The University of British Columbia (UBC)

A Model for Computing in Dynamic, Resource-Limited Environments

Ben Chugg

April 2018

Abstract

This thesis is focused on the problem of developing and analyzing algorithms in environments in which there is inherent uncertainty in the input data. Differentiating it from most previous work in this area is the fact that we are actively choosing which data contain the most uncertainty and which the least. Moreover, we are interested in maintaining solutions (or approximations thereof) to problems over some interval of time during which the data are continually changing in ways unknown to us. Therefore, any precision we have regarding the current state of the system degrades over time, and we must choose which data to "query" in order keep our solution up-to-date. Exacerbating the difficulty is the fact that we allow only a constant amount of the input to be queried every time step. We present a model enabling us develop and analyze algorithms operating in such environments, and develop optimal solutions for several specific problems.

Contents

List of Algorithms

List of Figures

Acknowledgements

When one searches Google for "whether acknowledgements should be included in a Bachelor of Science thesis", it quickly becomes apparent that there is no broad consensus on the topic. At the risk, therefore, of making an academic faux pas, I will thank several people.

I would like to express my sincere gratitude to my supervisor, Will Evans, who introduced me to this problem. His infectious intrigue in any of the problems discussed in this thesis was highly motivating, especially when my rate of progress was somewhat suspect. I hope to bring this same energy to any problem I encounter in the future.

Secondly, if the podcasts to which one listens does not contribute to the success of their thesis in theoretical computer science, then I don't know what does. In particular, the "Very Bad Wizards" podcast was a welcome reprieve from being confused about math on the bus ride home.

Somewhat more seriously, thank you, of course, to my parents, who for some reason still seem to support me even though I can rarely explain what it is I'm working on. Finally, I thank Emilie McConnachie for four years of smiles and laughter.

Chapter 1

Introduction

Traditional algorithm design and analysis takes place in scenarios in which the input is assumed to be unchanging over time. We assume, for example, when we are sorting a list of numbers that the numbers remain fixed throughout the procedure—that a "1" will not suddenly change to a "2". This assumption, especially in the past, has typically been well justified and pragmatic. One does not expect the biological data of a patient to vary from one time step to the next, nor the cities on a map to move when computing the shortest route between them.

Sometimes, however, this assumption is not well justified. As data sets grow and their scope becomes ever more discriminating, it is increasingly common to encounter scenarios in which the data are moving with time. That is, the environment is *dynamic*. Algorithms implemented by GPS systems, for example, compute over datasets representing the spatial coordinates of their targets. These coordinates are changing in real-time, thus requiring that these algorithms are sensitive to changing data. As another example, the systems embedded within autonomous vehicles must react to their immediate environment (both for their sake and the sake of pedestrians) which is rapidly changing as the vehicle moves. More generally, as the number of autonomous devices increases and consequently the number of signals sent between them, it will important to understand how to design efficient algorithms which operate well in this dynamic environment.

A crucial constraint in many of these environments is that it is often highly infeasible to maintain perfect knowledge of the state of all incoming data at all times. That is, the scenarios may be *resource-limited*: only the precise state of a fraction of the true data can be known to the algorithm at any one time. This imprecision may due to computational intractability of the environment, or with communicational issues in the environment. In a GPS system, for example, it is much too computationally demanding to track the location of every one of your targets at each second. It is more feasible to track a target once every fixed number of time steps, and make an educated guess as to where they are in the meantime. Conversely, we may be in a situation in which there is delay in communicating with a device; a satellite, for example. This lack of perfectly up-to-date information results in uncertainty in the system.

If, in these dynamic, resource-limited (DRL) environments, only a subset of the data can be known precisely the question becomes: Which data points do we keep up-to-date in order to achieve the best possible solution? Which people or objects should we decide to track closely with our GPS system, and which are we happy with knowing only their approximate location? If we can only communicate with one satellite at time, which one do we choose? Our goal is to develop algorithms which perform well in DRL environments.

However, it is not immediately clear what is meant by "performing well" in this setting. Since any algorithm operating in a DRL environment will face inherent uncertainty in its input (and most likely its output), against what do we judge its performance? For example, we can judge a traditional sorting algorithm by whether it sorts its input and how fast it does so. However, if the numbers are changing and no algorithm can know all the numbers precisely at any time, it seems unfair to judge an algorithm's output as either "wrong" or "right". Moreover, these terms are only well-defined with respect to a specific point in time. The output could be sorted at an instant in time but then change and throw the whole sequence into disarray! The first question explored by this thesis therefore, is focused on defining an appropriate model for developing and analyzing algorithms in these environments. The second question is: For what problems can we develop efficient and well-performing algorithms?

If one is prone to pessimism, it may be hard to imagine how an algorithm can hope to accomplish anything meaningful in such a demanding environment. Problems are hard enough to solve when the input is time invariant, you might say, so how can we hope to succeed when it starts to change erratically. While this concern is well justified, the situation is often not as dire as the pessimist suggests. In his M.Sc. thesis [Bus15], Daniel Busto gives the following illuminating example. Suppose you are a pedestrian who is considering crossing the street (forgetting for the moment your moral stance on jaywalking), and do not see approaching cars. If you decide to cross then it is unnecessary to keep looking to your left and right to check for traffic. This is because knowing that there were no cars a moment ago is sufficient to guarantee that there can be no cars there now, because car speeds obey constraints (physics-based constraints if not mandated speed constraints).

Similarly, if the data obey certain constraints restricting how much they can change at a given time, it is often possible to make guarantees about the output even if we don't know the precise input values. Returning to our dynamic list of numbers, suppose we knew somewhat recently that the largest element in a given list of numbers is significantly larger than the rest. If we are guaranteed that values cannot change by, say, more than plus or minus one every time step, then we are guaranteed that this number will be the largest number in the list in the near future. Conversely, if the values of the numbers are grouped closely together then while it may be difficult to maintain an accurately sorted list, it would be any for *any* algorithm to maintain a sorted list. In this case, we can settle for knowing that we are performing as well as can be reasonably expected.

1.1 Contribution

In Section 2.3 we develop a general model in which to analyze algorithms in DRL settings. Here, we introduce the *impedance* of an algorithm, which, informally speaking, is a measurement of how well an algorithm performs. The impedance of an algorithm is measured as a function of both the true and perceived locations of the data points. This function, called the "evaluation measure" is specific to the problem at hand. Any algorithm having the same perceived location of the input data at a point in time will have the same impedance at this time. In this way therefore, the performance of an algorithm is measured solely by its query strategy. Two algorithms having the same query strategy will have the same impedance, and thus the same performance. We also give several different metrics for judging the intrusion of different algorithms. These metrics correspond to measuring the impedance of algorithms at a specific time (a "target" time), as an average over the course of a time interval, or as the maximum impedance over the course of an algorithm.

In Chapter 3 we explore what we believe is perhaps the first natural question to ask in this model. Namely, if we are given a function which takes as input the location of the data points, how closely may we approximate the value of this function? For example, if you want to compute the mean of your input points (suppose the points are on the real line), what is the optimal query strategy in order to maintain as close an approximation to the true mean as possible? This Chapter is largely focused on exploring for what kinds of functions a natural "Round-Robin" querying is the optimal strategy. We give sufficient conditions to ensures its optimality and explore several classes of functions on which it performs optimally.

Chapter 4 then demonstrates a fundamental relation between our model and a real time scheduling problem: pinwheel scheduling. In particular, we ask a natural question relating to limiting the size of a point's "uncertainty region" (a hypersphere centered at the point's last known location) representing the area in which the point could be, and cast this is a dynamic version of the pinwheel scheduling problem. Looking at this problem from the perspective of DRL environments, we are able to give an asymptotically optimal algorithm.

1.2 Related Work

It is natural to examine the scenario of data changing over time from a geometric standpoint. Given a set of inputs, each of which has parameters $\theta_1, \theta_2, \ldots, \theta_d$ where θ_i takes values in a set X_i, we can view a changing parameter as movement in the space

$$\mathcal{X} = \prod_{i=1}^{d} X_i.$$

If this space is assumed to be "nice" in any formal way (e.g., a metric space), then powerful tools of geometry and topology and can be applied. Consequently, the largest impact on the problem of changing data has come from the computational geometry community in the form of dynamic and kinetic algorithms.

Kinetic algorithms are typically concerned with maintaining a data structure containing elements which move over time. Algorithms are typically analyzed by how many updates the data structures must incur [Gui98]. Kinetic algorithms have been developed for many traditional problems, including sorting [AdB05], maintaining convex hulls [AKS07], and collision detection [KSS02]. Oftentimes in these algorithms, full knowledge of the trajectories of the points is assumed [Spe08]. This assumption makes this work highly distinct from our own problem. However, there is a branch of the kinetic algorithms literature which deals in the "black box" model, which removes this assumption. In this model, the locations of all points are reported at some fixed times $t_0, t_1, \ldots$ [dBRS11]. While this does not embody the spirit of our model precisely (as it lacks any notion of "querying" points), it does deal with the uncertainty of points. We consider this model akin to a close cousin of whom we are quite fond. Convex Hulls and Delauney Triangulations [dBRS11], computing Euclidean 2-centers [dBRS13] and s-spanners [GGN06] are all examples of problems studied under this model. See the survey by Chiang and Tamassia for an overview common techniques and applications in this area [CT92].

There has also been lots of work in *online algorithms* [Alb03], in which the data arrives as a series of updates. After every update, the algorithm seeks to produce the best solution with the input seen thus far. While this is similar to our model insofar as we are also aiming to produce the best output based on our (imprecise) knowledge of the data, there is typically no choice involved

in online algorithms as to which piece of input to receive next. This makes the model distinct from our own, but techniques from this area still prove useful. Our, model for example, will rely on the concept of *competitive analysis* [ST85, BIL09] to analyze the performance of our algorithms—a concept originating with online algorithms.

1.2.1 Data in Motion

In his influential Ph.D thesis, Simon Kahan introduced a model for studying "data in motion" [Kah91]. In Kahan's model, the input is a set of "objects" in a metric space whose locations are changing in some fashion unbeknownst to an observer. At some point in time, we are asked to compute some function which depends on the locations of the objects. An algorithm can request an object's true location by "querying" the object, and an algorithm's efficiency is measured in part by how many queries it makes. Kahan's model has been widely used (e.g., [ABDB+00, JNRS00]) and has served as influence for other models [EHK+08]. Our model as well will draw a lot of inspiration from Kahan's.

1.2.2 Ply and Interference Minimization

The first problem relating directly to this model was the problem of ply minimization studied originally by Evans et al. [EKLS13, EKLS16]. In this setting, the input is a set of entities moving in a metric space, each with some bounded speed. Only one entity can be queried each time step (thus simulating a resource limited setting), and each time an entity is queried its current location is revealed. At any given time therefore, an entity could be anywhere in the hypersphere (its uncertainty region, as mentioned above) centered at its last known location with radius proportional to time since it was last queried. The goal is to develop a query strategy which minimizes the overlap of the uncertainty regions of the entities (the ply) at the target time. They give an asymptotically tight bound on the ply achieved by their algorithm. In his master's thesis, Daniel Busto introduces the problem of *continuous* ply minimization [Bus15]. That is, minimizing the ply at all times instead of only at a specified target time. An optimal algorithm was given for the special case in which a "target" value δ was given on the play. The algorithm would either maintain ply δ at all times, or fail, certifying that at some time there was a particularly bad configuration of the points and any algorithm would have admitted ply $\Omega(\delta)$. This algorithm was the "bucket algorithm", which we will put to use in Chapter 4. Recently, Busto et al. gave asymptotically algorithms for the general version of the problem [BEK18].

1.2.3 Real Time Scheduling and the Pinwheel Problem

As Chapter 4 relates our model to Pinwheel scheduling we believe it prudent to give some background on the problem.

Real time scheduling problems are concerned with scheduling a set of tasks which occur periodically, with differing deadlines, on a single processor [SAA+04]. In their seminal paper, Liu and Layland [LL73] began working on scheduling problems obeying certain assumptions, including: 1) Tasks are periodic; 2) they must be completed once per period; 3) all tasks are independent; 4) they have a fixed computation time; 5) they must all be scheduled onto a single processor. See [SAA+04] for a survey of the main results. This model has been extremely influential and has been the focus of much attention [Cer03, Tab07, ABRW91, DL78, LSST02, DB11].

A specific example of a real time scheduling problem is the pinwheel scheduling problem, which was first introduced by Holte et al. [HMR+89] and was motivated by satellite transmission. Satellites typically broadcast their messages for some set duration, after which point they move onto their next message. A ground station which is in contact with multiple satellites must therefore "ping" each satellite within a given time frame to receive each message. Assuming that a station cannot ping multiple satellites simultaneously (for computational reasons perhaps), in what order should the satellites be pinged as to avoid data loss? Pinwheel scheduling can be formalized as follows. Given vector $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{N}^n$ a *pinwheel schedule* for $\mathbf{v}$ is an infinite sequence of elements of $\mathbf{v}$ such v_i is in the sequence at least once every v_i consecutive terms.

The *pinwheel decision problem* asks whether a given vector $\mathbf{v}$ admits a pinwheel schedule. In the first and seminal paper on the topic, Holte et al. prove several important results [HMR+89]. First they prove that the decision problem is in PSPACE and that if $\mathbf{v}$ is schedulable, then there exists a periodic schedule with period at most $\prod_i v_i$. They also give an algorithm, SimpleGreedy, which schedules $\mathbf{v}$ if its demand (i.e., $\sum_i 1/v_i$) is at most $1/2$ or if $v_i < v_j$ implies that $v_i | v_j$. Additionally, they show that for vectors with density equal to one the decision problem is NP-Complete if the input is represented using "repetition factors". In [HRTV92], Holte et al. demonstrate that any vector $\mathbf{v}$ with at most two distinct values and demand at most 1 is schedulable. It was subsequently shown by Chan and Chin that a bound of $2/3$ on the demand is a sufficient condition to guarantee a valid schedule [CC93], and this was later improved to a bound of $7/10$ by the same authors [CC92]. Chan and Chin conjecture that $5/6$ is smallest density which will guarantee schedulability. In 2002, Fishburn and Lagarius improved the bound to $3/4$ and also prove that for vectors with $v_i = 2$ for some i, that $\lambda(\mathbf{v}) \leq 5/6$ does guarantee that $\mathbf{v}$ is schedulable [FL02]. These appear to be the last known improvements on the decision problem. However, several papers have explored generalizations or applications of pinwheel scheduling [FC05, YLH04, LH06].

