i) \leq \tau$, equations (1) and (2) together imply,

$$ps > n - \tau + m\left(s - \frac{\tau}{2(\kappa\Delta - p - m)}\right).$$

Since $s = n/(\kappa\Delta)$ and $\tau = n(1 + 1/\alpha)$, this is the same as

$$\frac{p}{\kappa\Delta} > -\frac{1}{\alpha} + m\left(\frac{1}{\kappa\Delta} - \frac{1 + 1/\alpha}{2(\kappa\Delta - p - m)}\right)$$

which, after substantial rearranging, is the same as

$$p(1 - 1/\alpha) - \frac{p^2}{\kappa\Delta} > -\frac{\kappa\Delta}{\alpha} + \frac{m}{2}(1 + 1/\alpha) - \frac{m^2}{\kappa\Delta}.$$

This implies

$$p > -\frac{\kappa\Delta}{\alpha} + \frac{m}{2} - \frac{m^2}{\kappa\Delta}.$$

Since $m = \Delta(\kappa/2)^{d/(d+1)}$ and $\alpha \geq 16\kappa^{1/(d+1)}$,

$$p > -\frac{\kappa^{\frac{d}{d+1}}}{16}\Delta + \left(\frac{\kappa}{2}\right)^{\frac{d}{d+1}}\frac{\Delta}{2} - \left(\frac{\kappa}{2}\right)^{\frac{d-1}{d+1}}\frac{\Delta}{2},$$

which is the same as,

$$p > \frac{\kappa^{d/(d+1)}\Delta}{8}\left(-\frac{1}{2} + \frac{4}{2^{d/(d+1)}} - \frac{4}{\kappa^{1/(d+1)}2^{(d-1)/(d+1)}}\right),$$

and so $p > \kappa^{d/(d+1)}\Delta/8$ if $\kappa > 8^{d+1}$. □

We conclude:

THEOREM 3.2. *There exists a set of n entities that have intrinsic ply Δ at time $t^* = t^0 + \tau$ with uncertainty regions at time t^* of diameter $\ell + \tau$ such that any query strategy can be forced, by relabeling of entities with identical projected uncertainty regions, to achieve final ply (where $\lambda = \left(\frac{\ell+\tau}{\tau}\right)^d$):*

$$\begin{cases} \Omega(1)\Delta, & \text{if } \tau \geq 2n, \\ \Omega(\min\{n/\Delta, \lambda, \alpha^{d+1}\}^{d/(d+1)})\Delta, & \text{if } \tau = n + n/\alpha \text{ and } 1 < \alpha \leq n, \\ \Omega(\min\{n/\Delta, \lambda\}^{d/(d+1)})\Delta, & \text{otherwise (i.e. if } \tau \leq n). \end{cases}$$

4. HARDNESS OF THE SINGLE SHOT PROBLEM

In this section, we show that it is NP-hard to compute the intrinsic ply of a collection of entities at a fixed point in time, even if the entities are 1-dimensional points, and we know their trajectories in advance. We first show that the following, simpler, problem is NP-hard. We refer to this problem as GRACEFUL SEGMENT COVER.

PROBLEM 4.1. *Given a set X of n numbers (points in $\mathbb{R}^1$), compute n disjoint intervals of lengths 1 to n, such that the mid point of each interval is a point in X.*

Figure 5 shows an instance of this problem. We now prove:

THEOREM 4.2. *The decision version of* GRACEFUL SEGMENT COVER *is NP-complete.*

Proof: We give a reduction from CNF-SAT. That is, given a CNF formula ϕ, we construct an instance of GRACEFUL SEGMENT COVER that has a solution if and only if ϕ is satisfiable. To avoid fractions, we build a GRACEFUL SEGMENT COVER that is scaled by a factor of 2, that is, the intervals have radii $1, .., n$, and diameters/lengths $2, 4, 6, .., 2n$. We refer to the intervals by their radius, so interval i is the interval of radius i.

The main idea in the reduction is to make gadgets that force any solution to use the 3 or 4 smallest intervals left available. That is, the first gadget we build consists of, say, 3 points, and can only be satisfied by mapping the intervals 1, 2 and 3 to those three points, in some permutation. Then the next gadget might consists of 4 points and can only be satisfied by using intervals 4 to 7. In general, we let S_i be the total number of intervals used by the first i gadgets, and then the $i+1$th gadget will use intervals S_i+1, S_i+2, S_i+3, and possibly $S_i + 4$. Since the S_i part will be present in all points of the $i + 1$th gadget, we can conceptually ignore it, and we will build all gadgets using only intervals of radii 1, 2, 3, and possibly 4.

