Computing Functions of Imprecise Inputs using Query Models

by

Simon Aloysius Suyadi

BASc., The University of British Columbia, 2009

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES
(Computer Science)

The University Of British Columbia
(Vancouver)

April 2012

© Simon Aloysius Suyadi, 2012

Abstract

Suppose we want to compute some function (such as convex hull or k-th smallest element), but the input values are imprecise. Can we compute the answer? Perhaps we need some of the input values to be more precise. What is the smallest additional input precision we need for each input to compute the function? We explore a model in which a query to an input allows us to uncover one more "unit" of its precision, at unit cost. Unfortunately, we cannot predict the results of a query in advance. This motivates us to study online algorithms that attempt to minimize the number of queries to compute the function.

We compare the cost of online algorithms against the minimum query cost to compute the function. We obtain lower bounds on the ratio of these costs for a variety of simple functions, and create algorithms with matching upper bounds. We also consider a kinetic model in which the results of a query become more imprecise over time (i.e., the inputs move) and our goal is to compute the function of the inputs at some fixed time.

Table of Contents

Abstract . . . . **ii**

Table of Contents . . . . **iii**

List of Figures . . . . **iv**

Acknowledgments . . . . **v**

1 Introduction **1**
- 1.1 Input and Query 2
  - 1.1.1 Kinetic Input Model 2
  - 1.1.2 Static Input Model 3
- 1.2 Computing Functions and Certificates 3
- 1.3 Online Algorithm and Competitive Analysis 4
- 1.4 Naive Algorithm 5
- 1.5 Report All Queries and Refinements 5
- 1.6 Uniqueness and Order Relations 7

2 Related Work **8**
- 2.1 Kinetic Input Model 8
- 2.2 Static Input Model 9

3 Kinetic Input Model **12**
- 3.1 Understanding the Problem in 1-Dimension 13
- 3.2 Minimum Element 14

3.3 Sorting 15

4 Static Input Model 17

4.1 Minimum Element 17

4.2 Sorting 18

4.3 Set of k Smallest Elements 19

4.4 k-th Smallest Element 21

4.5 Extreme Elements 22

4.6 Mode Problem 24

5 Multiset Model 26

5.1 Mode Problem 27

5.1.1 QueryAllMode 28

5.1.2 QueryMaxSpan 29

6 Conclusion 31

6.1 Future Work 32

Bibliography 33

List of Figures

Figure 3.1 Trajectory Function $F(t,x)$ 13

Figure 3.2 A pair of elements, in which element $i+1$ moves away from element i. 15

Figure 4.1 Top-Down view. The small circles are spherical regions. The black dots are point elements. The gray dashed lines show that each of the spherical regions are guaranteed to be part of the convex hull. The large circle and the gray line are drawn for clarity. 23

Figure 4.2 Side view. The small circles are spherical regions. The black dots are point elements. The gray lines are drawn for clarity. . 24

Figure 5.1 Proof of Lemma 5.1.3. The lines are regions and the dots are point regions. The crosses are the multiset of final points; there are three final points located at coordinate -1. 28

Figure 5.2 Proof of Lemma 5.1.4. The lines are regions and the dots are point regions. The crosses indicate one possible multiset of final points. The circles indicate another possible multiset of final points. In both multisets, there are three final points located at coordinate -1. 29

Acknowledgments

First and foremost I would like to express my deepest gratitude for my supervisor, Will Evans, for guiding and helping me through the entire degree. Without him I would not have accomplished this far.

I would like to thank Maarten Löffler from University of California, Irvine, for providing many ideas and suggestions.

I would also like to thank David Kirkpatrick for agreeing to read my thesis and giving helpful suggestions and improvements.

Finally, I would like to thank my family and friends for their support and care.

Chapter 1

Introduction

Whenever a measurement is not exact, imprecision exists. In this thesis, two different models are discussed: kinetic imprecision and static imprecision.

Kinetic imprecision is due to objects moving. Any measurements, even precise ones, at a fixed point in time will become less precise as time passes. Movement of real world objects are rarely confined to a strict trajectory. Fortunately, most objects on earth have a finite maximum speed.

Even when an input element is completely static, what we know about it is imprecise. This imprecision is known as static imprecision. In the real world, measurement instruments have limited precision. Obtaining more precise measurement requires more resources. For example, more measurements need to be taken, or better instruments need to be used. Sometimes, imprecise data are used because they have simpler representation, for example, using fewer bits.

In computing geometric functions, we often ignore the imprecision and assume that the input is exact. This is because the imprecision is tiny compared to the scale of the input. Thus we obtain an answer that is exact, or very close to the true value. But what happens if the imprecision is significant?

In order to compute a function, we do not always require precise data for every element. It is sometimes possible, based on imprecise data, to discard data that have no impact on the final answer. For example, in computing a nearest neighbour, one can easily discard elements that are too far from other elements. It is also possible that the full precision of every element is not always needed. In the nearest

neighbour example, we only need to know enough precision so that the nearest element is distinguishable from the next nearest element. Thus having imprecision inspires us to wonder what precision is required to compute a function.

So how can we measure the amount of precision used by an algorithm? The model we use is a query based model. In this model, we can perform an action on an imprecise element to uncover more precision. This action is called a ”query”.

1.1 Input and Query

The input I consists of many elements in Euclidean space in fixed dimension. Each element has a final point and a continuous region. Although the final point is initially hidden to us, the region bounds its location; the final point is guaranteed to lie inside it. Thus an element's region represents what we currently know about its final point. A query to an element (or we might say to a region) reduces the size of the region. There are two types of input models: kinetic and static.

1.1.1 Kinetic Input Model

In kinetic input model, we want to calculate a function of the input at a future time t_0. We will separate the time into two phases.

The first phase happens before t_0. We can make queries for free, subject to the limitation that we can make only one query at every unit time interval. A query at time t in this phase returns the exact position of the element at time t. We assume that every input has bounded speed v. Since we define the imprecise region of an element to be relative to t_0, the imprecise region of an element queried at time t is a hypersphere of radius $v \cdot \max(0, t_0 - t)$ at time t_0.

The second phase happens from t_0 onwards. In this phase, a query will return the exact position of the element at time t_0. There is no limit to the number of queries that can be made. However, in this phase, each query has a unit cost.

The first phase exists to allow a good algorithm to ”preprocess” the input, that is, to make intelligent queries in order to minimize the number of queries required in the second phase.

1.1.2 Static Input Model

In the static input model, the elements do not move, and thus imprecision does not increase with time. This is how the kinetic input model behaves after time t_0. However, unlike in that model, we do not restrict a query to always return a final point. We allow a query to an element to return a new region that is a shrunken version of the old region. The new region is always a subset of the old region, and must contain the final point.

The region can be various shapes depending on the dimension and type of the problem. In one dimension, a region is an interval, represented by its upper and lower bounds. In higher dimensions, there exist multiple shapes to represent imprecision. One common shape is a hypersphere, represented by a center and radius. The radius denotes the degree of error or imprecision. Another common shape is an orthotope, a hyperrectangle or a box. In this shape, an element has one interval for each dimension.