Chapter 2

Preliminaries

The main goal of this chapter is to develop the model needed to study problems in a dynamic, resource limited setting. This is given in Section 2.3. Beforehand, we give preliminary background.

2.1 Background and Notation

We use conventional notation for sets of real-valued numbers: $\mathbb{R}$ to denote the real numbers, $\mathbb{Q}$ for the rational, $\mathbb{C}$ for the complex, $\mathbb{Z}$ for the integers and $\mathbb{N}$ for the non-negative integers (including zero). For any $n \in \mathbb{N}$ we set $[n] \triangleq \{1, \ldots, n\}$. We will commonly use the shorthand "iff" to mean "if and only if". Upper case letters, e.g. A, B, etc, will be used primarily to denote sets. For a set A, we use A^c to denote the complement of A (with respect to which set will be understood from context). We use S_n to denote the set of permutations on n points, i.e., all bijections $\pi : [n] \to [n]$. Given a function $f : A \to B$, we use the usual notions of image and pre-image. Given $A' \subseteq A$, the *image* of A' is the set $f(A') \triangleq \{b \in B : f(a) = b \text{ for some } a \in A'\}$. Similarly, given $B' \subseteq B$, the *pre-image* of B' is $f^{-1}(B') \triangleq \{a \in A : f(a) = b \text{ for some } b \in B'\}$. The image of the function f is the image of the domain of f, $f(A)$.

The concepts will be rarely used, but it will be occasionally useful to refer to lower and bounds of a set of real-valued points and to the least or greatest of these bounds. Given a set $E \subseteq \mathbb{R}$, an upper bound of E is any point $u \in \mathbb{R}$ such that $u \geq x$ for all $x \in E$. The *supremum* (equivalently, the least upper bound) of E, denoted $\sup E$, is the (unique) value u^* such that $u^* \leq u$ for all upper bounds u of E. Similarly, a lower bound of E is any value ℓ such that $\ell \leq x$ for all $x \in E$ and the *infimum* (equivalently, the greatest lower bound) of E, $\inf E$, is the value ℓ^* such that $\ell^* \geq \ell$ for all lower bounds ℓ of E.

2.1.1 Metric and Vector Spaces

Given a non-empty set $\mathcal{X}$, a *metric* on $\mathcal{X}$ is a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that for all $x, y, z \in \mathcal{X}$: (1) (non-negativity) $d(x, y) \geq 0$ with equality iff $x = y$; (2) (symmetry) $d(x, y) = d(y, x)$; and (3) (triangle inequality) $d(x, y) \leq d(x, z) + d(z, y)$. If d is a metric on $\mathcal{X}$, the pair $(\mathcal{X}, d)$ is called a *metric space*.

Fix a metric space $(\mathcal{X}, d)$. Let $x \in \mathcal{X}$. The (open) *ball of radius r centered at x* is $\mathcal{B}_r(x) \triangleq \{y \in \mathcal{X} : d(x,y) < r\}$. We will let $\mathcal{O}(\mathcal{X}) \triangleq \{\mathcal{B}_r(x) : r \in \mathbb{R}, x \in \mathcal{X}\}$, be the set of all open balls in $\mathcal{X}$. Given an open ball $U \in \mathcal{O}(\mathcal{X})$, we will let $\text{rad}(U) = \sup_{x,y \in U} d(x,y)$ be its radius.

A *vector space* V over a field $\mathbb{F}$ [Rud64] (in our case $\mathbb{R}$ or $\mathbb{C}$), is a set of elements called *vectors* along with two operations $+ : V \times V \to \mathbb{F} : (\mathbf{v}, \mathbf{w}) \mapsto \mathbf{v} + \mathbf{w}$ (addition) and $\cdot : \mathbb{F} \times V \to V : (a, \mathbf{v}) \mapsto a\mathbf{v}$ (multiplication) such that for all $a, b \in \mathbb{F}$, $\mathbf{u}, \mathbf{v}, \mathbf{w} \in V$: (1) $\mathbf{v} + \mathbf{w} = \mathbf{w} + \mathbf{v}$; (2) $\mathbf{v} + (\mathbf{w} + \mathbf{u}) = (\mathbf{v} + \mathbf{w}) + \mathbf{u}$; (3) there exists $\mathbf{0} \in V$ such that $\mathbf{0} + \mathbf{v} = \mathbf{v}$; (4) for every $\mathbf{v} \in V$ there exists some $-\mathbf{v} \in V$ such that $\mathbf{v} + -\mathbf{v} = \mathbf{0}$; (5) $a(b\mathbf{v}) = (ab)\mathbf{v}$; (6) $1\mathbf{v} = \mathbf{v}$ where 1 is the multiplicative identity of $\mathbb{F}$; (7) $a(\mathbf{u} + \mathbf{v}) = a\mathbf{u} + a\mathbf{v}$; (8) $(a + b)\mathbf{v} = a\mathbf{v} + b\mathbf{v}$. For any vector $\mathbf{v}$ in a vector space, we will write v_i for the i-th coordinate of v.

A *norm* on a vector space V over the real or complex numbers is a function $\|\cdot\| : V \to \mathbb{R}$ such that for all $\mathbf{v}, \mathbf{u} \in V$ and $a \in \mathbb{F}$, (1) $\|\mathbf{v}\| \geq 0$ with equality iff $\mathbf{v} = \mathbf{0}$; (2) $\|\mathbf{v} + \mathbf{u}\| \leq \|\mathbf{v}\| + \|\mathbf{u}\|$, and (3) $\|a\mathbf{v}\| = |a|\|\mathbf{v}\|$. If $\|\cdot\|_V$ is a norm on V we call the pair $(V, \|\cdot\|_V)$ a *normed vector space*.

2.2 Competitive Analysis

Competitive analysis was introduced by Sleator and Tarjan [ST85] as a method for analyzing *online algorithms*: algorithms for which the input arrives over a window of time and the algorithm must base its decision at the current time step based only on input seen previously, without any knowledge of what is coming next. More formally, for an input sequence $p_1, \ldots, p_n$, for every $i \in [n]$ an online algorithm must produce an output given $p_1, p_2, \ldots, p_i$ without knowledge of $p_{i+1}, \ldots, p_n$. The caching problem is the archetypal example of an online algorithm. A sequence of page requests are made to the cache, and we must decide which pages to keep in the cache and which pages to evict (to disk) in order to maximize the number of "cache hits", i.e., the number of times to incoming page resides in the cache. The traditional method of "worst-case analysis" is absolutely uninsightful when applied to this problem: any algorithm can be made to incur n cache misses, where n is the number of pages requested. Indeed, an adversary may simply choose to request a page which is not in the cache at every step. In this way, if $f_{\mathscr{A}}(p_1, \ldots, p_n)$ is the number of cache misses of algorithm $\mathscr{A}$ on an input sequence $p_1, \ldots, p_n$ of pages, then

$$\max_{p_1, \ldots, p_n} f_{\mathscr{A}}(p_1, \ldots, p_n) = \Omega(n).$$

Thus, worst case analysis does not deliver any understanding of the difference between different algorithms for this problem. Competitive analysis was designed to circumvent this problem. Generalizing slightly from our cache example, let $f(I)$ be a function of the input sequence I and suppose we are interested in minimizing f. We say an algorithm $\mathscr{A}$ is *c-competitive* [BIL09] if

$$\sup_I \frac{f_{\mathscr{A}}(I)}{f_{\text{OPT}}(I)} \leq c.$$

In our model, competitive analysis will not be applied precisely as it is here. However, we believe it's important to present because the underlying ideas influence the analyses of our model.

2.3 Model

Fix a metric space $(\mathcal{X}, d)$ and let $P \triangleq [n]$ be a set of n *data points*, hereafter referred to as simply points, in $\mathcal{X}$. To simulate a dynamic environment, we assume the points P are moving in $\mathcal{X}$ over

some time interval $T = [t_0, t_m] \in \mathbb{R}$, where $t_0, t_m \in \mathbb{N}$.

The location of point $i \in [n]$ at every point in time is given by a continuous function $x_i : T \to \mathcal{X}$. As will become clear as the model is further expounded, for any algorithm to have a chance at performing well (concrete performance metrics will be given later), we will require that the "speed" of each point is bounded as it moves. Therefore, we assume that for each point i there exists a constant $\sigma_i \in \mathbb{R}^+$—the *speed bound* of point i—such that for all $t_1, t_2 \in T$

$$d(x_i(t_1), x_2(t_2)) \leq \sigma_i |t_2 - t_1|.$$

One might just as well provide a bound on the derivative on the x_i's, however we are not requiring that the functions be differentiable. Indeed, roughly speaking, our model permits the points to be traveling in one direction at time t, and at time $t+1$ to be traveling in the opposite direction at the same speed. In this sense, the model is very unrestrictive. That being said, an interesting extension to this model might be to consider speed bounds which change over time and require the functions to be differentiable. In this case, a speed bound might translate to a bound on the *curvature* of the functions. For the problems examined in this thesis, we will assume that $\sigma_i = 1$ for all i.

We investigate problems which involve computing some function over the set of points P. Furthermore, we assume that whatever the space in which we're working is, that gathering information about the points is inherently difficult. We simulate this by allowing only one query to be made to any point every unit time step. Let $T_D = \{t_0, t_0 + 1 \ldots, t_m\}$ be a discretization of T into unit time steps. We define a *query strategy* as a function

$$\mathscr{Q} : T_D \to P \cup \{\emptyset\},$$

which assigns a point to every time $t \in T$ (or no point, in which case $\mathscr{Q}(t) = \emptyset$). In our model, an algorithm is defined by its query strategy. Therefore, we will use the terms algorithm and query strategy interchangeably. We will also often say strategy instead of query strategy. We denote by $\mathfrak{S}(n)$ the space of all query strategies on n points.

Since an algorithm can only query one point every time step, there will be inherent uncertainty in any algorithm's knowledge of the location of the points. If a point i has not been queried in r time steps, then it could be anywhere within a hypersphere of radius $\sigma_i r$ centered at its last known location. We formalize this uncertainty as follows. Let $\mathscr{Q}$ be a query strategy. For every $i \in P$, define the function $\rho_i^{\mathscr{Q}} : T \to \mathcal{X}$ by

$$\rho_i^{\mathscr{Q}}(t) = x_i(t^*) \quad \text{where} \quad t^* = \text{argmax}_{t' \leq t}\{\mathscr{Q}(t') = i\},$$

which gives the *perceived location* of i at time t. Additionally, define

$$u^{\mathscr{Q}} : P \times T \to \mathcal{O}(\mathcal{X}),$$

by

$$u^{\mathscr{Q}}(i, t) = \mathcal{B}_{\sigma_i(t-t*)+1}(\rho_i^{\mathscr{Q}}(t)),$$

where t^* is as above. We will often write $u_i(t)$ in place of $u(i, t)$ for compactness. The function $u(i, t)$ gives the *uncertainty region* of the point i at time t. See Figure 2.1 for a pictorial example. The astute reader may wonder why we have chosen the radius of the uncertainty region to be what it is. The term $\sigma_i(t - t^*)$ is the maximum distance i could have "travelled" in $t - t^*$ time steps—why, then add the plus one? Roughly speaking, there is a question of when the uncertainty region is measured: For a time $t \in T_D$, is $u_i^{\mathscr{Q}}(t)$ measured before or after the query to $\mathscr{Q}(t)$? For somewhat

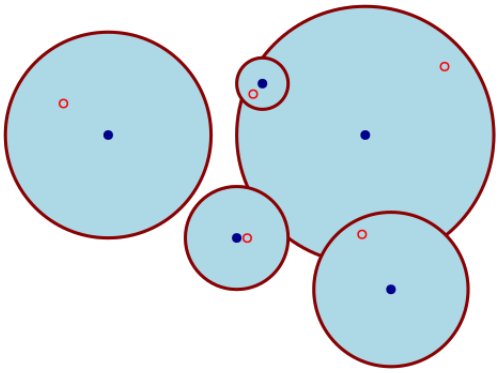


Figure 2.1: A set of points in $\mathbb{R}^2$ and their uncertainty regions at a specific time t. The large, light blue disks with dark boundaries represent the uncertainty regions (note that all have different sizes), centered at the perceived locations (the small, dark blue disks) of the points. The red circles represent the true locations of each point. Here, the points are assumed to all have the same speed.

arbitrary reasons, we assume the measurement occurs (immediately) *before* the query. This has the side effect of ensuring that all uncertainty regions have a radius of at least one. It will be useful to refer to the vector of the locations or perceived locations of the points. As such, we will often write

$$\boldsymbol{\rho}^{\mathscr{Q}}(t) = (\rho_1^{\mathscr{Q}}(t), \ldots, \rho_n^{\mathscr{Q}}(t)), \quad \text{and} \quad \mathbf{x}(t) = (x_1(t), \ldots, x_n(t)).$$

We now explain how to evaluate a query strategy. While this is highly dependent on the specific problem being solved, we define the general structure of several functions enabling us to compare different strategies. Then, given a particular problem, one should choose how the function behaves.