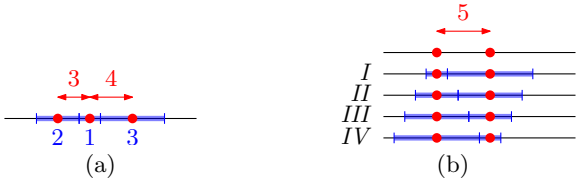

Figure 6: (a) A wall. (b) Half of a jump gadget.

Basic gadgets.

The first gadget we need is the *wall gadget*. It consists of three consecutive points, with gaps of size 3 and 4. The only way to satisfy this gadget is by mapping interval 2 to the first point, interval 1 to the second point, and interval 3 to the third point. Figure 6(a) illustrates this. The wall gadget can only be satisfied in one way, therefore, it creates a fixed region in $\mathbb{R}$ that is filled, and needs to be avoided by all other gadgets.

The second gadget we need is the *jump gadget*. It consists of two pairs of consecutive points. In both pairs (p, q) the distance (gap) between p and q is 5, but the pairs themselves can be arbitrary far away from each other. There are several ways to satisfy this gadget, but since $1 + 2 + 3 + 4$ equals the sum of the gaps (two times 5), any solution must use intervals 1 to 4, and furthermore, intervals 1 and 4 must be used together at one of the pairs, and 2 and 3 at the other pair. Figure 6(b) illustrates this. Note that the intervals used in one pair together use 10 space.

Composite gadgets.

Let ϕ be a CNF-SAT instance, consisting of V, a set of variables; C, a set of clauses; and $E \subset V \times C \times \{true, false\}$, a set of occurrences of variables in clauses, possibly negated. We also call E the edges of the instance. We will create a jump gadget for each element of E, and one more for each variable of V. We construct variable and clause gadgets on consecutive segments of $\mathbb{R}$, which are separated by wall gadgets. Hence, we use $|V| + |E|$ jump gadgets and $|V| + |C| + 1$ wall gadgets in total.

Let $v \in V$ be a variable of the SAT-instance, and suppose there are k edges in E that use v. We start by making a gap of length $10k+21$ between two walls for the variable gadget. Inside, we place a special jump gadget; one pair at distance 2 from the left wall and one pair at distance 2 from the right wall. Then for each $(v, c_i, b_i) \in E$, we place one pair of the jump gadget (the other pair will be in the clause gadget) at distance $10i + 2$ from the left wall if $b = true$, or at distance $10i + 4$ if $b = false$. Figure 7(a) illustrates the construction.

There are $k+2$ half jump gadgets in the construction, each of which will take up 10 space. The total space is $10k + 21$, which leaves one unit empty. The outer two jump gadgets that are linked to each other ensure that we cannot build all the way to the left and right wall simultaneously; therefore, there are only two possible states that satisfy the variable gadget: either the leftmost unit is empty or the rightmost unit is empty. We use these to encode the states *true* and *false* of the variable. Once this choice is made, the way to fill in all the jump gadgets is fixed. Note that for each edge (v, c, b), we use intervals 1 and 4 if the variable is set to true and $b = true$ or if the variable is set to false and $b = false$, and intervals 2 and 3 otherwise.

Let $c \in C$ be a clause of the SAT-instance, and suppose

Figure 5: (a) An example input for Graceful Segment Cover. (b) A possible solution. (The lengths of the intervals and gaps are scaled by a factor 2 to avoid fractions; e.g. the blue interval labeled '4' has length 8; 4 on each side of the point.)

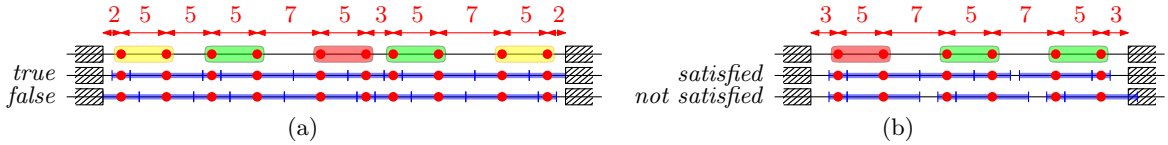

Figure 7: (a) A variable. The first and last (yellow) jumpers are linked to each other to ensure there are only two possible states. Green jumpers correspond to positive literals of the variable, and are linked to clauses where the variable appears positively. Red jumpers correspond to negative literals, and link to clauses where the variable appears negatively. (b) A clause.

there are k edges in E that use c. We make a gap of length $12k-1$ between two walls for the clause gadget. Inside, we place the k half jump gadgets, each gadget i at distance $12i-9$ from the left wall. Figure 7(b) illustrates the construction. Recall that each jump gadget in a variable uses intervals 1 and 4 if its sign matches the state of the variable, and 2 and 3 otherwise. This means that the other halves of the gadget, in the clause, must use intervals 2 and 3 if the literal is true, and 1 and 4 if it is false. If all literals in the clause are not satisfied, then there is no solution, since somewhere two intervals 4 would need to overlap. If at least one of the literals is true, then there is enough space for any assignment of the other literals.