A special case of the static input model is the shrink-to-point model. This is similar to the second phase of the kinetic input model. A query to an element will return the final point immediately.

1.2 Computing Functions and Certificates

The problems described here are simple common functions that normally take in input points. These include sorting, k-th element, k-partitioning, extreme elements, and finding the mode.

If the inputs are uncertain then their actual value may be unknown and we can identify them only as, for example, the fifth input element. Our answer to a problem is then the identity of the element that is the k-th smallest, or an order of the elements that is sorted. To insure that our answer is correct, we must guarantee that no matter what the actual values of the elements are, our answer remains correct.

To be more precise, we can define a consistent input as follows. Given a set of elements and their associated regions R, a consistent input J is a set of final points such that there exists a one-to-one mapping of a region in R to a final point in J. To solve a problem means to make a sequence of queries until the family of all consistent inputs will give the same answer. This sequence of queries constitutes a

certificate.

In the kinetic input model, the ordering of the queries matters, because time affects the imprecision of the element. In the static input model, the imprecision of the element is only dependent on the total number of queries made to that element.

1.3 Online Algorithm and Competitive Analysis

The algorithms described here are in a sense online algorithms. This is because only portions of the entire input I are initially provided. An online algorithm needs to use queries on elements to reveal more information about the input I, and makes more queries based on the query responses, until the problem is solved.

Simple worst-case analysis is not suitable for the algorithms described here. This is because we can generate a bad input I for which every algorithm must query all elements. For example, suppose we are interested in finding the minimum element (the element whose final point has the minimum coordinate). Create n elements. The i-th element has region $[i, 2n]$, and final point $2n - i$. Assume that a query to an element will return the final point. As you can see, the n-th element is the minimum element. A certificate requires a query of all elements, this is because all other elements have regions that may contain final points less than the n-th element's final point, and thus must be queried.

The most common way to analyse online algorithms is through a comparative method, called competitive analysis. The idea was first mentioned by Sleator and Tarjan [13]. A more detailed explanation is available in the book by Borodin and El-Yaniv [2].

Suppose we are given an algorithm A and input I. Let $cost(A, I)$ be the total query cost of algorithm A on input I. In competitive analysis, one common way to define $\rho(n)$-competitiveness is as follows.

Definition 1.3.1. *An algorithm A is $\rho(n)$-competitive against another algorithm B if there exist an additive constant c, such that for every input I of size n, $cost(A, I) \leq \rho(n) \times cost(B, I) + c$.*

$\rho(n)$ can be a constant, or a function of n. Note that our competitive analysis focus on the worst case scenario of deterministic algorithm. This distinguishes us

from other analysis that use randomized algorithm. Also, note that unlike most competitive analysis literature, the input I here is not a sequence, but rather elements which can be queried in the way described previously.

But what do we compare an algorithm against? In the most common competitive analysis, we define an offline optimal algorithm that knows everything about the input I. The task of the optimal algorithm is to produce the shortest sequence of queries to generate a certificate. A good algorithm performs comparatively well against an optimal algorithm.

1.4 Naive Algorithm

In both input models, there always exists a naive algorithm that is n-competitive against an optimal algorithm. We define algorithm NAIVE as follows. In the kinetic input model, NAIVE will wait until time t_0 before querying all the n elements. Thus, NAIVE will always require a cost of n queries.

In the static input model, NAIVE simply queries all elements at each step. Thus, every query made by an optimal algorithm will correspond to at most n queries by NAIVE. Thus, NAIVE will know at least as much information about the input. If the optimal algorithm solves the problem, so will NAIVE. This proves for all the problems in the static input model, NAIVE is n-competitive against the optimal algorithm.

1.5 Report All Queries and Refinements

Now that we have introduced competitive analysis, we return to our simple problem in the static input model: finding the minimum element. The minimum element is the element whose final point is the smallest or leftmost. As pointed out by Khanna and Tan, we obtain the following result.

Lemma 1.5.1. *[9] No online query algorithm that reports and certifies one minimum element under the static input model is better than n-competitive against an optimal algorithm.*

Proof. Create n elements having the interval $[0,1]$. Let us assume for simplicity that we are in the shrink-to-point model and a query to an element will return a final

point. Only one of the elements, when queried, will return a final point 0. Other elements, when queried, will return a final point 1. In the worst case, an online algorithm queries the wrong $n-1$ elements before querying the right element. Thus it ends up querying all the elements to find the answer. □

Thus, as in the example shown here, in the worst case, an online algorithm will end up doing more queries than the optimal algorithm. In the example above, it turns out that the optimal algorithm has certified only *one* possible solution; it has not proven that it is the *only* solution. Although the online algorithm has made more queries, it has certified more than the optimal algorithm. If it makes n queries, then it has certified that the other $n-1$ elements cannot be the answer. Thus, the queries made by the online algorithm give us useful information about the uniqueness of the answer.

Also unfortunately, making use of randomized algorithm and analysing the expected cost on the input does not seem to help. In the example above, suppose we use a randomized algorithm that will arbitrarily query an element that has not been queried before. The probability of having any number of tries is $\frac{1}{n}$, thus the expected number of tries is $\frac{1}{2}(n+1)(n)\frac{1}{n} = \frac{n+1}{2}$. This is still not a good competitive ratio against an optimal algorithm.

This motivates an important modification of the requirements, that is to require that all possible answers be reported: the queries must prove that no other possible solutions exist. Thus, in the event that some set of elements are candidates for the answer, then some queries need to be done in order to prove or disprove that the elements are part of the answer. Such a change in the requirements of the problems can lead to a change in the competitive ratio. For example, to find the minimum element, we can make use of k-partitioning with $k=1$. As shown later in Section 4.3, there exists an online query algorithm that solves not just the minimum element, but the k-th smallest element in general, with competitive ratio 2.

Lastly, even with this added requirement, not all problems have algorithms with very good competitive ratio. One such problem is the mode problem, described later in the chapter. The standard model of competitive analysis may not be sufficient to analyse competitive ratio. One idea is to define another algorithm which is less powerful than the optimal, but still knows additional information about the

input I compared to an online algorithm. This algorithm, called MULTISET, will know the multiset of final points, but not the mapping of the final points to elements. This model of competitive analysis will be further discussed in Chapter 5.

1.6 Uniqueness and Order Relations

As stated previously, all answers should be reported, because there may be more than one possible answer. Or even if the answer is unique, the uniqueness has to be proven.

Multiple answers usually arise because of multiplicity in the final points which are related to the answer. In the minimum element example above, the answer is not unique if there exist more than one element whose final point is 0.

Nevertheless, the term "minimum" suggests that some kind of ordering exists. In one dimension, we define an order relation on elements (i.e. intervals) that take into account multiplicity.

Given two elements i and j.

- $i \prec j$ if and only if the rightmost point of i lies to the left of the leftmost point of j.
- $i \equiv j$ if and only if both elements' final points are known and coincide.
- $i \preceq j$ if the rightmost point of i lies on or to the left of the leftmost point of j.