To evaluate the performance of an algorithm at any particular time $t \in T$, we define the *evaluation objective* to be a function acting on the uncertainty regions of the entities and their true locations, and giving some real number $\mathbb{R}$. In this way, the evaluation objective acts a "metric" on a given configuration of points and their uncertainty regions. Formally, the evaluation objective is a function

$$\varphi : \mathcal{O}(\mathcal{X})^n \times \mathcal{X}^n \to \mathbb{R}.$$

Given a query strategy $\mathscr{Q}$, we measure its performance at time t, which we call the *impedance of* $\mathscr{Q}$ *at time* t as

$$\Delta^{\mathscr{Q}}(t) = \varphi(\boldsymbol{\rho}^{\mathscr{Q}}(t), \mathbf{x}(t)).$$

We assume that the goal of strategies is to achieve low impedance (as it suggested by the name). For example, in the ply minimization model [EKLS13, EKLS16, BEK18], the impedance at time t (which they call the ply) is given by the maximum number of overlapping uncertainty intervals, i.e.,

$$\varphi(\boldsymbol{\rho}^{\mathscr{Q}}(t), \mathbf{x}(t)) = \max_{p \in \mathcal{X}} \{\kappa(p) : \kappa(p) = |\{i \in P : p \in u_i^{\mathscr{Q}}(t)|\}.$$

As illustrated by this example, it is not necessary that the evaluation objective be dependent on $\mathbf{x}(t)$.

It remains to define how we evaluate the performance of an algorithm over the entire time interval T, which we call the *overall impedance*. We consider three different evaluation measures. The first, which was originally studied by Evans et al. in [EKLS13], is the *targeted time* measure. In

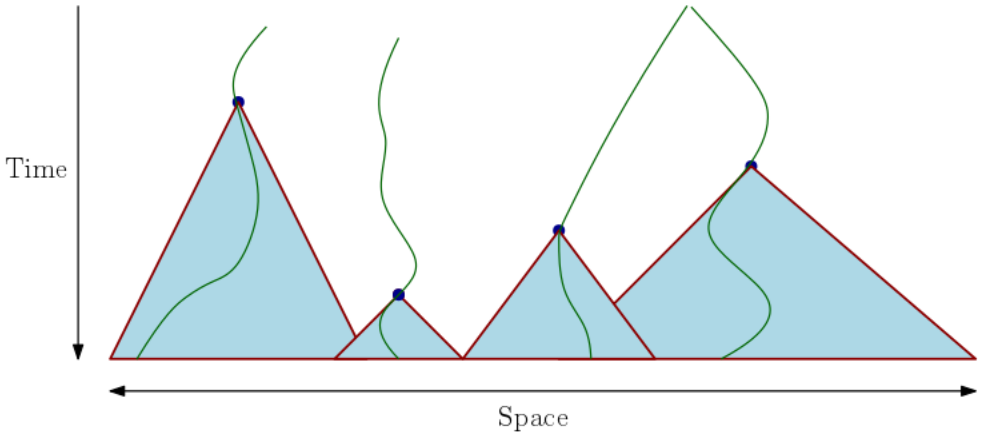


Figure 2.2: Representation taken from [EKLS16]. A set of four points in $\mathbb{R}$ as their uncertainty regions and trajectories change over time. The light blue triangles represent the uncertainty of points, whose perceived locations are at the top of the triangles. The trajectories of the points are seen as light green lines. Here, the points are assumed to have different speeds, as indicated by the different slopes of the triangles.

this case we would like to evaluate φ only at some target time $\tau \in T$. Here, the overall impedance of $\mathscr{Q}$ is given by

$$\Delta^{\mathscr{Q}}_{\mathsf{TT}} = \Delta^{\mathscr{Q}}(\tau).$$

Conversely, we might be interested in a more "continuous time" version of the problem, in which case the goal is to maintain low impedance over the course of T. In this case, we can measure either the maximum impedance achieved by a strategy—which was the case in both [EKLS16, BEK18] in the setting of ply minimization—or the average impedance achieved. These measures are defined as one would expect. For the maximization measure, the overall impedance is given by

$$\Delta^{\mathscr{Q}}_{\mathsf{Max}} = \max_{t \in T} \Delta^{\mathscr{Q}}(t),$$

and for the average measure is given by

$$\Delta^{\mathscr{Q}}_{\mathsf{Avg}} = \frac{1}{t_m - t_0} \int_{t \in T} \Delta^{\mathscr{Q}}(t) \mathrm{d}t.$$

To spare us from the notation, if the algorithm is clear from context, we will often drop the superscript $\mathscr{Q}$ from the notation defined above. Furthermore, when it is clear from context whether we are referring to the impedance at some time or the overall impedance, we will often simply say impedance. Finally, we remark that the time interval T can be recovered from its discretization T_D. Consequently, when specifying a problem in a DRL environment, it is enough to define the associated discretized time interval. We will call a set of times representing discretized time interval a *time horizon*. Note that if T is a time horizon, then $|T|$ is the length of the associated time interval.

Given all of this these definitions, we are finally able to rigorously define the class of problems in which we are interested.

Definition 1. A *Dynamic Resource Limited Problem* (*DRLP*) is defined by four parameters: (1) the "movement space", characterized by a metric space $(\mathcal{X}, d)$ and time horizon T; (2) a set of n continuous functions $x_i : T \to \mathcal{X}$; (3) a evaluation objective $\varphi : \mathcal{O}(\mathcal{X})^n \times \mathcal{X}^n \to V$; and (4) an evaluation measure from $\{\Delta_{\mathsf{TT}}, \Delta_{\mathsf{Max}}, \Delta_{\mathsf{Avg}}\}$.

2.4 Oblivious algorithms

Whatever the specific problem at hand, there is a class of query strategies which are natural strategies to attempt. These strategies, known as "oblivious algorithms", simply pick an ordering on the points and query according to that ordering. They are called oblivious because they do not change their strategy according to the (perceived) locations of the points. The formal definition is as follows.

Definition 2. We say a strategy $\mathscr{Q}$ is oblivious if there exists an interval $I \subseteq T$ and a function $A : I \to P$ such that for all times t, $\mathscr{Q}(t) = A(t \mod |I|)$.

In other words, an oblivious algorithm simply partitions the time steps into n sets, say $\{P_1, \ldots, P_2, \ldots, P_n\}$ and queries point i at all times in P_i. The most straightforward oblivious algorithm is the round-robin strategy (Algorithm 1), which queries point i at times t such that $i \equiv t \mod n$. Henceforth, we will refer to this strategy as RoundRobin. In large part, Chapter 3 will explore the performance of RoundRobin in a specific DRL environment.

Algorithm 1 RoundRobin

Input: A set of points P and a time horizon T

1: Arbitrarily label the points of P $1, 2, \ldots, n$.
2: **for** every time step t in T **do**
3: Query the point $t \mod n$
4: **end for**

Chapter 3

Function Estimation

In this chapter we are concerned with evaluating a function f defined on $\mathcal{X}^n$ over the time horizon T, whose arguments are the locations of the points P, as precisely as possible. Developing bounds on the precision of estimating a function value seems to be a natural question to investigate in this model. As a motivating example, suppose that each point in P represents the electricity usage of a house. While the usage per house is constantly changing and it is most likely infeasible to obtain perfectly accurate information from each house, we may still want to calculate the mean electricity usage. Calculating the mean based on our perceived values of the usage will certainly contain error—however, can we provide any guarantees on the bound between our estimate and the true value?

To pose the problem more formally, we let the metric space $(\mathcal{X}, d)$ be either the complex numbers equipped the the modulus $(\mathbb{C}, |\cdot|)$, or the real numbers equipped with the typical euclidean metric, $(\mathbb{R}, |\cdot|)$. We consider evaluating a function f which acts on the location of the points in $\mathcal{X}$ and produces values in some arbitrary normed vector space $(V, \|\cdot\|_V)$, that is

$$f : \mathcal{X}^n \to V.$$

If the goal is to estimate f as precisely as possible, it seems natural to define the evaluation objective for a query strategy $\mathscr{Q}$ in this case as

$$\varphi(\boldsymbol{\rho}^{\mathscr{Q}}(t), \mathbf{x}(t)) = \left\| f(\boldsymbol{\rho}^{\mathscr{Q}}(t)) - f(\mathbf{x}(t)) \right\|_V, \tag{3.1}$$

that is, the difference between an algorithm's *perceived value* of the function (meaning the value of the function when evaluated with the perceived values), and the true value of the function. As one might imagine, it is quite difficult for any algorithm to keep φ small if it is at all sensitive to its input. (If f is constant, for example, then any strategy will estimate f perfectly at all times). Indeed, we will see that for many functions it is impossible to maintain impedance less than $\Omega(n)$, even for an omniscient algorithm knowing the trajectories of the points.

When comparing algorithms in this chapter, we will be interested in how well an algorithm performs across all inputs. In this sense, we are performing a style of competitive analysis (Section 2.2). Specifically, we will be interested in the worst impedance (according to one of the three evaluation measures) admitted by a strategy across all inputs, say

$$\omega^{\mathscr{Q}} = \sup_P \Delta_M^{\mathscr{Q}},$$

for a strategy $\mathscr{Q}$ where $M \in \{\mathsf{TT}, \mathsf{Max}, \mathsf{Avg}\}$. This will be the analogue to the "competitive ratio" of $\mathscr{Q}$. We will say that a strategy $\mathscr{A}$ is *optimal*, if every strategy $\mathscr{Q}$, $\omega^{\mathscr{Q}} \geq \omega^{\mathscr{A}}$, and we will say that it is $O(1)$-*optimal* if for every $\mathscr{Q}$, $\omega^{\mathscr{Q}} = \Omega(\omega^{\mathscr{A}})$.

The primary goal of this chapter is largely to examine for which functions the RoundRobin strategy is optimal. In Section 3.1 we will give conditions on f which ensure that RoundRobin gives the lowest impedance. In Section 3.2 we will then apply this result to demonstrate that RoundRobin is optimal for linear functions and affine functions. Section 3.3 will then demonstrate that RoundRobin is optimal for order statistics and Section 3.4 then turns its attention to graph theoretic functions, i.e., functions defined on graphs. Here, once again, we will show that RoundRobin is optimal for a specific class of functions.

3.1 Sufficient Condition for Round Robin Optimality

We begin this section by discussing the structural differences in the uncertainty regions of points queried by RoundRobin versus other strategies. We then capitalize on these differences to state a condition on any function f which guarantees that RoundRobin is optimal.

Fix a set of points P and suppose they are being queried according to RoundRobin. At every time $t \in T$ therefore, every point has been queried once in the last n time steps, meaning that there exists a permutation $\pi \in S_n$ such that

$$\operatorname{rad}(u_i(t)) = \pi_i.$$

To compare RoundRobin with other strategies, it will be important to understand the sizes of the uncertainty regions admitted by these other strategies. Let $\mathcal{U}$ denote the set of all vectors $\mathbf{s} \in \mathbb{N}^n$ such that there exists a (valid) query strategy $\mathscr{Q}$ and a time t where s_i represents the radius of $u_i^{\mathscr{Q}}(t)$. More concisely,

$$\mathcal{U} \triangleq \{\mathbf{s} \in \mathbb{N}^n : \exists \mathscr{Q} \in \mathfrak{S}(n), \exists t \in T, \forall i, \operatorname{rad}(u_i^{\mathscr{Q}}(t)) = s_i\}.$$

Let $\mathbf{s} \in \mathcal{U}$. Consider the last n time steps in the query strategy which gives rise to $\mathbf{s}$. Let $E \subseteq P$ be the set of points queried in the last n time steps. Since only one point can be queried every time step, the uncertainty region of every point has a different radius. By the pigeonhole principle therefore, there exists a permutation $\{i_1, \ldots, i_{|E|}\}$ on the points of E such that

$$\operatorname{rad}(u_{i_1}^{\mathscr{Q}}(t)) \geq 1, \quad \operatorname{rad}(u_{i_2}^{\mathscr{Q}}(t)) \geq 2, \quad \ldots, \quad \operatorname{rad}(u_{i_{|E|}}(t)) \geq |E|.$$

By similar reasoning, since none of the points in $P \setminus E$ were queried in the last n time steps, there exists a permutation $\{i_{|E|+1}, \ldots, i_n\}$ on $P \setminus E$ such that

$$\operatorname{rad}(u_{i_{|E|+1}}(t)) \geq n+1, \quad \ldots, \quad \operatorname{rad}(u_{i_n}(t)) \geq n + |P \setminus E|.$$

If RoundRobin is to be optimal for a function f, it should be the case that querying some points more often than others is "suboptimal", loosely speaking. Querying some points more frequently than others results, as seen above, in uncertainty regions with radii of size $\Omega(n)$, thus the sizes of the uncertainty regions are more "spread out". This notion is captured by Observation 3.1. Before it is stated however, let us define some terminology. Given a permutation $\pi \in S_n$, we will let $\boldsymbol{\pi}$ be the associated vector in $\mathbb{R}^n$, that is, $(\pi_1, \pi_2, \ldots, \pi_n)$. Let $\epsilon \in S_n$ denote the identity permutation $(1, 2, \ldots, n) \mapsto (1, 2, \ldots, n)$, (thus $\boldsymbol{\epsilon}$ will be the vector $(1, 2, \ldots, n)$). Furthermore, for a vector

$\mathbf{a} \in \mathbb{R}^n$, let $\mathbf{a}^\uparrow$ denote that vector with the same coordinates as $\mathbf{a}$, but with coordinates sorted in increasing order. For example, if $\mathbf{a} = (-3, 1, 15, 4, 0)$, then $\mathbf{a}^\uparrow = (-3, 0, 1, 4, 15)$. Note that $\boldsymbol{\epsilon} = \boldsymbol{\pi}^\uparrow$. In an effort to present the following results in full generality, we will state the them using the general *p-norm* on $\mathbb{R}^n$ or $\mathbb{C}^n$. Given a vector $\mathbf{v}$, recall that the p-norm for $1 \leq p \leq \infty$ is given by

$$\|\mathbf{v}\|_p \triangleq \left(\sum_{i=1}^{n} |v_i|^p \right)^{1/p},$$

for $p < \infty$, and

$$\|\mathbf{v}\|_\infty \triangleq \max_i |v_i|,$$

for $p = \infty$. Note that $|v_i|$ here refers to the absolute value (modulus) if v_i in $\mathbb{R}$ or $\mathbb{C}$.