Finally, to complete the proof, we need to consider the actual lengths again, including the S_i factors that we ignored so far. Recall we used $2|V|+|C|+|E|+1$ gadgets. We order and number them arbitrarily. Now, for any gap in the construction between two points belonging to gadgets i and j, we increase the gap size by S_i+S_j. □

Now, we can prove hardness of the original problem.

THEOREM 4.3. *Let $\mathcal{E}$ be a set of n entities, let F be their associated trajectories, and let t be a timestamp. Computing the intrinsic ply of $\mathcal{E}$ at time t is NP-hard.*

Proof: We reduce from GRACEFUL SEGMENT COVER. Consider an input set X of n points in $\mathbb{R}$. Now simply place an entity on position x for every input number x in X, and keep the entities at their starting position. An optimal solution must query the entities in some order, and will result in a set of intervals of lengths 1 to n. Since the entities are stationary, these intervals will be centered at the corresponding points. To test whether a solution of ply 1 is possible it is necessary to decide if there exists a placement of disjoint intervals of lengths 1 to n, centered at the points in X. The theorem follows. □

5. A COMPETITIVE ALGORITHM FOR THE RECURRENT PROBLEM

Recall that in the recurrent version of ply minimisation the goal is to minimise the ply of the uncertainty regions at regularly spaced checkpoints. As before, we will address this in a competitive framework, by trying to minimise, at every checkpoint, the competitive ratio of the ply attained by our algorithm with the intrinsic ply of the trajectories at that point. Since the intrinsic ply corresponds to the minimum ply achievable at one single point in time by an algorithm that has complete knowledge of the trajectories, we are in effect contrasting the results achievable by our algorithm with those achievable by a family of fully informed algorithms designed to optimise the ply at each checkpoint separately.

As it turns out a simple modification of our single shot strategy works remarkably well: it is $O(1)$-competitive, when the checkpoint spacing is any constant fraction of the number of entities. To be precise, suppose that we have n entities and we want to minimise the ply of their associated uncertainty regions at regular checkpoints, spaced 2τ time units apart. Our algorithm interleaves two processes, RoundRobin and SingleShot, running in odd and even time steps respectively.

RoundRobin: query all entities in a round-robin fashion.

SingleShot: after each checkpoint follow our general single shot ply minimisation algorithm MinimisePly with the goal of minimising the ply at the next checkpoint.

Since RoundRobin queries every entity once every $2n$ time steps, we know that, after at most $2n$ time steps (or right from the the start, if the initial uncertainty regions are of size at most $2n$), the diameter of all uncertainty regions, projected to the next checkpoint, never exceeds $2(n+\tau)$. Furthermore, since SingleShot executes on even time steps, its behaviour is indistinguishable from that of algorithm MinimisePly, when entities are moving at two times the normal speed. Thus, by Observation 2.11, we know that a competitive ratio of $O(\min\{n/\Delta,\lambda\}^{d/(d+1)})$ is always achievable, where $\lambda^{1/d}\tau \le 2(n+\tau)$, or equivalently $\lambda \le (2(n+\tau)/\tau)^d$. We summarise this result in the following:

THEOREM 5.1. *The interleaved* RoundRobin–SingleShot *algorithm guarantees a ply at most a factor $O((2k+2)^{d^2/(d+1)})$ larger than the intrinsic ply at all checkpoints, provided that the checkpoint spacing τ satisfies $\tau \ge n/k$.*

6. CONCLUDING REMARKS

We studied a new model for dealing with unpredictable motion in which obtaining the location of a moving entity costs a certain amount of time. We developed algorithms to keep track of the positions of moving entities in this model by minimising the ply of their uncertainty regions. We showed that our algorithms are within a constant factor of the optimal solution.

Interesting directions for future work include extending this work to deal with uncertainty in time (e.g. what if we want to minimise the ply of the uncertainty regions at an unknown time $t^* \in [t^-, t^+]$), and to handle varying query time costs. In particular, what happens if the time cost of a query depends on the running time of the querying algorithm. Another possible extension would be to allow multiple queries to be executed simultaneously.