It should be noted that $i \prec j$ or $i \equiv j$ implies $i \preceq j$, however $i \preceq j$ can be true even if neither $i \prec j$ nor $i = j$ is true. A simple case would be $i = [0,1]$ and $j = [1,2]$. Also, the ordering between elements i and j is defined if and only if either $i \equiv j$, $i \prec j$ or $j \prec i$. Note that, from this definition, the ordering between two elements i and j is defined if and only if we can certify either they are both distinct or identical.

Chapter 2

Related Work

The analysis of online algorithms was first studied by Sleator and Tarjan [13]. Borodin and El-Yaniv wrote a book [2] that further explains competitive analysis. Since standard competitive analysis may produce unsatisfactory results,

Dorrigiv and López-Ortiz [5] reviewed and compared various performance measures for online algorithms. Many of these alternative measures attempt to weaken the power of the optimal algorithm. We also consider a weakened optimal algorithm in Chapter 5 in an attempt to obtain better competitive bounds for the mode problem.

2.1 Kinetic Input Model

Much research has been done that deals with moving objects. For example, one very popular model is the (KDS) kinetic data structure framework [1]. But unlike our model, there is no imprecision at all, because the trajectory of each element is not only fully known, but also typically linear. KDS research emphasises minimizing some computation cost to maintain a structure like the convex hull. Lastly, KDS related research does not have the concept of queries.

An early model that is related to this work is the paper by Simon Kahan [8]. Similar to the model here, the input is a set of objects which are moving in real time. One can make queries (called updates) to several elements in order to get their exact locations at the time a user requests some function of the elements. Our model

allows queries prior to the *known* time t_0 when the function should be compute. However, there can be only one query per unit time. In some ways, Kahan's model is more similar to our static model since multiple queries are made at the time of the user's request in order to compute the function, without considering their benefit to answering future requests.

The idea of preprocessing comes from the paper by Buchan, Löffler, Morin and Mulzer [4]. In ther paper, imprecise input is initially available. Preprocessing is done on the imprecise input so that when the more precise data is provided at a later time, less computation is needed. We focus on minimizing the number of queries required, while they focus on minimizing the computation time.

2.2 Static Input Model

Research on imprecision has been around for a long time. One such early work related to imprecision is ε-geometry [12]. The framework tries to cope with computational errors in geometric algorithms that arise from the use of finite precision arithmetic. We view imprecision as arising from the input rather from computation and we allow more precision at a cost.

The main idea for the static model described in this paper has been earlier mentioned in the paper by Kirkpatrick [10], and in a subsequent paper by Kirkpatrick and Tseng [14]. In both papers, Kirkpatrick and Tseng described a model in which an algorithm is provided input numbers up to some precision, and additional precision can be obtained with a cost. An algorithm that can solve the problem using as little input precision as possible is called an *input-thrifty algorithm*. In [10] and [14], the query model need not be restricted to the "bit model". The notion of intrinsic cost applies to a more general query model. To quote from [14] " Although the restriction of accessing input information one bit at a time in order of decreasing significance is perfectly natural, one could also adopt a more general model in which individual inputs are represented as a sequence of nested uncertainty intervals. It turns out all of our results apply in this more general model. Nevertheless, we choose to first develop our results in the more restrictive, but less cumbersome, LIB-cost model. "

Later in Chapter 5, we will talk about the MULTISET algorithm. The MULTI-

SET algorithm is a weaker optimal algorithm that knows only the multiset of final values but not their corresponding element. Its cost is the same as the *intrinsic cost* described by Kirkpatrick [10], and by Kirkpatrick and Tseng [14]. which takes into account the presentation of the input.

A query model similar to that of this thesis is also earlier described by Olston and Widom [11]. Here an interesting replication system is proposed; a cache stores imprecise data and its values can be obtained cheaply. Precise data for each element can be obtained from a different source using a query, but at a high cost. Like this thesis, we want to compute some function, such as finding the minimum, maximum and summation, while minimizing the query cost.

Unlike this thesis, Olston and Widom allow an imprecise answer as long as it is within a specified tolerance. For example, in the k-th smallest element problem, a user can specify a tolerance parameter p. An answer to the problem will be any element whose actual rank differs at most p from the desired rank. Having a higher tolerance p may reduce the number of queries needed to solve the problem. Thus, a user can control the tradeoff between precision and query cost. Additionally, a single query to an element will give the final result, but the cost of the query may be different for different elements. The model proposed by Olston and Widom [11] appears in several related papers.

Khanna and Tan [9] use the Olston and Widom model and deal with the problems of selection, summation/averaging and composition of functions. Here, the notion of online query algorithm is defined, and in order to analyse such an algorithm, competitive analysis is used. Therefore, we encounter the problem as stated in Lemma 1.5.1 in that there exist certain inputs for which no online query algorithm can do well against the optimal.

Bruce, Hoffmann, Kirzanc and Raman [3] use the Olston and Widom model, online query algorithm and competitive analysis. They consider the problem of finding maximal points and convex hull.

Feder, Motwani, O'Callaghan, Olston and Panigrahy [6] also use the Olston and Widom model, and deal with the median element problem. However, unlike the other papers, they consider both offline and online query algorithms. In an online query model, an algorithm selects one element to query, receives a response, and if the problem is not solved, the process repeats. But in an offline query model,

an algorithm must specify *all* elements that it wants to query, and it must be that regardless of the query response, the problem must be solved. Note that the offline query model is not considered in this thesis. For such a model to apply an algorithm needs to know how many queries it needs to make to obtain the final value of an element.

The model described by Olston and Widom [11] can also be used for graph problems. Feder, Motwani, O'Callaghan, Olston and Panigrahy [7] deal with the shortest path problem. In this paper, the input is a graph, and the length of edges have imprecise values. An algorithm can query an edge to find its exact length. This paper is also different in that it only uses an *offline* query model.

The techniques we use in this thesis are based on a general scheme used in the papers above and identified by Bruce, Hoffmann, Kirzanc and Raman [3]. If we want to create an online algorithm that is k-competitive, for some integer k, we will prove that for every k queries made by this online algorithm, at least one will correspond to a query that must be made by the optimal algorithm. However, one difference to note is that in the static model, an element can be queried multiple times without changing its region.

Chapter 3

Kinetic Input Model

In this model, all elements are moving about in completely arbitrary direction with bounded maximum speed. One example is gas particles experiencing Brownian motion. We want to compute a function of the elements' positions at time t_0. However, we can only obtain the exact location of an element by querying it. If we wanted to minimize the total number of queries, we would wait until time t_0 before making queries, reducing the problem to the static case. Instead we want to minimize the number of queries that occur from time t_0 onwards, by making intelligent queries before time t_0. As mentioned in Section 1.1.1, the time before t_0 is called the first phase, and the time after that is called the second phase.