Observation 3.1. *For all $\mathbf{s} \in \mathcal{U}$, $\mathbf{s}^\uparrow \geq \boldsymbol{\epsilon}$ and for any $1 \leq p \leq \infty$, $\|\mathbf{s}\|_p \geq \|\boldsymbol{\epsilon}\|_p$.*

Proof. From the discussion above there exists a permutation on the coordinates of $\mathbf{s}$, say $\{i_1, \ldots, i_n\}$, and a set E such that $s_{i_j} \geq j$ for $j \leq |E|$ and $s_{i_j} \geq n + j$ for $j > |E|$. Therefore, we obtain immediately that $s_i^\uparrow \geq i$ for all $i \in [|E|]$ and $s_i^\uparrow \geq n + i$ for $i > |E|$, demonstrating the first part of the claim. The second part now follows easily as well: we have $|s_i^\uparrow| \geq i$, hence

$$\|\mathbf{s}\|_p = \|\mathbf{s}^\uparrow\|_p = \left(\sum_{i=1}^{n} |s_i^\uparrow|^p \right)^{1/p} \geq \left(\sum_{i=1}^{n} |i|^p \right)^{1/p} = \|\boldsymbol{\epsilon}\|_p. \qquad \blacksquare$$

It may be worth remarking now to ensure clarity later on that for any $\pi \in S_n$, $\|\boldsymbol{\pi}\|_p = \|\boldsymbol{\epsilon}\|_p$ for any p. We are now ready to state a condition on f to ensure that RoundRobin is optimal.

Proposition 3.1. *For a function $f : \mathcal{X}^n \to V$ where $(V, \|\cdot\|_V)$ is a normed vector space and $\mathcal{X} \in \{\mathbb{C}, \mathbb{R}\}$,* RoundRobin *is an optimal strategy on all three evaluation measures if for all $\mathbf{x} \in \mathbb{R}^n$,*

$$\|f(\mathbf{x} - \mathbf{u}) - f(\mathbf{x})\|_V \leq \|f(\mathbf{x} - \mathbf{v}) - f(\mathbf{x})\|_V, \tag{3.2}$$

and $O(1)$-competitive if

$$\|f(\mathbf{x} - \mathbf{u}) - f(\mathbf{x})\|_V = O(\|f(\mathbf{x} - \mathbf{v}) - f(\mathbf{x})\|_V), \tag{3.3}$$

for all $\mathbf{u}, \mathbf{v} \in \mathcal{X}^n$ such that $\|\mathbf{u}\|_p \leq \|\mathbf{v}\|_p$.

Proof. Suppose f satisfies condition (3.6). We will show that RoundRobin is optimally competitive. Let $\mathscr{R}$ be the query strategy of RoundRobin. First we bound the evaluation objective of RoundRobin on any point set P. Consider any $t \in T$. For all i, let $z_i \in \mathbb{R}$ be the the offset between i's perceived location and true location, i.e., $x_i(t) = \rho_i(t) - z_i$. By definition of RoundRobin there exists a permutation π such that $|\rho_i(t) - x_i(t)| \leq \pi_i$. Note that $|z_i| = |\rho_i(t) - x_i(t)|$, thus $|z_i| \leq \pi_i$. Letting $\mathbf{z} = (z_1, \ldots, z_n)$, we have that

$$\sum_{i=1}^{n} |z_i|^p \leq \sum_{i=1}^{n} |\pi_i|^p,$$

hence

$$\|\mathbf{z}\|_p \leq \left(\sum_{i=1}^{n} |z_i|^p \right)^{1/p} \leq \left(\sum_{i=1}^{n} |\pi_i|^p \right)^{1/p} = \|\boldsymbol{\pi}\|_p.$$

By assumption therefore,

$$\varphi(\boldsymbol{\rho}^{\mathscr{R}}(t), \mathbf{x}(t)) = \|f(\mathbf{x}(t) - \mathbf{z}) - f(\mathbf{x}(t))\|_V \leq \|f(\mathbf{x}(t) - \boldsymbol{\pi}) - f(\mathbf{x}(t))\|_V. \tag{3.4}$$

Conversely, consider a set of n points such that $x_i(t) = t$ for all $t \in T$. Here, $|\rho_i(t) - x_i(t)| = \text{rad}(u_i(t))$ for all i and all t. Fix a particular t and let $\mathbf{s} \in \mathcal{U}$ describe the radii of the uncertainty regions of the entities according to the query strategy $\mathscr{Q}$. Then, $\rho_i(t) = x_i(t) - s_i$ and by Observation 3.1, $\|\mathbf{s}\|_p \geq \|\boldsymbol{\epsilon}\|_p = \|\boldsymbol{\pi}\|_p$. Applying (3.6),

$$\varphi(\boldsymbol{\rho}^{\mathscr{Q}}(t), \mathbf{x}(t)) = \|f(\mathbf{x}(t) - \mathbf{s}) - f(\mathbf{x}(t))\|_V \geq \|f(\mathbf{x}(t) - \boldsymbol{\pi}) - f(\mathbf{x}(t))\|_V. \tag{3.5}$$

Hence, $\varphi(\boldsymbol{\rho}^{\mathscr{R}}(t), \mathbf{x}(t)) \leq \|f(\mathbf{x}(t) - \boldsymbol{\pi}) - f(\mathbf{x}(t))\|_V \leq \varphi(\boldsymbol{\rho}^{\mathscr{Q}}(t), \mathbf{x}(t))$. Notice that this analysis held regardless of which t was chosen. We conclude that RoundRobin is optimal on all three evaluation measures. If f satisfies condition 3.7 instead, the proof is identical, except that Equation (3.4) is replaced by $\|f(\mathbf{x}(t) - \mathbf{z}) - f(\mathbf{x}(t))\|_V = O(\|f(\mathbf{x}(t) - \boldsymbol{\pi}) - f(\mathbf{x}(t))\|_V)$ and similarly with Equation (3.5). This completes the proof. ■

While the conditions of 3.1 are easy to state, if $\mathcal{X} = \mathbb{R}$ we are able to remove some of the assumptions of $\mathbf{u}$ and $\mathbf{v}$. Admittedly, however, the result becomes less digestible. We state it now.

Proposition 3.2. *For a function $f : \mathbb{R}^n \to V$ where $(V, \|\cdot\|_V)$ is a normed vector space,* RoundRobin *is an optimally competitive strategy on all three evaluation measures if for all $\mathbf{x} \in \mathbb{R}^n$ and $\pi \in S_n$,*

$$\|f(\mathbf{x} - \mathbf{u}) - f(\mathbf{x})\|_V \leq \|f(\mathbf{x} - \boldsymbol{\pi}) - f(\mathbf{x})\|_V \leq \|f(\mathbf{x} - \mathbf{v}) - f(\mathbf{x})\|_V, \tag{3.6}$$

and similarly $O(1)$-optimal if

$$\|f(\mathbf{x} - \mathbf{u}) - f(\mathbf{x})\|_V = O(\|f(\mathbf{x} - \boldsymbol{\pi}) - f(\mathbf{x})\|_V) = O(\|f(\mathbf{x} - \mathbf{v}) - f(\mathbf{x})\|_V), \tag{3.7}$$

for all $\mathbf{u}, \mathbf{v}$ such that $\mathbf{u} \leq \boldsymbol{\pi}$ and $\boldsymbol{\epsilon} \leq \mathbf{v}^{\uparrow}$.

Proof. The proof follows precisely the same format as above, however, we are able to take advantage of the fact that $\mathbb{R}$ is well-ordered. If z_i and π are as in the previous proof, then $z_i \leq |z_i| \leq \pi_i$ and so $\mathbf{z} \leq \boldsymbol{\pi}$. Therefore $\varphi(\boldsymbol{\rho}^{\mathscr{R}}(t), \mathbf{x}(t)) = \|f(\mathbf{x}(t) - \mathbf{z}) - f(\mathbf{x}(t))\|_V \leq \|f(\mathbf{x}(t) - \boldsymbol{\pi}) - f(\mathbf{x}(t))\|_V$. Conversely, if $\mathbf{s} \in \mathcal{U}$ is as in the previous proof, then $\mathbf{s}^{\uparrow} \geq \boldsymbol{\epsilon}$ by Observation 3.1, hence $\varphi(\boldsymbol{\rho}^{\mathscr{Q}}(t), \mathbf{x}(t)) = \|f(\mathbf{x}(t) - \mathbf{s}) - f(\mathbf{x}(t))\|_V \geq \|f(\mathbf{x}(t) - \boldsymbol{\pi}) - f(\mathbf{x}(t))\|_V$. The rest of the argument is identical to above. ■

We now use Proposition 3.2 in the upcoming sections to demonstrate the several classes of functions for which RoundRobin is optimally competitive.

3.2 Affine and Linear Functions

This section is an exercise in demonstrating the certain functions obey the hypotheses of Propositions above. We begin with affine functions. Let T_n denote $\sum_{i=1}^{n} i = n(n+1)/2$.

Corollary 3.2.1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be an affine function $\mathbf{x} \mapsto \mathbf{c}^T\mathbf{x} + b$ for some $\mathbf{c} \in (\mathbb{R}_{>0})^n$. Then* RoundRobin *is $O(1)$ optimal for f.*

Proof. We will demonstrate that f satisfies condition (3.7). First notice that for any $\pi \in S_n$,

$$|f(\mathbf{x}-\boldsymbol{\pi}) - f(\mathbf{x})| = \left|\sum_{i=1}^{n} c_i \pi_i\right| = \Theta(T_n). \tag{3.8}$$

Let $\pi \in S_n$ and let $\mathbf{u} \in \mathbb{R}^n$ obey $\mathbf{u} \leq \boldsymbol{\pi}$. Then,

$$|f(\mathbf{x}-\mathbf{u}) - f(\mathbf{x})| = \left|\sum_{i=1}^{n} c_i u_i\right| \leq \sum_{i=1}^{n} |c_i u_i| \leq \max_j |c_j| \sum_{i=1}^{n} |u_i| = O(\|\mathbf{u}\|_1) = O(T_n). \tag{3.9}$$

Conversely, suppose $\mathbf{v} \in \mathbb{R}^n$ obeys $\mathbf{v}^{\uparrow} \geq \boldsymbol{\epsilon}$. Then,

$$|f(\mathbf{x}-\mathbf{v}) - f(\mathbf{x})| = \left|\sum_{i=1}^{n} c_i v_i\right| = \sum_{i=1}^{n} c_i v_i = \Omega(\|\mathbf{v}\|_1) = \Omega(T_n), \tag{3.10}$$

where to remove the absolute value we've used the fact that $c_i \in \mathbb{R}_{>0}$ and $v_i \geq \epsilon_i \geq 1$, thus $c_i v_i > 0$. Combining lines (3.8), (3.9), and (3.10) we obtain that $|f(\mathbf{x}-\mathbf{u}) - f(\mathbf{x})| = O(|f(\mathbf{x}-\boldsymbol{\pi}) - f(\mathbf{x})|) = O(|f(\mathbf{x}-\mathbf{v}) - f(\mathbf{x})|)$, as was claimed. ■

As remarked in the proof above, the non-negativity of c_i was crucial to obtain inequality (3.10). Indeed, if this was not known, then it's not clear whether one could say that $|\sum_i c_i v_i| = \Omega(T_n)$. Meanwhile, obtaining inequality (3.9) does not require non-negativity of $\mathbf{c}$, as is even demonstrated by the proof. It does not use this fact. This result also demonstrated the necessity of the hypothesis of Proposition 3.2 as oppposed to Proposition 3.1. It is not true in general that $|\sum_i c_i v_i| = O(|\sum_i c_i u_i|)$ if $\mathbf{u} \leq \mathbf{v}$, hence f does not meet (3.3).

We now remark that we obtain $O(1)$ optimality of linear functions for free.

Corollary 3.2.2. *Let $f : \mathbb{R}^n \to \mathbb{R} : \mathbf{x} \mapsto \mathbf{c}^T\mathbf{x}$ for $\mathbf{c} \in (\mathbb{R}_{>0})^n$ be a linear function. Then* RoundRobin *is $O(1)$ optimal.*

Proof. Linear functions are affine functions with no offset. Apply Corollary 3.2.1. ■

It is worth pointing out as an important special case that we have obtained that RoundRobin is optimal for the mean, i.e., for the function $\mu(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i$, since it is a linear function.

3.3 Order Statistics

Recall that the *i-th order statistic* of $\mathbf{x}$, $H^i(\mathbf{x})$ is defined as the i-th smallest element of $\mathbf{x}$. We will show that RoundRobin is $O(1)$-optimal for estimating H^i. Interestingly, in general, H^i does not obey the hypothesis of Proposition 3.2. Thus, the results of this section demonstrate that the conditions of the proposition cannot be necessary conditions to ensure optimality of RoundRobin. The following example demonstrates that H^i does not obey condition (3.7), by showing it for $H^1 = \min$.

Example 3.1. Let $\mathbf{x} = \mathbf{v} = (1, 2, \ldots, n)$, and consider the permutation $\boldsymbol{\pi} = (n, n-1, \ldots, 1)$. Then $\min(\mathbf{x}) = 1$, $\min(\mathbf{x}-\boldsymbol{\pi}) = 1-n$ and $\min(\mathbf{x}-\mathbf{v}) = 0$. Therefore,

$$|\min(\mathbf{x}-\boldsymbol{\pi}) - \min(\mathbf{x})| = n = \omega(1) = |\min(\mathbf{x}-\mathbf{v}) - \min(\mathbf{x})|,$$

and since $\mathbf{v}^{\uparrow} \geq \boldsymbol{\epsilon}$, we see that (3.7) does not hold.

We now demonstrate that any strategy can perform poorly when estimating H^i. We remark that for the special cases H^1 and H^n (min and max, respectively), this fact was pointed by Simon Suyadi in his M.Sc. thesis [Suy12].