Acknowledgements

This work has been partially supported by the Netherlands Organisation for Scientific Research (NWO) under grants 639.021.123 and 612.001.022, and by NSERC of Canada.

7. REFERENCES

[1] P. K. Agarwal, J. Erickson, and L. J. Guibas. Kinetic BSPs for intersecting segments and disjoint triangles. In *Proc. 9th ACM-SIAM Symp. Discrete Algorithms*, pages 107–116, 1998.

[2] J. Almeida and R. Araujo. Tracking multiple moving objects in a dynamic environment for autonomous navigation. In *Advanced Motion Control, 2008. AMC'08. 10th IEEE International Workshop on*, pages 21–26, 2008.

[3] J. Basch, L. J. Guibas, C. Silverstein, and L. Zhang. A practical evaluation of kinetic data structures. In *Proc. 13th ACM Symp. Comput. Geom.*, pages 388–390, 1997.

[4] R. Bruce, M. Hoffmann, D. Krizanc, and R. Raman. Efficient update strategies for geometric computing with uncertainty. *Theory of Computing Systems*, 38(4):411–423, 2005.

[5] K. Buchin, M. Löffler, P. Morin, and W. Mulzer. Delaunay triangulation of imprecise points simplified and extended. *Algorithmica*, 61(3):674–693, 2011.

[6] M. Cho, D. M. Mount, and E. Park. Maintaining nets and net trees under incremental motion. In *Proc. 20th International Symposium on Algorithms and Computation*, ISAAC '09, pages 1134–1143. Springer, 2009.

[7] M. de Berg, M. Roeloffzen, and B. Speckmann. Kinetic convex hulls and Delaunay triangulations in the black-box model. In *Proc. 27th ACM Symp. on Comput. Geom.*, pages 244–253, 2011.

[8] S. B. Eisenman. *People-centric mobile sensing networks*. PhD thesis, Columbia University, New York, NY, USA, 2008.

[9] D. Eppstein, M. T. Goodrich, and M. Löffler. Tracking moving objects with few handovers. In *Proc. 12th Algorithms and Data Structures Symposium*, pages 362–373, 2011.

[10] P. G. Franciosa, C. Gaibisso, G. Gambosi, and M. Talamo. A convex hull algorithm for points with approximately known positions. *International Journal of Computational Geometry and Applications*, 4(2):153–163, 1994.

[11] J. Gao, L. Guibas, and A. Nguyen. Deformable spanners and their applications. *Computational Geometry: Theory and Applications*, 35:2–19, 2006.

[12] L. Guibas, J. Hershberger, S. Suri, and L. Zhang. Kinetic connectivity for unit disks. In *Proc. 16th ACM Symp. Comput. Geom.*, pages 331–340, 2000.

[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] M. Hoffmann, T. Erlebach, D. Krizanc, M. Mihalák, and R. Raman. Computing minimum spanning trees with uncertainty. In *STACS*, pages 277–288, 2008.

[15] S. H. Kahan. *Real-Time Processing of Moving Data*. PhD thesis, University of Washington, 1991.

[16] M. Löffler and M. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010.

[17] G. Miller, S. Teng, W. Thurston, and S. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *Journal of the ACM*, 44(1):1–29, 1992.

[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 ACM Symp. on Comput. Geom.*, pages 200–209, 2004.

[19] M. Schneider. Moving Objects in Databases and GIS: State-of-the-Art and Open Problems. *Research Trends in Geographic Information Science*, pages 169–187, 2009.

[20] K. Sreenath, F. L. Lewis, and D. O. Popa. Simultaneous adaptive localization of a wireless sensor network. *SIGMOBILE Mob. Comput. Commun. Rev.*, 11(2):14–28, 2007.

[21] K.-C. R. Tseng and D. G. Kirkpatrick. Input-thrifty extrema testing. In *ISAAC*, pages 554–563, 2011.

[22] J. van den Berg and M. Overmars. Planning time-minimal safe paths amidst unpredictably moving obstacles. *International Journal of Robotics Research*, 27(11–12):1274–1294, 2008.

[23] K. Yi and Q. Zhang. Multi-dimensional online tracking. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 1098–1107. SIAM, 2009.

[24] C. Zhu, L. Shu, T. Hara, L. Wang, and S. Nishio. Research issues on mobile sensor networks. In *Communications and Networking in China (CHINACOM), 2010 5th International ICST Conference on*, pages 1–6. IEEE, 2010.