The input I is a set of elements, each with a trajectory whose maximum speed is bounded by v. Let $p_i(t)$ be the trajectory function of element i, i.e. the location of element i at time t. Let o_i be the last query (observed) time of element i (if an element has never been observed we can set $o_i = -\infty$). A query to an element in the first phase at time $t < t_0$ will return the exact location of the element at time t. A query to an element in the second phase at time $t \geq t_0$ will return the exact location of the element at time t_0. Since we want to compute a function at time t_0, the imprecision region r_i associated with the element is simply a hypersphere centered at $p_i(o_i)$ with radius $v \cdot \max(t_0 - o_i, 0)$,

We define a certificate of a solution to an input to be a set of last query times for each element, such that the imprecision region from the elements is sufficient to compute the function. The optimal algorithm knows the trajectories of all elements

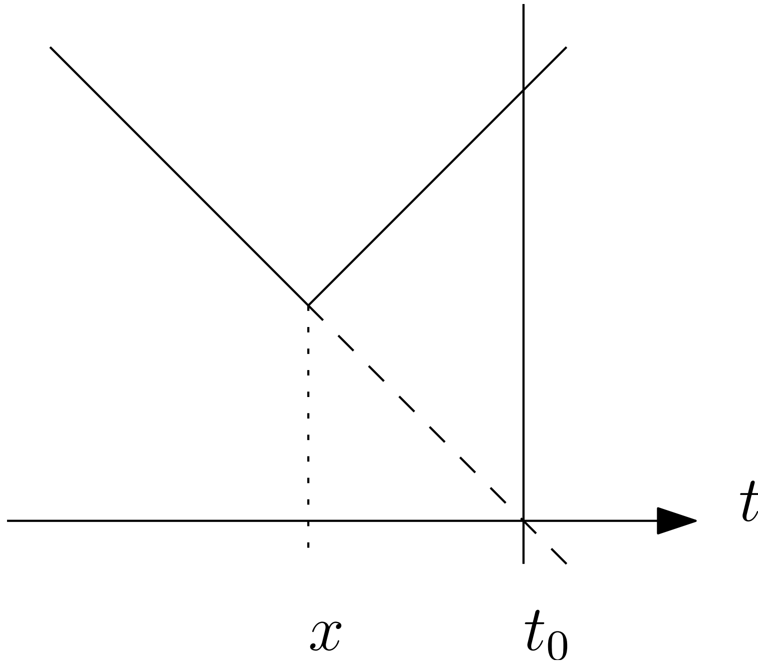

Figure 3.1: Trajectory Function $F(t,x)$

in advance. It only has to perform queries to certify the solution.

3.1 Understanding the Problem in 1-Dimension

For all inputs that are restricted to one dimension, we can create a 2-dimensional diagram, with the x-axis as time, and y-axis as position. Every element has a trajectory with maximum slope v. If a particle is last queried at o_i, we can draw two lines with slopes v and $-v$, starting from the particle location at o_i, to t_0. The triangle created by the 2 slopes and the line t_0 is the region of imprecision for element i. If element i is queried again, this triangle must be readjusted accordingly.

In the examples below, we make use of trajectory functions of the form $F(t,x) = v|t-x| - vx$ for different values of x. These functions describe a trajectory $p(t) = -vt$ for $t \leq x$, after which it reverses direction, changing from a slope of $-v$ to v. See Figure 3.1.

3.2 Minimum Element

Lemma 3.2.1. *No online algorithm that reports and certifies one of the minimum elements at time t_0 in a set of n kinetic elements is better than n competitive against an optimal algorithm.*

Proof. To make things simpler, let $t_0 = 0$. Create one element with trajectory $p(t) = -vt$, this will be the minimum element. Create $n-1$ elements with trajectory $p(t) = F(t, -1)$. If any elements are queried at time $t < 0$, we would get an imprecision region with lower bound 0. Thus no algorithm can distinguish the elements using queries for $t \leq -1$. In the worst case, an algorithm will end up querying all elements for $t \geq 0$, thus requiring n units of time. An optimal algorithm simply queries the correct minimum element at $t = 0$, taking only 1 unit of time. □

The situation is no better if all minimum elements are desired.

Lemma 3.2.2. *No online algorithm that reports and certifies all of the minimum elements at time t_0 in a set of n kinetic elements is better than n competitive against an optimal algorithm.*

Proof. To make things simpler, let $t_0 = 0$. Create n elements, in which the i-th element has trajectory $F(t, 1-i)$. Observe that the first element has trajectory $p_1(t) = -vt$ for $t \leq 0$. To an online algorithm, the i-th element (for $i > 1$) is indistinguishable from the first, unless the i-th element is queried at time $t \geq -i+1$.

Consider the latest query times for each element for $t \leq 0$. In the worst case, the element queried at time $-t$ by the algorithm happens to be the t-th element. Thus, the algorithm is unable to distinguish the i-th element from the first, and is required to query every element after time $t_0 - 1$, thus taking n units of time. The optimal algorithm simply queries the i-th element at time $t \geq i+1$. The last element queried is the first element at time $t = 0$, using 1 unit of time. Thus the online algorithm takes a factor of n more units of time after $t = 0$ compared to the optimal. □

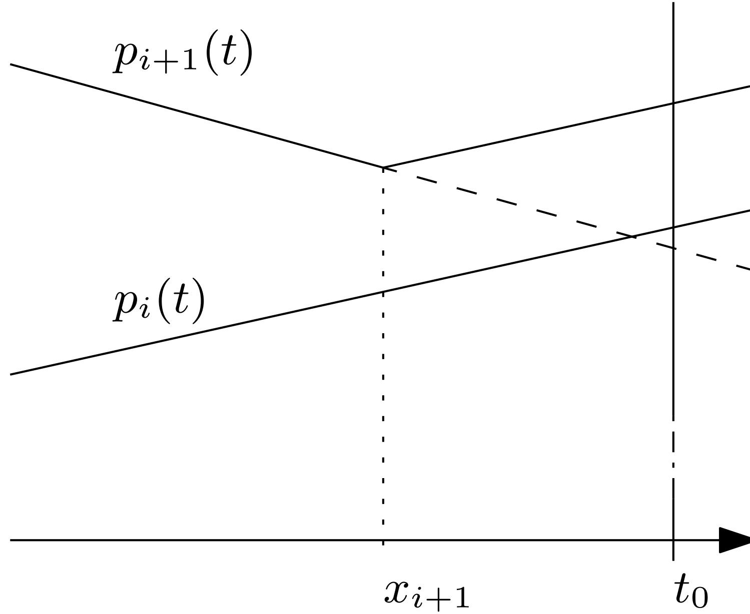

Figure 3.2: A pair of elements, in which element $i+1$ moves away from element i.

3.3 Sorting

In the sorting problem, we would like to find a total ordering of the elements at time t_0. Enough queries need to be made such that the ordering between any pair of elements is defined. As stated earlier, the ordering between elements i and j is defined if and only if either $i \equiv j$, $i \prec j$ or $j \prec i$.

Lemma 3.3.1. *No online algorithm that reports and certifies an element ordering at time t_0 of a set of n kinetic elements is better than n competitive against an optimal algorithm.*

Proof. Let $t_0 = 0$. Create n elements, where n is an even number. Let $p_i(t)$ denote the trajectory of the i-th element.

Group the n elements into pairs, in which a pair has consecutive elements. Choose m be the distance between different pairs to be much greater than $n \cdot v$ since we want the trajectories of different pairs not to intersect. We make the input such that, up until t_0, the input is consistent with the set of trajectories described as

follows. The pair of elements i and $i+1$ (i even) have default trajectories $p_i(t) = mi + vt + \varepsilon$ and $p_{i+1}(t) = mi - vt - \varepsilon$. Since the trajectories intersect just before t_0, no online algorithm, can separate the regions before t_0. Let x_i be the last query time for element i before t_0 by an online algorithm.