Lemma 3.1. *Any strategy maintains impedance $\Omega(n)$ on any evaluation measure on the i-th order statistic, H^i.*

Proof. First, suppose that $i \leq n/2$. Define n points with trajectories $x_i(t) = t$. Let $\mathscr{Q}$ be any query strategy. At any time t, let $\mathbf{s} \in \mathcal{U}$ describe the uncertainty regions of the points. Suppose without loss of generality that $\mathbf{s}$ is in decreasing order, so $s_1 > s_2 > \cdots > s_n$ (note that they must be strict inequalities by definition of $\mathcal{U}$). Then $t - s_1 < t - s_2 < \cdots < t - s_n$, hence $H^i(\mathbf{x} - \mathbf{s}) = t - s_i$. Since $\mathbf{s}^{\uparrow} \geq \boldsymbol{\epsilon}$ by Observation 3.1, we have that $s_i \geq n - i + 1$. Therefore,

$$\Delta^{\mathscr{Q}}(t) = |H^i(\mathbf{x}(t) - \mathbf{s}) - H^i(\mathbf{x}(t))| = t - (t - s_i) = s_i \geq n - i + 1 \geq n/2 + 1 = \Omega(n),$$

for all times t. If $i > n$, then we define the trajectories as $x_i(t) = -t$. The proof that the impedance at all times is $\Omega(n)$ follows a similar structure. ■

It remains to show that RoundRobin maintains impedance $O(n)$ at all times.

Lemma 3.2. *For the i-order statistic, H^i,* RoundRobin *maintains impedance $O(n)$ at all times.*

Proof. Suppose RoundRobin is querying according to the permutation π. Fix any time t. Relabel the points such that $x_1(t) \leq x_2(t) \leq \cdots \leq x_n(t)$. Hence $H^i(\mathbf{x}(t)) = x_i(t)$. For every point k, let $z_k = x_k(t) - \rho_k(t)$. Suppose that $H^i(\boldsymbol{\rho}(t)) = \rho_j(t) = x_j(t) - z_j$, $|z_j| \leq \pi_j$. First we claim that $x_j(t) - z_j \geq x_i(t) - n$. Note that for every point $k \geq i$, we have $x_k(t) - z_k \geq x_i(t) - n$ since $x_k(t) \geq x_i(t)$ and $|z_k| \leq n$. There are $n - i + 1$ of these points, therefore, at most $i - 1$ points can have perceived locations less than $x_i(t) - n$. It follows by the definition of H^i that $x_j(t) - z_j \geq x_i(t) - n$. Next, we claim that $x_j(t) - z_j \leq x_i(t) + n$. This is easier to see: for every $k = 1, \ldots, i$, since $x_k(t) \leq x_i(t)$, we have $x_k(t) - z_k \leq x_i(t) + n$. Therefore, at least i points have perceived locations at most $x_i(t) + n$, again implying that $x_j(t) - z_j \leq x_i(t) + n$. We have thus obtained that

$$H^i(\boldsymbol{\rho}(t)) = [H^i(\mathbf{x}(t)) - n, H^i(\mathbf{x}(t)) + n],$$

which in turn implies that

$$\Delta(t) = |H^i(\boldsymbol{\rho}(t)) - H^i(\mathbf{x}(t))| = O(n).$$

Since t was arbitrary, we have obtained the desired result. ■

3.4 Graph Theoretic Functions

In this section we turn our attention to functions defined on graphs. Let $\mathscr{G}$ be the set of all graphs. A *graph theoretic function* is a function

$$f : \mathscr{G} \to \mathbb{R},$$

which measures some property of a graph. For example, the chromatic number ($\chi(G)$), the number of triangles, or the clique number of a graph are all examples of graph theoretic functions. We

will apply the our dynamic resource limited model to graph theoretic functions as follows. We will denote a weighted graph as a triple, $G = (V(G), E(G), \mathbf{x})$, where x_i assigns a weight to edge $e_i \in E$. Typically we will write V and E instead of $V(G)$ and $E(G)$ when the graph G is clear from context. We will be interested in estimating a graph theoretic function f on a graph whose edges are changing over time, i.e., when $x_i = x_i(t)$ is a function of time. We will call such graphs *dynamic graphs* and write $G(t)$ to be the graph at time t: $G(t) = (V, E, x_i(t))$. For example, if G is a (connected) dynamic graph and $u, v \in E(G)$, then at every time t there exists a shortest path between u and v. If $f(G(t))$ is the length of this path, we can ask how well we can estimate f over time.

What follows is an examination of a specific class of graph theoretic functions, for which RoundRobin is optimally competitive.

Definition 3. We say $f : \mathscr{G} \to \mathbb{R}$ is an *optimizing edge function* if

$$f(G = (V, E, \mathbf{x})) = \max_{A \subseteq \mathscr{F}(G)} \sum_{e_i \in A} x_i, \quad \text{or} \quad f(G = (V, E, \mathbf{x})) = \min_{A \subseteq \mathscr{F}(G)} \sum_{e_i \in A} x_i,$$

where $\mathscr{F}(G) \subseteq 2^E$ is a feasible set of edges of the given graph G. In the former case, we say f is a *maximizing edge function*, and a *minimizing edge function* in the latter case.

Let us unpack this definition slightly. As an example, we can express the shortest path and the minimum spanning tree problems as computing optimizing edge functions. Suppose we are interested in the shortest path between u and v in G. Here, $\mathscr{F}(G)$ is the set of all paths from u and v and the shortest path is a minimizing edge function. (Similarly, the maximum path would be a maximizing edge function). For the minimum spanning tree problem, $\mathscr{F}(G)$ would be the set of all spanning trees of G and the length of the minimum tree would be the value of minimizing edge function over $\mathscr{F}(G)$.

By way of contrast, the value of the maximum flow in an s-t graph cannot be cast as optimizing edge function. This is because in general, the amount of flow being pushed along an edge is not equal to the weight (capacity) of that edge. To remedy this, we might consider generalizing the definition of maximizing edge function to be

$$\max_{A \subseteq \mathscr{F}(G)} \sum_{e_i \in A} f^A(x_i),$$

where f^A assigns a value to an edge e_i based on the other elements in A. In this case, we could let $f^A(x_i)$ be the flow along e_i, where A is the set of feasible edges along which flow is being pushed. However, generalizing Definition 3 in this way would cause problems in the following proposition. These problems are illustrated after the proof.

Proposition 3.3. *Let f be an optimizing edge function. Given any graph $G = (V, E, \mathbf{x})$,* RoundRobin *achieves impedance $O(|E|^2)$ at all times t.*

Proof. First suppose that f is a minimizing edge function. We will later show how to modify the following proof in the other case. Let G be given and let $\mathscr{F}(G)$ be the set of feasible edges over which f is being computed. Fix a time t and let

$$A \equiv \{e_1, e_2, \ldots, e_k\} = \text{argmin}_{A \subseteq \mathscr{F}(G)} \sum_{e \in A} x_i(t),$$

give the optimal solution to f at time t. Similarly, let $B = \{e'_1, e'_2, \ldots, e'_j\}$ be *perceived* optimal solution to f at time t, i.e.,

$$B = \operatorname{argmin}_{A \subseteq \mathscr{F}(G)} \sum_{e \in B} \rho_i(t).$$

Therefore,

$$\sum_{e_i \in A} x_i(t) \leq \sum_{e'_i \in B} x_i(t), \quad \text{and} \quad \sum_{e_i \in A} \rho_i(t) \geq \sum_{e'_i \in B} \rho_i(t). \tag{3.11}$$

Set $S_{\text{RR}} = \sum_{e'_i \in B} \rho_i(t)$ and $S_{\text{OPT}} = \sum_{e_i \in A} x_i(t)$ and consider the quantity $|S_{\text{RR}} - S_{\text{OPT}}|$. By definition of RoundRobin, there exists a permutation π such that $|\rho_j(t) - x_j(t)| \leq \pi_j$. Therefore, for all i, $\rho_i(t) - x_i(t) \leq |\rho_i(t) - x_i(t)| \leq \pi_i$, so $x_i(t) \geq \rho_i(t) - \pi_i$ and $\rho_i(t) \leq x_i(t) + \pi_i$. If $S_{\text{RR}} \geq S_{\text{OPT}}$, then

$$|S_{\text{RR}} - S_{\text{OPT}}| = S_{\text{RR}} - S_{\text{OPT}} \leq \sum_{e'_i \in B} \rho_i(t) - \sum_{e_i \in A} (\rho_i(t) - \pi_i) \leq \sum_{e_i \in A} \pi_i = O(|E|^2), \tag{3.12}$$

where we've employed (3.11) to obtain the final inequality. Otherwise, if $S_{\text{OPT}} \geq S_{\text{RR}}$, then

$$|S_{\text{RR}} - S_{\text{OPT}}| = S_{\text{OPT}} - S_{\text{RR}} \leq \sum_{e'_i \in B} (x_i(t) + \pi_i) - \sum_{e_i \in A} x_i(t) \leq \sum_{e_i \in A} \pi_i = O(|E|^2), \tag{3.13}$$

where again we've used (3.11). We conclude that

$$\varphi(\boldsymbol{\rho}^{\mathscr{R}}(t), \mathbf{x}(t)) = |S_{\text{RR}} - S_{\text{OPT}}| = O(|E|^2). \qquad \blacksquare$$

If we were to generalize Definition 3 as suggested above, then the arithmetic on lines (3.12) and (3.13) would not hold. While the equivalent versions of inequalities (3.11) would hold (e.g., $\sum_{e_i \in A} f^A(x_i(t)) \leq \sum_{e_i \in B} f^A(x_i(t))$), the relation between $f^A(\rho_i(t))$ and $f^A(\rho_i(t) - \pi_i)$ is unclear. Thus, bounding $\sum_{e'_i \in B} f^A(\rho_i(t)) - \sum_{e_i \in A} f^A(\rho_i(t) - \pi_i)$ becomes a more daunting (and potentially impossible) task.

We now demonstrate that for several specific problems which can be cast as computing minimizing edge functions, that RoundRobin provides an $O(1)$-optimal bound on the impedance.

Lemma 3.3. *Suppose $f(G)$ is the length of the shortest path in G, or the length of the minimum spanning tree. There exists a dynamic graph such that any query strategy admits impedance $\Omega(|E|^2)$ at all times when estimating f.*

Proof. Let G be a simple path on n vertices between s and t. On this graph, notice that the length of the minimum spanning tree is the same as the minimum length s-t path. For each edge e_i define $x_i(t) = t$. Let $\mathscr{Q}$ be any querying strategy. At time t, let $\mathbf{s} \in \mathcal{U}$ describe the radii of the uncertainty regions of the edges. Thus $\rho_i(t) = t - s_i$ and

$$\varphi(\boldsymbol{\rho}^{\mathscr{Q}}(t), \mathbf{x}(t)) = \left| \sum_{i=1}^{|E|} t - s_i - \sum_{i=1}^{|E|} t \right| = \left| \sum_{i=1}^{|E|} s_i \right| = \sum_{i=1}^{|E|} s_i \geq \sum_{i=1}^{|E|} \pi_i = \Omega(|E|^2),$$

completing the proof. $\blacksquare$

Interestingly, while we have now demonstrated that RoundRobin is optimally competitive for the shortest path and MST problems, it is not clear that the corresponding minimizing edge functions satisfy the conditions of Proposition 3.1 or 3.2. If not, then this would provide further demonstration (in addition to Section 3.3) that the conditions of these propositions are not necessary to guarantee the optimality of RoundRobin.

While it is not clear how RoundRobin performs on the maximum flow problem, we can provide a lower bound on any query strategy. This requires a slightly different construction than in Lemma 3.3.

Lemma 3.4. *For an s-t graph G suppose that $f(G)$ computes the maximum flow from s to t. Then there exists a dynamic graph such that any query strategy admits an impedance $\Omega(|E|^2)$ at all times when estimating f.*

Proof. Construct an s-t graph as follows. Let each edge be a direct edge from s to t. Then let $x_i(t) = t$, as above. Clearly, the maximum flow at time t is $t|E|$. Let $\mathscr{Q}$ be an query strategy, and let $\mathbf{s} \in \mathcal{U}$ describe the radii of the uncertainty regions of the edges so that, again as above, $\rho_i(t) = x_i(t) - s_i$. Then,

$$\varphi(\boldsymbol{\rho}^{\mathscr{Q}}(t), \mathbf{x}(t)) = \left| \sum_{i=1}^{|E|} (t - s_i) - t|E| \right| = \sum_{i=1}^{|E|} s_i = \Omega(|E|^2). \qquad \blacksquare$$

Chapter 4

The Threshold Problem

In this chapter we shift our attention away from function estimation and begin investigating our model from a more geometric perspective. Specifically, we will focus on the uncertainty regions of the points and try to limit their size. Instead of concerning ourselves about how closely the perceived value of a point lies to its true location, we now assume that there is a limit which specifies how big the uncertainty region of each point is allowed to grow. Of course, this limit is specific to each point otherwise the problem becomes trivial. Indeed, according to a discussion in Section 3.1, every strategy at every time will admit a point which has an uncertainty region of size at least n. Therefore, if a uniform bound of less than n was given, no strategy could achieve this bound. Otherwise, if a uniform bound of greater than n was given, RoundRobin would suit our needs.

Our setting is an arbitrary metric space $(\mathcal{X}, d)$. We will impose a limit on the size of i's uncertainty region by fixing a point $o \in \mathcal{X}$, which we will call the *origin*, and attempting to query i before its uncertainty region overlaps the origin. The goal is to develop a query strategy $\mathscr{Q}$ such that

$$\operatorname{rad}(u_i^{\mathscr{Q}}(t)) \leq d(\rho_i^{\mathscr{Q}}(t), o), \tag{4.1}$$

for all $i \in P$ and times t. We call this the *threshold problem*. At a specific time t, this problem is equivalent to the pinwheel scheduling problem introduced in Section 1.2.3. At any given time, given the current perceived locations of the entities, we would like a query strategy which queries each entity i within the next $d(\rho_i^{\mathscr{Q}}(t), o)$. Therefore, if the locations of the entities are fixed over time, meaning that $\rho_i^{\mathscr{Q}}(t_1) = \rho_i^{\mathscr{Q}}(t_2)$ for all $t_1, t_2 \in T$, then this is precisely equivalent to the pinwheel scheduling problem. Consequently, the threshold problem seems a perfect analogy to a "dynamic pinwheel scheduling problem".