Note that the actual trajectories for any element i between x_i and t_0 is not known to the online algorithm, and thus we are able to create any trajectory we want in that time frame, as long as the maximum velocity is bounded. For the pair of elements i and $i+1$, we either make element i move away at time x_i, or make element $i+1$ move away at time x_{i+1}. See Figure 3.2.

From t_0 onwards, the online algorithm still needs to order the pair. However, the online algorithm does not known which of the elements in each pair moves away. In the worst case, for each pair, the online algorithm will query the element that does not move away. For the pair of elements i and $i+1$, if element i does not move away, a query will result in coordinate $mi+\varepsilon$, and if element $i+1$ does not move away, a query will result in coordinate $mi-\varepsilon$. This means that the online algorithm is still unable to order the elements in the pair. In the worst case, the online algorithm will perform n queries starting from time t_0.

If i does not move away, it retains its default trajectory which is $p_i(t) = mi + vt + \varepsilon$, otherwise, it has $p_i(t) = mi - F(t, x_i) + \varepsilon$. Similarly, if $i+1$ does not move away, it retains its default trajectory which is $p_{i+1}(t) = mi - vt - \varepsilon$, otherwise, it has $p_{i+1}(t) = mi + F(t, x_{i+1}) - \varepsilon$.

An optimal algorithm knows the entire trajectory of all elements. Thus, in a pair of elements i and $i+1$, if i moves away, the optimal algorithm can query the element after time x_i. Similarly, if $i+1$ moves away, the optimal algorithm can query after time x_{i+1}. In this way, the optimal algorithm is able to order all regions.

Since $x_i < t_0$ for all i, the latest query made by the optimal algorithm is at time t_0. Thus, the optimal algorithm performs one query from time t_0. $\square$

Chapter 4

Static Input Model

As you can see from the previous section, having points that move does not allow an algorithm to achieve a very good competitive ratio. In this section, we consider input elements whose underlying points are stationary.

In some of the problems, when only one solution is required, no online algorithm can hope to do well against the optimal algorithm. By changing a problem to require every correct algorithm to report and certify *all* possible answers, an online algorithm may be able to achieve a good competitive ratio against the optimal algorithm.

4.1 Minimum Element

Many of the problems stated here require one to know some form of ordering relation between two or more elements. The simplest case is the problem of finding the minimum element. As stated in Lemma 1.5.1, an online query algorithm that needs to report and certify one of many minimum elements can be n-competitive in the worst case. However, when we require that all possible minimum elements be reported and certified, the lower bound changes.

Lemma 4.1.1. *No online query algorithm that reports and certifies all possible minimum elements from a set of n elements is better than 2-competitive against an optimal algorithm.*

Proof. Create one element with interval $[0,2]$ and another element with interval

$[1,3]$. Note that the elements intersect at $[1,2]$. One of the elements, when queried, will always give the same interval. The other element will return the same interval when queried the first $m-1$ times, and an interval outside of $[1,2]$ afterwards.

An online query algorithm is unable to distinguish which of the two elements will give an interval outside of their intersection. Thus, in the worst case, the online query algorithm will take at least $2m$ queries to find which of the elements is the minimum. The optimal algorithm, on the other hand, queries the correct element m times. □

Note that the competitive ratio is 2 for any input size n, where $n \geq 2$. When $n = 2$, we are trying to find the ordering of the two elements. As mentioned in section 1.6, finding ordering is equivalent to certifying distinctiveness. Thus, we can also provide the following lemma:

Lemma 4.1.2. *No online query algorithm that certifies if two elements are distinct or identical is better than* 2*-competitive against an optimal algorithm.*

We will later show in Section 4.4 that 2-competitiveness is achievable in a more generalized problem of finding the k-th smallest element.

4.2 Sorting

We would like to perform the minimum number of queries in order to determine a total order on the input elements. As mentioned in Section 1.6, finding the ordering between elements is equivalent to certifying distinctiveness. Thus, the equivalent problem is to certify if every pair of elements is distinct or identical. By definition, there can only be one possible answer.

Lemma 4.2.1. *No online query algorithm that reports and certifies the total order on n elements is better than* 2*-competitive against an optimal algorithm.*

Proof. Since we need to certify if every pair of elements is distinct or identical, we can make use of the proof from Lemma 4.1.2 □

Let QueryOverlappingPair be an algorithm that arbitrarily chooses an overlapping pair of elements and queries both of them until no pairs overlap.

Lemma 4.2.2. *Algorithm QueryOverlappingPair reports and certifies the total order on n elements and is 2-competitive against an optimal algorithm.*

Proof. Suppose we pause QueryOverlappingPair after it queries a pair of overlapping i and j elements. Let q_i and q_j be the number of times QueryOverlappingPair has queried elements i and j. Since QueryOverlappingPair queried elements i and j, they must overlap when they are queried less than q_i and q_j times respectively. If elements i and j overlap and no further query is made to them, the ordering will not be known. Thus the optimal algorithm needs to query element i at least q_i times, or element j at least q_j times.

This means that, for every pair of elements i and j that algorithm QueryOverlappingPair queries, the optimal needs to match the number of queries of at least one of them. Thus for every pair of elements that QueryOverlappingPair queries, the optimal algorithm must query at least one of them. Thus algorithm QueryOverlappingPair is 2-competitive. □

4.3 Set of k Smallest Elements

In this problem, we would like to partition the input into two sets A and B, such that for all $a \in A$ and for all $b \in B$, $a \preceq b$ and $|A| = k$. We can extend the results from Lemma 1.5.1 as follows.

Lemma 4.3.1. *No online query algorithm that must report and certify one set of k smallest elements (of possibly many) from a set of n elements is better than* $n-k+1$*-competitive against an optimal algorithm.*

Proof. Create $n-k+1$ ranges $[1,2]$. Create $k-1$ points at coordinate 0. Only one of the $n-k+1$ ranges, when queried, will return a point at coordinate 1, the others will return a point with coordinate in $(1,2]$. To certify the k smallest elements, an algorithm must determine the minimum element among the $n-k+1$ ranges. This reduces to the minimum element problem, and by Lemma 1.5.1 no online query algorithm can be better than $n-k+1$ competitive against an optimal algorithm. □

For this model in which only one solution need be certified, Khanna and Tan

[9] obtain a p-competitive algorithm where p is the maximum number of regions (intervals) that share a common point. They also show that this is optimal.

The situation changes if we require a correct algorithm to certify the existence of all solutions. When k is 1, the problem becomes the minimum element problem. Thus by Lemma 4.1.1, no online query algorithm that reports and certifies all possible sets of k smallest elements is better than 2-competitive to the optimal algorithm.

Let l_i, r_i be the lower(left) and upper(right) bounds of element i. Let $l_{[k]}$ be the k-th smallest lower(left) bound. Let $r_{[k]}$ be the k-th smallest upper(right) bound.