As mentioned in the introduced, due to the dynamic nature of DRL environments, it is often impossible to maintain a perfect solution to many problems. Therefore, we relax condition (4.1) and search for a strategy $\mathscr{Q}$ such that $\operatorname{rad}(u_i^{\mathscr{Q}}(t)) = O(d(\rho_i^{\mathscr{Q}}(t), o))$ at all times t, or provides a "proof" that for any algorithm, there is some point in time at which the algorithm would have performed "poorly". The result is formalized below.

Proposition 4.1. *There exists an algorithm $\mathscr{A}$ such that either $\operatorname{rad}(u_i^{\mathscr{A}}(t)) = O(d(\rho_i^{\mathscr{A}}(t), o))$ for all times $t \in T$ and $i \in P$, or else fails and certifies that for any strategy $\mathscr{Q}$, there is a time t and $i \in P$ such that $\operatorname{rad}(u_i^{\mathscr{Q}}(t)) = \Omega(d(\rho_i^{\mathscr{Q}}(t), o))$.*

To formally relate the threshold problem to the model as defined in Section 2.3, here we take

the impedance to be the largest factor by which a point's uncertainty region exceeds its desired size, that is,

$$\varphi(\boldsymbol{\rho}^{\mathscr{Q}}(t), \mathbf{x}(t)) = \max_i \left\{ \left\lfloor \frac{\mathrm{rad}(u_i^{\mathscr{Q}}(t))}{d(\rho_i^{\mathscr{Q}}(t), o)} \right\rfloor \right\}.$$

Proposition 4.1 is then translated as providing an algorithm $\mathscr{A}$ such that $\Delta^{\mathscr{A}}_{\mathsf{Max}}(t) = O(1)$ or certifies that any strategy $\mathscr{Q}$ has $\Delta^{\mathscr{Q}}_{\mathsf{Max}}(t) = \Omega(1)$.

In order to simplify the notation of this section, we will often abuse notation and simply write $\rho_i^{\mathscr{Q}}(t)$ in place of $d(\rho_i^{\mathscr{Q}}(t), o)$. The reader is encouraged to think of $\rho_i^{\mathscr{Q}}(t)$ as values on the real line, and as the origin o as the number $0 \in \mathbb{R}$. In this case, $|\rho_i^{\mathscr{Q}}(t)| = d(\rho_i^{\mathscr{Q}}(t), 0)$, so the abuse of notation is not as pernicious as one imagines. In the spirit of relating our problem to the pinwheel scheduling problem, we will sometimes refer to $\rho_i^{\mathscr{Q}}(t)$ as the *demand* of i at time t. The rational behind this term will become clear in Section 4.1.

Several tools and observations from the original pinwheel scheduling problem—henceforth referred to as static pinwheel scheduling—will prove useful. The next section briefly reviews some of the results.

4.1 Static Pinwheel Scheduling

Recall from Section 1.2.3 that given vector $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{N}^n$ a *pinwheel schedule* for $\mathbf{v}$ is an infinite sequence $s \in \mathbf{v}^{\mathbb{N}}$ such that for every $i \in [n]$ and $j \in \mathbb{N}$, $v_i \in s(\{j, j+1, \ldots, j+v_i\})$. If there exists a pinwheel schedule for $\mathbf{v}$ we say that $\mathbf{v}$ is *schedulable* or that it *admits a pinwheel schedule*. Otherwise, we say $\mathbf{v}$ is *unschedulable*. A sequence (or schedule) in $\mathbf{v}^{\mathbb{N}}$ is *valid* if it proves that $\mathbf{v}$ is schedulable, and *invalid* otherwise. We call v_i the demand of v. A quantity which is intimately related to the schedulability of $\mathbf{v}$ is the inverse sum

$$\lambda(\mathbf{v}) \triangleq \sum_{i=1}^{n} \frac{1}{v_i},$$

which is called the *density* of $\mathbf{v}$. Intuitively, the contribution of $1/v_i$ to the density of $\mathbf{v}$ dictates the minimum fraction of attention which any schedule must give to v_i in order to be a valid schedule. More precisely, an a pinwheel schedule for $\mathbf{v}$, v_i must be scheduled at least a $1/v_i$-fraction of the time. This yields the following simple observation.

Observation 4.1 ([HMR+89])**.** *The vector* $\mathbf{v} = (v_1, \ldots, v_n)$ *admits a pinwheel schedule only if* $\lambda(\mathbf{v}) \leq 1$.

Somewhat surprisingly however, the converse of Observation 4.1 is not true. Consider the vector $\mathbf{v} = (2, 3, k)$ for any $k \geq 6$ which has $\lambda(\mathbf{v}) \leq 1$. Given any schedule, consider where k appears in the sequence. In order for 2 to meet its demand, it must be scheduled both immediately before and immediately after k. However, 3 is then not scheduled in three time steps, meaning that it is not queried according to its demand.

Demonstrating that the demand of a vector is sufficiently large is therefore a way to demonstrate that there is no pinwheel schedule for a vector. In our setting, we apply this as follows.

Observation 4.2. *For any query strategy* $\mathscr{Q}$*, if there exists a time* t *such that* $\lambda(\boldsymbol{\rho}^{\mathscr{Q}}(t)) = \Omega(1)$*, then there exists some point* i *such that* $\mathrm{rad}(u_i(t)) = \Omega(\rho_i^{\mathscr{Q}}(t))$.

Proof. If $\text{rad}(u_i(t)) = O(\rho_i^{\mathscr{Q}}(t))$ for all i, then i has been queried in the last $c_i\rho_i^{\mathscr{Q}}(t))$ time steps for some constant c_i, hence

$$\lambda(\boldsymbol{\rho}^{\mathscr{Q}}(t)) = \sum_{i=1}^{n} \frac{1}{\rho_i^{\mathscr{Q}}(t)} \leq \max_{j=1}^{n} c_j \sum_{i=1}^{n} \frac{1}{c_i\rho_i^{\mathscr{Q}}(t)} \leq \max_{j=1}^{n} c_j = O(1),$$

which proves the contrapositive. ■

As a consequence of Observation 4.2, if our algorithm fails it will suffice to demonstrate that for any strategy $\mathscr{Q}$, that there exists some time t such that $\lambda(\boldsymbol{\rho}^{\mathscr{Q}}(t)) = \Omega(1)$. This will prove one part of Proposition 4.1. As a final remark, we note that the equivalent version of Observation 4.1 holds in our problem.

Observation 4.3. *Suppose there exists a strategy $\mathscr{Q}$ which maintains impedance zero on a set $A \subseteq P$ over the time interval M. Then, for all times $t \in M$, $\sum_{i\in A} \frac{1}{\rho_i^{\mathscr{Q}}(t)} \leq 1$.*

Proof. Fix a time t and consider $\rho_i^{\mathscr{Q}}(t)$. To maintain impedance zero, i must be queried within the next $\rho_i^{\mathscr{Q}}(t)$ time steps, regardless of how its position changes in the meantime. The claim then follows from Observation 4.1. ■

4.2 Lower Bounds on Round-Robin and Greedy Strategies

In this Section we demonstrate why certain naive strategies don't work. In particular, we demonstrate lower bounds on RoundRobin and two natural greedy approaches. Throughout this section, we will say that point i *exceeds its demand* if it has not been queried in $\rho_i^{\mathscr{Q}}(t)$ time steps, meaning that its uncertainty region is covering the origin.

The first greedy strategy we examine is the first approach one might try: simply query the next point which will exceed its demand. If some points have already exceeded their demand, query the point which has exceed their demand by the most. We call this strategy NaiveGreedy. It is formalized as Algorithm 2.

Algorithm 2 NaiveGreedy

Input: Set of points P, time horizon T.
for all $t \in T$ **do**
Query point which exceeds its demand by the largest margin, breaking ties arbitrarily
If there is no such point, query the point which will next exceed its demand.
end for

Following is a set of points and trajectories demonstrating that both querying points greedily or in round-robin fashion does much worse than the optimal algorithm.

Proposition 4.2. *There exists sets of points such that both* RoundRobin *and* NaiveGreedy *strategies achieve $\Delta_{Max} = \Omega(n)$ while $\Delta_{Max}^{\text{OPT}} = O(1)$.*

Proof. Consider a set of n points with unchanging demands defined as follows. Let $x_1(t) = x_1 \equiv 1$ and $x_i(t) = x_i \equiv n$ for all i and all times t. Consider the following strategy. Query point 1 every 2 time steps and with every other time step, perform round-robin like querying on the rest of the

points. For this strategy, point 1 is queried every $2 = 2x_1$ time steps, and point i for $i \geq 2$ is queried every $2n = 2x_i$ time steps. Therefore, $\Delta^{\text{OPT}}(t) \leq 2$ for all t.

Now, consider the behaviour of RoundRobin on this point set. Point 1 would be queried every $n = \frac{nx_1}{2} = \Omega(n)d_1$ time steps. Thus $\Delta^{\mathscr{R}}_{\mathsf{Max}} = \Omega(n)$. The SimpleGreedy strategy meanwhile, queries point 1 for the first $n-2$ time steps. At this time, all points are equally close to exceeding their demand. During the next n time steps therefore, n distinct points are queried. Suppose point 1 is queried at step c of the next n time steps. Thus, it exceeds its margin by $\max\{n-c, c\}$ at some time. If $c = o(n)$, then $n - c = \Omega(n)$. Otherwise, $c = \Omega(n)$. Hence $\max\{n-c, c\} = \Omega(n)$. Therefore, $\Delta^{\text{NaiveGreedy}}_{\mathsf{Max}} = \Omega(n)$. ■

Based on the example given in the proof above, one might wonder if we can develop a slightly more clever greedy algorithm which avoids ply $\Omega(n)$. In particular, instead of querying the points whose uncertainty regions overlap the origin by the largest margin, we might query the point which is (perceived to be) closest to the origin. In the given example, this would translate to querying point 1 every two time steps and would achieve an optimal impedance. We call this strategy ALittleLessGreedy—it is formalized as Algorithm 3.

Algorithm 3 ALittleLessGreedy

Input: Set of points P, time horizon T.
  for all $t \in T$ **do**
    if there are points which have exceeded their demands **then**
      Let $G \subseteq P$ be this set of points.
      Query point in G with the smallest demand, i.e., point $i = \text{argmin}_{i \in G}\{\rho_i(t)\}$
    else
      Query the point which is closest to exceeding its demand.
    end if
  end for

Unfortunately, it is straightforward to dash our hopes that this algorithm holds water. In fact, it can easily be shown to be *worse* (the horror!) than both RoundRobin and NaiveGreedy over a sufficiently long time horizon.

Proposition 4.3. *There exists a set of points P on which* ALittleLessGreedy *achieves $\Delta_{Max} = \Omega(|T|)$.*

Proof. Let $x_1(t) = x_2(t) = 1$ and $x_i(t) = n$ for $i \geq 3$ and all t. The strategy which queries 1 and 2 every three time steps and uses the third time step to round-robin query the other $n-2$ points achieve $\Delta_{\mathsf{Max}} = 3$. Therefore, $\Delta^{\text{OPT}}_{\mathsf{Max}} = O(1)$. However, ALittleLessGreedy would never query any of the points $\{3, \ldots, n\}$, as points 1 and 2 would always take precedence. The result follows. ■

4.3 The Bucket Algorithm

Seeing as Section 4.2 decimated our hopes of providing a simple algorithm for this problem, we now turn to developing a more sophisticated strategy. Luckily for us however, we have prior work on ply minimization from which we can borrow useful machinery. In particular, the "bucket algorithm"

—first proposed in Daniel Busto's M.Sc thesis [Bus15] and recently extended and polished by Busto et al. in [BEK18]—provides the perfect framework. This rest of the chapter introduces this bucket algorithm, BucketQueuing, and proves its optimality for our problem.

Recall that $T = \{t_0, \dots, t_m\}$. To simplify the discussion, we assume throughout this section that $|T| = 2^L$ for some $L \in \mathbb{N}$. We begin by providing some intuition behind BucketQueuing. The algorithm proceeds by constructing L partitions of T. The ℓ-th partition, B^ℓ, is composed of the intervals $B_{j,\ell} = \{t_0 + j2^\ell, t_0 + (j+1)2^\ell - 1\}$ for $j = 0, \dots, 2^{L-\ell}$. For every j, $B_{j,k}$ has length 2^k and $B_{j,k} \cap B_{j',k}$ for all $j \neq j'$. We call each $B_{j,k}$ a *bucket*. We call a bucket B *smaller* (resp., *larger*) than bucket B' if its associated time interval is shorter (resp., longer). As well as representing an interval of time, bucket $B_{j,k}$ will have an associated list of points which we call the bucket's *queue*. We will often say *place a point in bucket* B, meaning that the point should be placed in B's associated queue.

Intuitively, the BucketQueuingPromise follows a simple scheme. At every time step, it looks for the smallest non-empty bucket. It then removes a point from its queue, queries it, and places it in the queue of a bucket with length proportional to its current location. Importantly, the bucket in which it is placed is the next bucket of the appropriate size which *does not intersect* the current time step. In this way, as time progresses, we are clearing out buckets which contain the current time and moving the points to buckets associated with times further in the future. Furthermore, queues associated with smaller buckets have a higher priority than queues associated with larger buckets since they contain points which are closer to the origin.

The algorithm fails if at any time t, there is a bucket containing only times $t' < t$ which has a non-empty queue. In this case, there were too many points in this bucket (or in smaller buckets) and the algorithm could not clear them all in time. This will translate nicely into a proof that the demands of the points were simply too big at some point, and that no algorithm could have avoided high impedance. The algorithm is formalized as Algorithm 4.

Algorithm 4 BucketQueuing

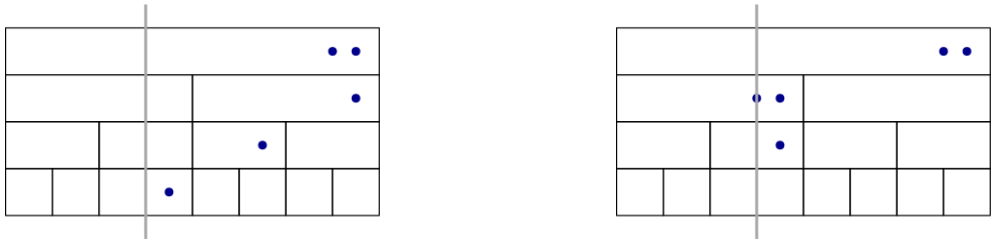
Input: Set of points P, a time horizon T.

for $t = t_0$ to t_m **do**
  if there exists a non-empty bucket $B_{j,\ell}$ such that $2^\ell(j+1) < t$ **then**
    FAIL.
  end if
  Dequeue i from smallest non-empty bucket
  Query i
  Set $q_i(t) = \lfloor \log(x_i(t)) \rfloor$
  Place i in next bucket $B_{h,q_i(t)}$ where $h = \lceil t/q_i(t) \rceil$.
end for

The remainder of the section will prove the following proposition, which, when combined with Observation 4.2, proves Proposition 4.1. Throughout these proofs, we will drop the superscripts on the perceived locations and uncertainty regions of the points. It will be understand that these quantities are in reference to the strategy BucketQueuing.

Proposition 4.4. *If* BucketQueuing *never fails, then* $\mathrm{rad}(u_i(t)) = O(\rho_i(t))$ *for all times* t *and points* i. *If it fails, then for any query strategy* $\mathcal{Q}$, *there exists a time* t *such that* $\lambda(\boldsymbol{\rho}^{\mathcal{Q}}(t)) = \Omega(1)$.

The forward direction of Proposition 4.4 is easy to see given the following observation.



(a) (b)

Figure 4.1: An example of two executions of BucketQueuing. The light gray line depicts the current time step. The smallest row of buckets are each a single time step. In (a), the algorithm will query the element in the smallest bucket and successfully move to the next time step. In (b) however, there are too many points in small buckets and the algorithm will fail.

Observation 4.4. *Suppose that i is in bucket B at a particular time during the course of the algorithm. Then i has been queried within the previous $2|B|$ time steps.*

Proof. When i was last queried at time t_i, it was placed in the next bucket of length $|B|$ which didn't overlap t_i. Thus, this bucket could have started at most $|B|$ time steps from t_i. Hence the end of the bucket is at most $2|B|$ time steps from t_i. Either i is queried by the time this bucket ends, or the algorithm fails and stops. In either case, the conclusion holds. ■

The following Lemma completes the first part of the proof of Proposition 4.4.

Lemma 4.1. *Over the course of the algorithm,* $\mathrm{rad}(u_i(t)) = O(\rho_i(t))$ *for any* $i \in P$ *and* $t \in T$.

Proof. Fix a time t, and let t_i be the last time that i was queried. At time t_i, i is placed in the next bucket with length $2^{q_i(t_i)}$. By Observation 4.4 therefore, i is queried within the next

$$2 \cdot 2^{q_i(t)} \leq 2 \cdot 2^{\log(\rho_i(t))} = \rho_i(t),$$

time steps. Therefore, $t - t_i \leq 2\rho_i(t) = O(\rho_i(t))$. Noting that $\mathrm{rad}(u_i(t)) = t - t_i$ completes the proof. ■

We now proceed to proving the second part of Proposition 4.4. The following Lemma demonstrates that if the true locations of points are grouped such that their average density of over some time is large enough, that no algorithm can hope to maintain low impedance. The statement and intuition behind the Lemma are quite straightforward, although the proof is slightly technical.

When we say a group of dynamic points is *infeasible*, we mean that no algorithm can maintain impedance zero on the points.

Lemma 4.2. *Let $A \subseteq X$ be a set of points such that over some finite, contiguous time interval $M \subseteq T$,*

$$\sum_{t \in M} \sum_{i \in A} \frac{1}{x_i(t)} > |M|.$$

Then for any strategy $\mathcal{Q}$, $\lambda(\boldsymbol{\rho}^{\mathcal{Q}}(t)) > 1$ for some $t \in M$ and A is infeasible.

Proof. To see that A is infeasible, it suffices by Observation 4.3 to demonstrate that $\lambda(\boldsymbol{\rho}^{\mathscr{Q}}(t)) > 1$ for some $t \in M$ for any strategy $\mathscr{Q}$. Suppose, aiming for contradiction, that $\lambda(\boldsymbol{\rho}^{\mathscr{Q}}(t)) \leq 1$ for all $t \in M$. First we claim that for any $i \in A$,

$$\sum_{t \in M} \frac{1}{x_i(t)} \leq \sum_{t \in M} \frac{1}{\rho_i^{\mathscr{Q}}(t)}. \tag{4.2}$$

We construct a time interval $M_i = \{t_1, \ldots, t_m\} \supseteq M$ as follows. Let t_1 be the last time prior to the beginning of M at which i was queried. That is, $t_1 = \operatorname{argmax}_{t \leq \min(M)}\{t : \mathscr{Q}(t) = i\}$ (where $\min(M)$ denotes the first time in M). Let $t_m = \max(M)$. Thus M_i extends further backwards in time than M but ends at the same time as M. Now, we iteratively partition M_i into contiguous and disjoint subintervals $M_i(1), M_i(2), \ldots, M_i(k)$, where $M_i(j) = \{t_1^j, \ldots, t_b^j\}$ and is defined as follows. Set $t_1^1 = t_1$. For $j = 1, \ldots, k$ let t_b^j be $t - 1$ where t is the next time i is queried or $x_i(t) > x_i(t_1^j)$ (or t_m, if we are at the end of M_i). More formally,

$$t_b^j \triangleq \min\left\{ \operatorname{argmin}_{t \geq t_1^j}\{t : \mathscr{Q}(t+1) = i\},\ \operatorname{argmax}_{t \geq t_1^j}\{t : x_i(t+1) > x_i(t_1^j + 1)\},\ t_m \right\}.$$

Then set $t_1^{j+1} = t_b^j + 1$ and repeat. We claim that for all j, $\rho_i^{\mathscr{Q}}(t_1^j) \leq x_i(t_1^j)$. This can be seen by an easy induction on the subintervals. Recall that $t_1 = t_1^1$ was chosen such that $\mathscr{Q}(t_1) = i$. Therefore $\rho_i(t_1^1) = x_i(t_1^1)$ so the claim holds for $M_i(1)$. Suppose it holds for $M_i(j-1)$. By construction, $t_1^j = t_b^j - 1$ is either chosen such that $\mathscr{Q}(t_1^j) = i$, in which case the claim holds. Otherwise, it is chosen such that $x_i(t_1^j) > x_i(t_1^{j-1})$. Since i is not queried at t_1^j then by construction we have $\rho_i(t_1^j) = \rho_i(t_1^{j+1})$. Therefore, by the induction hypothesis,

$$\rho_i^{\mathscr{Q}}(t_1^j) = \rho_i^{\mathscr{Q}}(t_1^{j-1}) \leq x_i(t_1^{j-1}) < x_i(t_1^j),$$

which completes the induction.

Now, observing that $x_i(t_1^j) \geq x_i(t)$ (again, by construction) for all $t \in M_i(j)$ and recalling that $\{M_i(j)\}$ partitions M_i,

$$\sum_{t \in M} \frac{1}{x_i(t)} = \sum_{j=1}^{k} \sum_{t \in M \cap M_i(j)} \frac{1}{x_i(t)} \leq \sum_{j=1}^{k} \sum_{t \in M \cap M_i(j)} \frac{1}{x_i(t_1^j)} \leq \sum_{j=1}^{k} \sum_{t \in M \cap M_i(j)} \frac{1}{\rho_i^{\mathscr{Q}}(t_1^j)}. \tag{4.3}$$

We now note that for all j and for all $t \in M \cap M_i(j)$, $\rho_i^{\mathscr{Q}}(t) = \rho_i^{\mathscr{Q}}(t_1^j)$, otherwise there would have been a new subinterval constructed. Hence, $\sum_{t \in M \cap M_i(j)} \frac{1}{\rho_i^{\mathscr{Q}}(t_1^j)} = \sum_{t \in M \cap M_i(j)} \frac{1}{\rho_i^{\mathscr{Q}}(t)}$. Combining this fact with (4.3) yields

$$\sum_{t \in M} \frac{1}{x_i(t)} \leq \sum_{j=1}^{k} \sum_{t \in M \cap M_i(j)} \frac{1}{\rho_i^{\mathscr{Q}}(t)} = \sum_{t \in T} \frac{1}{\rho_i^{\mathscr{Q}}(t)},$$

demonstrating that (4.2) holds. Therefore, summing over all $i \in A$ in (4.2) gives

$$\sum_{t \in M} \sum_{i \in A} \frac{1}{x_i(t)} \leq \sum_{t \in M} \sum_{i \in A} \frac{1}{\rho_i^{\mathscr{Q}}(t)} \leq \sum_{t \in M} 1 = |M|,$$

contradicting the hypothesis. Thus, no such query strategy exists. This completes the proof. ∎

Typically, it will not be the case that we can find a set of points which satisfy the hypothesis of Lemma 4.2. Consequently, it is the following corollary which will prove more useful.

Corollary 1. *Let $A \subseteq X$ be a set of points such that over some finite, contiguous time interval $M \subseteq T$,*

$$\sum_{t\in M}\sum_{i\in A}\frac{1}{x_i(t)} > \frac{|M|}{\alpha},$$

for any $\alpha \geq 1$. Then for any strategy $\mathscr{Q}$, $\lambda(\boldsymbol{\rho}^{\mathscr{Q}}(t)) = \Omega(1)$ at some time $t \in M$.

Proof. Define an alternative set of trajectories as $\hat{x}_i(t) = \alpha^{-1}x_i(t)$. (Note that the speeds of these trajectories are still bounded by 1 since $\alpha \geq 1$, hence are well defined). Then

$$\sum_{t\in M}\sum_{i\in A}\frac{1}{\tilde{x}_i(t)} > |M|,$$

so by Proposition 4.2 for any query strategy $\mathscr{Q}$ there exists a time t such that $\lambda(\tilde{\boldsymbol{\rho}}^{\mathscr{Q}}(t)) > 1$, where $\tilde{\boldsymbol{\rho}}$ are the perceived locations of the points according to the trajectories $\tilde{\mathbf{x}}$. Then,

$$\lambda(\boldsymbol{\rho}^{\mathscr{Q}}(t)) = \sum_{i=1}^{n}\frac{1}{x_i(t)} = \frac{1}{\alpha}\sum_{i=1}^{n}\frac{1}{\tilde{x}_i(t)} = \frac{1}{\alpha}\lambda(\tilde{\boldsymbol{\rho}}^{\mathscr{Q}}(t)) > \frac{1}{\alpha} = \Omega(1).$$

Since $\mathscr{Q}$ was arbitrary, this completes the proof. ■

Finally, we are able complete the proof of Proposition 4.4 and demonstrate that any strategy would have had high demand if BucketQueuing failed. The idea is to use the fact that use an "uncleared" bucket to obtain a group of entities whose average perceived demand is high, show this implies that their true demand must have been relatively high and then apply Corollary 1.

Lemma 4.3. *Suppose* BucketQueuingPromise *fails. Then for any strategy $\mathscr{Q}$, there exists a time t such that $\lambda(\boldsymbol{\rho}^{\mathscr{Q}}(t)) = \Omega(1)$.*

Proof. Let $B_{j,\ell}$ be the bucket that failed. If multiple buckets failed at the same time, i.e., all were uncleared, let $B_{j,\ell}$ have the minimum length of all such buckets. Set $M = [j2^\ell + t_0, (j+1)2^\ell + t_0]$, so M is precisely the interval of time bucket $B_{j,\ell}$ occupies. For all $t \in M$, let $\mathcal{J}(t)$ denote the set of buckets with lengths at most 2^ℓ which intersect t. Set

$$\mathcal{J} = \bigcup_{t\in M}\mathcal{J}(t),$$

so $\mathcal{J}$ is the set of buckets whose associated time intervals are contained in M. Let $A \subseteq P$ be the set of points which were queried by the algorithm at some time during M. Fix $t \in M$. Notice that since each point is in precisely one bucket at all times, $|A| \geq \sum_{B\in\mathcal{J}(t)}|\{i : i \in B\}|$ and consequently,

$$\sum_{i\in A}\frac{1}{\rho_i(t)} \geq \sum_{B\in\mathcal{J}(t)}\sum_{i\in B}\frac{1}{\rho_i(t)}.$$

Summing over all t gives

$$\sum_{t\in M}\sum_{i\in A}\frac{1}{\rho_i(t)} \geq \sum_{t\in M}\sum_{B\in\mathcal{J}(t)}\sum_{i\in B}\frac{1}{\rho_i(t)}. \tag{4.4}$$

To simplify the right hand side of the above quantity, let $\kappa(B)$ be some function of a bucket B. Observe both that $|B|\kappa(B) = \sum_{t\in M} \mathbf{1}_{[B\in\mathcal{J}(t)]}\kappa(B)$ and $\sum_{B\in J} \mathbf{1}_{[B\in\mathcal{J}(t)]}\kappa(B) = \sum_{B\in\mathcal{J}(t)} \kappa(B)$. Therefore,

$$\sum_{B\in\mathcal{J}} |B|\kappa(B) = \sum_{B\in\mathcal{J}}\sum_{t\in M} \mathbf{1}_{[B\in\mathcal{J}(t)]}\kappa(B) = \sum_{t\in M}\sum_{B\in J} \mathbf{1}_{[B\in\mathcal{J}(t)]}\kappa(B) = \sum_{t\in M}\sum_{B\in\mathcal{J}(t)} \kappa(B),$$

which is the right hand side of (4.4) if we let $\kappa(B) = \sum_{i\in B} 1/(\rho_i(t))$. Hence, (4.4) becomes

$$\sum_{t\in M}\sum_{i\in A} \frac{1}{\rho_i(t)} \geq \sum_{B\in\mathcal{J}} |B| \sum_{i\in B} \frac{1}{\rho_i(t)}. \tag{4.5}$$

Now we proceed to relating $x_i(t)$ and $\rho_i(t)$. Suppose at time t that $i \in B \in \mathcal{J}$. By definition of the algorithm, we have $\lfloor \log(\rho_i(t)/\gamma) \rfloor = \log(|B|)$ (since i was placed in bucket B), hence

$$\log(\rho_i(t)) - 1 \leq \log(|B|) \leq \log(\rho_i(t)) + 1.$$

This implies both that $\rho_i(t) \leq |B|$ and that $2|B| \leq 4\rho_i(t)$. Furthermore, by Observation 4.4, i was queried within the last $2|B|$ time steps, thus

$$x_i(t) \leq \rho_i(t) + 2|B| \leq 5\rho_i(t). \tag{4.6}$$

Combining this fact with (4.5) gives

$$\sum_{t\in M}\sum_{i\in A} \frac{1}{x_i(t)} \geq \frac{1}{5}\sum_{t\in M}\sum_{i\in A} \frac{1}{\rho_i(t)} \geq \frac{1}{5}\sum_{B\in\mathcal{J}} |B| \sum_{i\in B} \frac{1}{|B|} = \frac{1}{5}\sum_{B\in\mathcal{J}}\sum_{i\in B} 1 > \frac{|M|}{5}, \tag{4.7}$$

where the last inequality comes from the fact that bucket $B_{j,\ell}$ was uncleared, so there were more entities in the buckets of $\mathcal{J}$ than there were time steps in M. Applying Corollary 1, we obtain that any query strategy would have high density sometime in M. ■

4.4 Approximations to the Threshold Problem

If there exists no query strategy which satisfies the conditions of (4.1), then a natural question is to wonder what an appropriate approximation would be. In this section we explore three natural approximations and present apparent roadblocks to each.