Lemma 4.3.2. *For any k, there exists at least one element that has a region covering from $l_{[k]}$ to $r_{[k]}$.*

Proof. Let i be the element whose region contains $l_{[k]}$ that has the largest upper bound, we show that $r_i \geq r_{[k]}$. Observe that there exists at least k lower bounds at or to the left of $l_{[k]}$. Among all the regions with lower bounds $\leq l_{[k]}$, including i, i has the rightmost upper bound. Thus, there exist at least $k-1$ upper bounds at or to the left of r_i. Therefore, $r_{[k]}$ is at r_i, or to the left of it. Thus, element i covers from $l_{[k]}$ to $r_{[k]}$. □

Let p_i be the location of the final point of element i. Let $p_{[k]}$ be the location of the final point of (one of) the k-th smallest element. By definition, $l_{[k]} \leq p_{[k]} \leq r_{[k]}$. In other words, the final point of the k-th smallest element must lie in $[l_{[k]}, r_{[k]}]$. If we are not done, then there is no separation between the k-th smallest element and the $(k+1)$-th smallest element. In terms of region bounds, $l_{[k+1]} \leq r_{[k]}$.

Let i be any element that has a region covering from $l_{[k]}$ to $r_{[k]}$. Let j be any element that has a region covering from $l_{[k+1]}$ to $r_{[k+1]}$. By Lemma 4.3.2, i and j always exist (they may or may not be the same element). Let KPart be an online algorithm that finds such a pair of elements, i and j, queries both (if i and j are the same element, then the element is queried twice) and repeats until we can certify the answer.

Lemma 4.3.3. *Algorithm KPart reports and certifies all solutions to the first k smallest elements problem for a set of n elements and is* 2*-competitive against an optimal algorithm.*

Proof. Let i and j be a pair of elements chosen by KPart. Since the union of regions of elements i and j cover $[l_{[k]}, r_{[k+1]}]$, then $p_{[k]}$ and $p_{[k+1]}$ must be covered by this union. If we do not ever query i or j at their current state, we cannot certify the answer. Thus any algorithm, even the optimal, must query one of i, j at their current state. Thus every query made by KPart corresponds to at least one query made by the optimal algorithm. Thus, KPart is 2-competitive. □

4.4 k-th Smallest Element

In this problem, we would like to partition the input into three sets A, $\{i\}$ and B, such that (1) $|A| = k-1$, (2) for all $a \in A$, $a \preceq i$, and (3) for all $b \in B$, $i \preceq b$. The results from the set of k smallest elements problem can be applied here.

Since the minimum element problem is a specialization of the k-th smallest element problem, from Lemma 4.1.1, no online query algorithm that reports and certifies all possible k-th smallest elements from a set of n elements is better than 2-competitive against an optimal algorithm. Let DoubleKPart be an online algorithm that uses KPart to find the set of k smallest elements, and then uses KPart to find the set of (k+1) smallest elements making use of the queries made by the first run of KPart.

Lemma 4.4.1. *Algorithm DoubleKPart reports and certifies all solutions to the k-th smallest element problem for a set of n elements and is* 2*-competitive against an optimal algorithm.*

Proof. Notice that the k-th smallest element problem is solved if and only if both the set of k smallest elements problem and the set of (k+1) elements problem are solved. From the proof of Lemma 4.3.3, we know that every pair of queries made by KPart corresponds to at least one query made by the optimal algorithm to solve the set of k smallest elements problem. Therefore, every pair of queries made by DoubleKPart must correspond to at least one query made by the optimal algorithm to solve either the set of k smallest elements problem or the set of (k+1) smallest elements problem, or both. Therefore, DoubleKPart is 2-competitive. □

4.5 Extreme Elements

An extreme element is an element whose final points is extreme. In this problem, we want to find the set of all extreme elements. The problem in 2D has already been dealt with by [3]. According to their paper, if the input is restricted to the closure of open, connected areas or trivial areas, no online query algorithm that reports and certifies the set of extreme elements is better than 3-competitive. Additionally, they have provided an online query algorithm that is 3-competitive to the optimal.

We demonstrate that any online algorithm will perform badly against an optimal adversary in 3D. This is true even if we restrict the regions to be non-overlapping spheres.

Lemma 4.5.1. *No online query algorithm that must report and certify the set of extreme elements from a set of n elements in three or higher dimensions is better than $(n-4)$-competitive.*

Proof. Consider the situation in three dimensions. Let n be an even number. Draw a large semicircle in the plane $z=0$, let R be the radius of this semicircle. Place two point regions at the ends of the semicircle. Evenly place $n-4$ spherical regions of radius r on the circumference of the circle, not touching the two previous points, with the centers of the spheres having $z=0$. The radius r must be small enough, $r < \frac{R}{2}(1-\cos\frac{\pi}{2n})$, so that for each of the spherical regions, one can draw a plane that separates the region from all other elements. Place a point region near the center of the semicircle, but slightly inside the semicircle, with the point having z coordinate 0. Label this point region as i. Place a point region at the center of the semicircle again, but with z coordinate less than $-r$. See Figures 4.1 and 4.2.

Notice that region i is an extreme element if and only if all of the $n-4$ spherical regions has a final point $z<0$. We can set up the input such that a query will return a final point, and only one of the $n-4$ regions has a final point with $z>0$. Thus any algorithm, in the worst case, has to query all $n-4$ regions in order to find it, while the optimal only needs to query the right one. □