Additive Approximation. If no algorithm can maintain the invariant that no uncertainty region of any points intersect the origin, then we may want to limit the amount by which any uncertainty region overlaps it. That is, to minimize the radius of the maximum sized ball around o (recall that o is the origin) which is contained in another uncertainty region. More succinctly, to minimize

$$\text{argmax}_s\, \mathcal{B}_s(o) \subseteq u_i(t),$$

over all times t. This is equivalent to finding the minimum δ such that

$$\text{rad}(u_i^{\mathcal{Q}}(t)) \leq d(\rho_i^{\mathcal{Q}}(t), o) + \delta, \tag{4.8}$$

over all t. Here δ would be the impedance. While this is a reasonable goal, it seems rather infeasible in practice for the following reason. As was illustrated in Section 4.3, algorithms seeking the to

limit the size of the uncertainty regions have an intimate relationship with the density of points at a time t. Furthermore, this relationship is typically *asympotic*, i.e., it is related only via constant factors. However, for the additive approximation described by (4.8), demonstrating that the density is $\Omega(1)$ for an algorithm with a specific impedance of δ does not imply that any algorithm would require impedance $\Omega(\delta)$. (Note that this is what we did in Section 4.3). This is demonstrated by the following example.

Example 4.1. Suppose $\mathcal{X} = \mathbb{R}$ and let $x_i = n-1$ for all points. For $\delta = n+1$, we have $\sum_{i=1}^{n} \frac{1}{\delta + x_i} = 1/2$. However, an algorithm could maintain impedance 1 by simply querying each point in round-robin fashion.

Multiplicative Approximation. Instead of an additive approximation as above, we might be concerned about the multiplicative approximation, i.e., the minimum δ such that

$$\mathrm{rad}(u_i^{\mathcal{Q}}(t)) \leq \delta d(\rho_i^{\mathcal{Q}}(t), o). \tag{4.9}$$

In this case, it is tempting to think the bucket strategy could work the same as in Section 4.3 and that the bucket sizes would simply incorporate δ. To see a problem with this approach, consider the slightly simpler problem of simply trying to maintain impedance $O(\delta)$ or certifying that any algorithm admits impedance $\Omega(\delta)$ at some time (Section 4.3 solved this problem for $\delta = 1$). It is natural to want to take $q_i(t) = \lfloor \log(\delta x_i(t)) \rfloor$ in the bucket algorithm. However, the problem can be seen by considering Line (4.6) in the proof of Lemma 4.3. While it is still true that $x_i(t) \leq \rho_i(t) + 2|B|$ (since the equivalent of Observation 4.4 still holds), this gives that

$$\delta x_i(t) \leq \delta \rho_i(t) + \delta 2|B| \leq \delta \rho_i(t) + \delta^2 4 \rho_i(t).$$

The extra factor of δ prohibits line (4.7) from giving anything useful. Thus, for the multiplicative version of the problem, something more clever than the obvious extension of the bucket algorithm is required.

Minimum overlap. We might instead want to limit the number of uncertainty regions which are overlapping the origin at any one time. To elaborate, let us consider the static pinwheel problem once more. Let $\mathbf{v} \in \mathbb{N}^n$ and suppose that $\mathbf{v}$ is infeasible. If we were to "ignore" queries to certain entries in order to ensure that there there was a feasible scheduling for the remaining entries, which ones should we ignore? Intuitively, we would ignore the entries with the largest demand since they require more queries than the other entries. This leads to make the following conjecture.

Conjecture 4.1. *Given $\mathbf{v} \in \mathbb{N}^n$ re-label the entries such that $v_1 \leq v_2 \leq \cdots \leq v_n$. It is possible to maintain impedance p on $\mathbf{v}$ iff $(v_{p+1}, \ldots, v_n)$ is feasible.*

If this conjecture were true, then we would have an optimal algorithm for this "minimum overlap" problem in the static case. While the conjecture seems intuitively obvious, proving the result has caused severe headaches. We leave this as an open question.

Chapter 5

Conclusion

As explained in the introduction, previous work in what we have identified as dynamic and resource-limited environments has been mostly focused on ply minimization. The focus of this thesis has been to develop a general framework in which to develop and analyze algorithms in these environments, and then to explore several specific problems.

Attempting to generalize the notion of ply, we developed the notion of "impedance", by which we can judge an algorithm's performance. We also studied what we believe are two natural questions arising from the model. First, we examined what sort of algorithmic guarantees were possible when the goal is simply to estimate the value of a function whose input is the location of the points. We then shifted our perspective and examined the problem from a geometric viewpoint. This exposed a fundamental link between our model and the pinwheel scheduling problem. The problem we studied from this perspective, the threshold problem, seemed a natural extension of the pinwheel scheduling problem to a dynamic setting.

5.1 Future Work

There are, of course, a myriad of algorithmic questions to be investigated in this model. For our reader's sake, we outline only a few here.

To begin, there are several questions left open by this work. Most notably is Conjecture 4.1 in Chapter 4. While many approaches have been attempted to prove this result, none have proved correct. In Chapter 3, there is also the potential for proving a necessary condition on the functions in order for RoundRobin to be optimal. As was demonstrated by the results of Section 3.3, the conditions outlined in Propositions 3.1 and 3.2 were only sufficient. Besides questions left open by this work however, there are many exciting directions for future research.

Clustering. While optimal algorithms are known when we are trying to limit the "interference" of the uncertainty regions (i.e., ply minimization), a natural next step is to ask what happens if we partitions the points into disjoint sets and only penalize interference between points of different sets. The most natural interpretation of this is clustering; points can interfere with points in the same cluster but we try and maintain separation between distinct clusters.

Geometric Function Estimation. The functions we studied in Chapter 3 typically had values mapping to $\mathbb{R}$. But we can extend this question to more general functions—for example, computing the convex hull or the Delaunay triangulation of the points. For these cases however, it's not clear that the same evaluation objective that was used in Chapter 3 would be equally useful.

Probabilistic Motion. When developing the model, we made no assumptions on where the point was most likely to be within its uncertainty region. In reality though, it is often the case that even if we have imprecise data we have additional information about the data—be it historical trends or a probability distribution over the parameters. It would be interesting to try and incorporate probability distributions over the movement of points into the model. In this case, we might begin arguing about the expected impedance of a strategy, or about a specific impedance not being exceeded with high probability.

Bibliography

[ABDB+00] P.K. Agarwal, J Basch, M De Berg, L.J. Guibas, and J Hershberger. Lower bounds for kinetic planar subdivisions. *Discrete & Computational Geometry*, 24(4):721–733, 2000.

[ABRW91] Neil C Audsley, Alan Burns, Mike F Richardson, and Andy J Wellings. Hard real-time scheduling: The deadline-monotonic approach. *IFAC Proceedings Volumes*, 24(2):127–132, 1991.

[AdB05] M.A Abam and M de Berg. Kinetic sorting and kinetic convex hulls. In *Proceedings of the Twenty-first Annual Symposium on Computational Geometry*, SCG '05, pages 190–197, New York, NY, USA, 2005. ACM.

[AKS07] G. Alexandron, H. Kaplan, and M. Sharir. Kinetic and dynamic data structures for convex hulls and upper envelopes. *Computational Geometry*, 36(2):144 – 158, 2007.

[Alb03] S Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1):3–26, Jul 2003.

[BEK18] D. Busto, W. Evans, and D. Kirkpatrick. Minimizing interference potential among moving entities. Technical report, University of British Columbia, 2018.

[BIL09] J Boyar, S Irani, and K.S. Larsen. A comparison of performance measures for online algorithms. In *Workshop on Algorithms and Data Structures*, pages 119–130. Springer, 2009.

[Bus15] D. Busto. Continuous ply minimization. Master's thesis, The University of British Columbia, 7 2015.

[CC92] M. Y. Chan and F. Y. L. Chin. General schedulers for the pinwheel problem based on double-integer reduction. *IEEE Transactions on Computers*, 41(6):755–768, Jun 1992.

[CC93] M. Y. Chan and F. Chin. Schedulers for larger classes of pinwheel instances. *Algorithmica*, 9(5):425–462, May 1993.

[Cer03] Anton Cervin. Integrated control and real-time scheduling. *PhD Theses*, 2003.

[CT92] Y.J. Chiang and R. Tamassia. Dynamic algorithms in computational geometry. *Proc. IEEE 80*, pages 1412–1434, 1992.

[DB11] Robert I Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys (CSUR)*, 43(4):35, 2011.

[dBRS11] M de Berg, M Roeloffzen, and B Speckmann. Kinetic convex hulls and delaunay triangulations in the black-box model. In *Proceedings of the Twenty-seventh Annual Symposium on Computational Geometry*, SoCG '11, pages 244–253, New York, NY, USA, 2011. ACM.

[dBRS13] M de Berg, M Roeloffzen, and B Speckmann. Kinetic 2-centers in the black-box model. In *Proceedings of the Twenty-ninth Annual Symposium on Computational Geometry*, SoCG '13, pages 145–154, New York, NY, USA, 2013. ACM.

[DL78] Sudarshan K Dhall and Chung Laung Liu. On a real-time scheduling problem. *Operations research*, 26(1):127–140, 1978.

[EHK+08] T Erlebach, M Hoffmann, D Krizanc, M Mihal'Ák, and R Raman. Computing minimum spanning trees with uncertainty. *arXiv preprint arXiv:0802.2855*, 2008.

[EKLS13] W. Evans, D. Kirkpatrick, M. Loffler, and F. Staals. Competitive query strategies for minimising the ply of the potential locations of moving points. *29th ACM Symposium on Computational Geometry (SoCG)*, pages 155–165, June 2013.

[EKLS16] W. Evans, D. Kirkpatrick, M. Loffler, and F. Staals. Minimizing co-location potential of moving entities. *SIAM J. Computing*, 45(5), 2016.

[FC05] E.A. Feinberg and M.T. Curry. Generalized pinwheel problem. *Mathematical Methods of Operations Research*, 62(1):99–122, 09 2005.

[FL02] P.C. Fishburn and J.C. Lagarias. Pinwheel scheduling: Achievable densities. *Algorithmica*, 34(1):14–38, Sep 2002.

[GGN06] J Gao, LJ Guibas, and A Nguyen. Deformable spanners and applications. *Computational Geometry*, 35(1):2 – 19, 2006. Special Issue on the 20th ACM Symposium on Computational Geometry.

[Gui98] Leonidas J. Guibas. Kinetic data structures – a state of the art report, 1998.

[HMR+89] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: a real-time scheduling problem. In *[1989] Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences. Volume II: Software Track*, volume 2, pages 693–702 vol.2, Jan 1989.

[HRTV92] R. Holte, L. Rosier, I. Tulchinsky, and D. Varvel. Pinwheel scheduling with two distinct numbers. *Theoretical Computer Science*, 100(1):105 – 135, 1992.

[JNRS00] J Jiao, S Naqvi, D Raz, and B Sugla. Toward efficient monitoring. *IEEE Journal on Selected Areas in Communications*, 18(5):723–732, 2000.

[Kah91] S. Kahan. A model for data in motion. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, STOC '91, pages 265–277, 1991.

[KSS02] D. Kirkpatrick, J. Snoeyink, and B. Speckmann. Kinetic collision detection for simple polygons. *International Journal of Computational Geometry and Applications*, 12(1-2):3–27, 2002.

[LH06] Hsin-hung Lin and Chih-Wen Hsueh. Applying pinwheel scheduling and compiler profiling for power-aware real-time scheduling. *Real-Time Systems*, 34(1):37–51, Sep 2006.

[LL73] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[LSST02] Chenyang Lu, John A Stankovic, Sang H Son, and Gang Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems*, 23(1-2):85–126, 2002.

[Rud64] W Rudin. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1964.

[SAA+04] L Sha, T Abdelzaher, K Arzen, A Cervin, T Baker, A Burns, G Buttazzo, M Caccamo, J Lehoczky, and A.K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Syst.*, 28(2-3):101–155, November 2004.

[Spe08] B. Speckmann. *Kinetic Data Structures*, pages 1–99. Springer US, Boston, MA, 2008.

[ST85] D Sleator and R Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[Suy12] S. Suyadi. Computing functions of imprecise inputs using query models. Master's thesis, The University of British Columbia, 2012.

[Tab07] Paulo Tabuada. Event-triggered real-time scheduling of stabilizing control tasks. *IEEE Transactions on Automatic Control*, 52(9):1680–1685, 2007.

[YLH04] S. Yu, H. Lin, and C. Hsueh. An application using pinwheel scheduling model. In *Proceedings. Tenth International Conference on Parallel and Distributed Systems, 2004. ICPADS 2004.*, pages 683–689, July 2004.