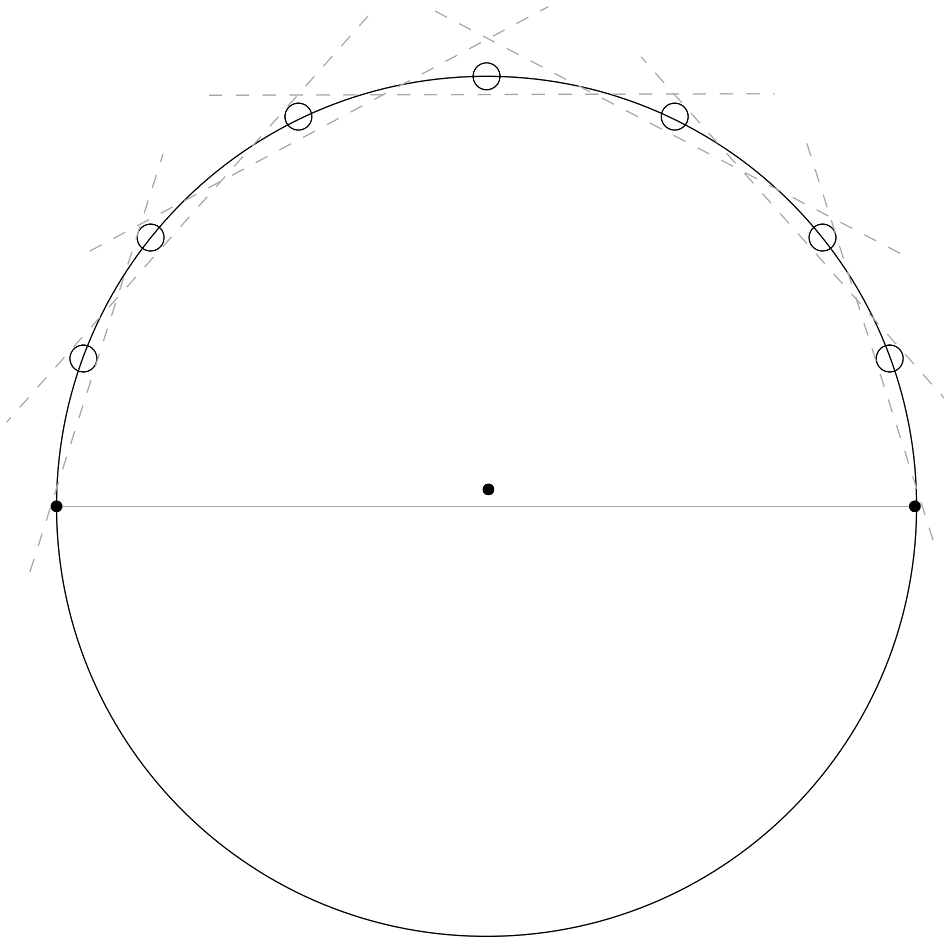

Figure 4.1: Top-Down view. The small circles are spherical regions. The black dots are point elements. The gray dashed lines show that each of the spherical regions are guaranteed to be part of the convex hull. The large circle and the gray line are drawn for clarity.

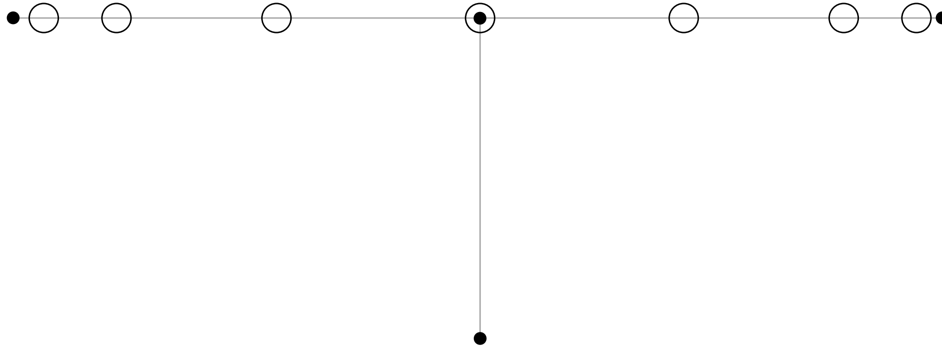

Figure 4.2: Side view. The small circles are spherical regions. The black dots are point elements. The gray lines are drawn for clarity.

4.6 Mode Problem

We want to find one coordinate that has a maximum number of final points in a set of n elements. This coordinate is called a mode of the set.

Lemma 4.6.1. *No online query algorithm that reports and certifies one mode (of possibly many) of a set of n elements is better than $(n/2)$-competitive against an optimal algorithm.*

Proof. Create $n/2$ pairs of overlapping regions. The i-th pair is $[6i, 6i+2]$ and $[6i+1, 6i+3]$. Only one of the pair has final points which overlap. In the worst case, an algorithm needs to query all pairs, while the optimal algorithm only needs to query the right pair. □

Suppose we require all modes.

Lemma 4.6.2. *No online query algorithm, even one that knows k, that reports and certifies all point modes of a set of n elements is better than k-competitive against an optimal algorithm, where k is the multiplicity of the mode.*

Proof. Let $k \geq 1$. Create k elements with point regions at coordinate $2k$. Create k elements, in which the i-th element has region $[i-k, 2+i]$. Note that the size of the mode is k, and an algorithm will have to certify if there is a mode in $[0, 2]$. One element, when queried, will return a final point outside of $[0, 2]$, while the rest will return a final point at coordinate 1. In the worst case, an online query algorithm

will have to query all k regions. An optimal algorithm needs only one query to the right element. □

Let QueryAllMode be an algorithm that finds a coordinate that intersects the largest number of regions, and queries all elements that intersect this coordinate. Notice that QueryAllMode does not know k, the multiplicity of the mode.

Lemma 4.6.3. *QueryAllMode reports and certifies all point modes of a set of n elements and is k-competitive (with additive constant $2k-2$) against an optimal algorithm, where k is the multiplicity of the mode.*

Proof. Let p be a point chosen by one step of the QueryAllMode algorithm. Suppose p is a mode. If multiple modes exist in the input, then all algorithms, even the optimal, must query all elements at that point. But this is not true if only one mode exists. There must be less than $k-1$ regions extending to the left from p that the optimal algorithm does not query, otherwise it is possible for k such regions to form a new mode. There also must be fewer than $k-1$ regions extending to the right. Thus, in total, there can be at most $2k-2$ regions intersecting p that the optimal does not query. This gives the additive constant in the competitive ratio.

Suppose p is not a mode and is contained in $m \geq k$ elements. To prove that p is not a mode, an optimal algorithm must query some elements until there are at most $k-1$ elements whose regions intersect p. The number of elements queried by the optimal algorithm is at least $m-k+1$, while QueryAllMode queries exactly m elements. Note that $m \leq k(m-k+1)$, with equality when $m=k$. Thus, for the m elements queried by QueryAllMode, at least $m-k+1$ of the elements are queried by the optimal algorithm in their current state.

In both cases, QueryAllMode maintains its k-competitiveness. □

Chapter 5

Multiset Model

In some of the problems, no online algorithm has good competitive ratio. Perhaps, using the standard model of competitive analysis is not sufficient, this motivates us to adopt a more precise competitive framework. We can define an algorithm that knows more about the input, but at the same time knows less than the optimal algorithm.

Let MULTISET be an algorithm that performs the fewest number of queries to certify an answer knowing the multiset of final points but not the exact mapping between these points and the input elements. When MULTISET queries an element, the final point that is returned may be any one of the multiset of final points that is consistent with the element so that a mapping between the remaining points and elements still exists. We consider the number of queries performed by an online algorithm versus the number performed by MULTISET.

We will use this MULTISET model to solve a problem in which an online algorithm cannot achieve a good competitive ratio. In particular, we pick the mode problem, since no online algorithm can do well against the optimal algorithm, even if we consider the problem where we require certification of all answers.

Since MULTISET is no better than the optimal algorithm, we can always achieve n competitiveness against MULTISET by using the NAIVE algorithm as described in Section 1.4.

5.1 Mode Problem

Lemma 5.1.1. *No online query algorithm that reports and certifies one of possibly many point modes from a set of n elements is better than $(n/2)$-competitive against MULTISET.*

Proof. We will use the example from Lemma 4.6.1. Then, we provide MULTISET with the multiset of final points. Notice that there exists a unique one-to-one mapping from each final point in the multiset to each element. Thus, MULTISET is able to identify which pair contains the point mode, and is thus able to query the right pair. □

Now suppose we require that all possible modes be reported and certified.

Lemma 5.1.2. *No online query algorithm that reports and certifies all possible point modes from a set of n elements is better than 2-competitive against MULTISET.*

Proof. Create a pair of elements whose coordinate is 0. This establishes the mode to be at least 2. Create n pairs of elements in which the i-th pair has regions $[4i, 4i+2]$ and $[4i+1, 4i+3]$. Observe that the i-th pair intersects at $[4i+1, 4i+2]$. Thus, the task of any algorithm is to prove or disprove if a mode exists in each of these n pairs. Let A be the set of these n pairs of elements.

The multiset of final points can be set up as follows. For the i-th pair from A, one of the elements has a final point inside $[4i+1, 4i+2]$, while the other does not. Notice that there exists a one-to-one mapping between an element and a final point from the multiset. Thus, for the i-th pair from A, MULTISET simply queries the element which is mapped to a final point outside of $[4i+1, 4i+2]$. Thus, for each pair, MULTISET requires one query.

An online query algorithm does not know this mapping, and has to guess which of the two elements from each pair from A have a final point outside of $[4i+1, 4i+2]$. In the worst case, for each pair from A, an online query algorithm will need two queries to disprove that a mode exists. Thus, an online query algorithm is 2-competitive at best. □

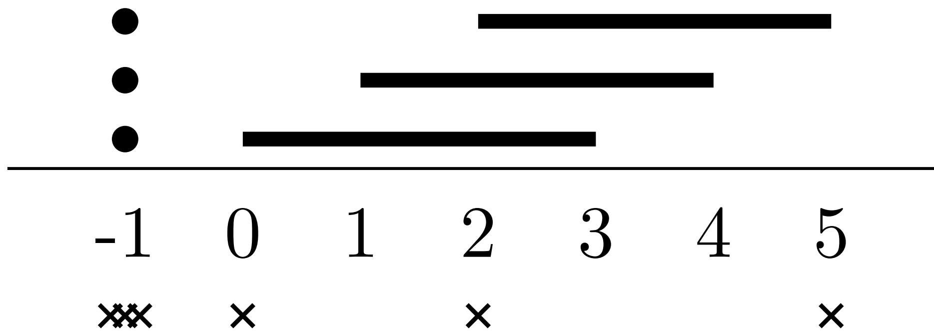

Figure 5.1: Proof of Lemma 5.1.3. The lines are regions and the dots are point regions. The crosses are the multiset of final points; there are three final points located at coordinate -1.

Having established a lower bound of 2-competitiveness, the next intuitive step is to create an algorithm that hopefully has the same upper bound. Unfortunately, QueryAllMode is not 2-competitive against MULTISET.

5.1.1 QueryAllMode

What happens if we apply algorithm QueryAllMode from Section 4.6 to this problem?

Lemma 5.1.3. *QueryAllMode reports and certifies all point modes of a set of n elements and is no better than* 3*-competitive against MULTISET.*

Proof. Create elements regions $[0,3]$, $[1,4]$ and $[2,5]$. Create 3 elements with point regions at coordinate -1.

From the 3 elements at coordinate -1, we know that the mode size is 3. The task of any online algorithm is thus to look at $[2,3]$, and report and certify if there exists another mode. QueryAllMode will query all the elements intersecting $[2,3]$, thus taking up three queries.

We can simply create a multiset of final points: $-1,-1,-1,0,2,5$. Note that there exists a one-to-one mapping of final points to regions. MULTISET can query the element with region $[0,3]$ to obtain a final point of 0, or query the element with region $[2,5]$ to obtain a final point of 5. See Figure 5.1.

Thus, QueryAllMode uses at least three queries, while MULTISET uses only

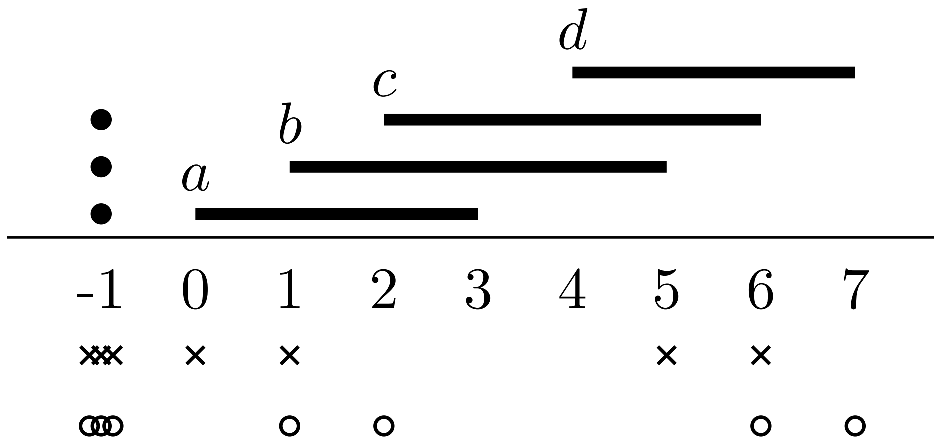

Figure 5.2: Proof of Lemma 5.1.4. The lines are regions and the dots are point regions. The crosses indicate one possible multiset of final points. The circles indicate another possible multiset of final points. In both multisets, there are three final points located at coordinate -1.

one. We can make as many independent (non-overlapping) copies of this example as we wish to show that competitiveness is at least a factor 3. □

Since QueryAllMode is proven to be k-competitive against the optimal algorithm, where k is the size of the mode not informed to QueryAllMode. QueryAllMode must also be no worse than k-competitive against MULTISET.

5.1.2 QueryMaxSpan

Perhaps we can do better than querying *all* the elements that overlap the coordinate contained in the most elements. In the spirit of the QueryOverlappingPair algorithm, we might choose two of these elements in the hope that MULTISET must query at least one of them. Let p be a coordinate contained in the largest number of elements. QueryMaxSpan queries a pair of elements containing p: one that stretches farthest left and one the stretches farthest right. Ties can be broken arbitrarily. In the previous example, algorithm QueryMaxSpan would query elements $[0,3]$ and $[2,5]$, which would return the final points 0 and 5, thus solving the problem using only two queries. It turns out that no competitive ratio better than 3 can be achieved using QueryMaxSpan.

Lemma 5.1.4. *QueryMaxSpan reports and certifies all point modes of a set of n elements and is no better than* 3*-competitive against an MULTISET algorithm.*

Proof. Create element a with region $[0,3]$, element b with region $[1,5]$, element c with region $[2,6]$ and element d with region $[4,7]$. Create three elements with point regions at coordinate -1. From the three elements at coordinate -1, we know that the mode size is 3. The task of QueryMaxSpan is to certify if that is the only point mode. See Figure 5.2.

Suppose QueryMaxSpan picks a coordinate p from $[2,3]$. Then a and c will be chosen and queried. Among the family of input values consistent, there exists a multiset of final points with coordinates: $-1,-1,-1,0,1,5,6$ (see the crosses in Figure 5.2). Thus, a query to a and c must return coordinates 0 and 5 respectively. QueryMaxSpan is not done because there still exists a possible point mode at coordinate 5. MULTISET queries b and obtains 1 as final point, certifying that there is no mode at $[0,7]$.

By symmetrical argument, QueryMaxSpan can also pick a coordinate p from $[4,5]$, in which it will query b and d. We can create the multiset of final points with $-1,-1,-1,1,2,6,7$ (see the circles in Figure 5.2). A query to b and d must return 2 and 7 respectively. QueryMaxSpan is not done because there still exists a possible point mode at coordinate 2. MULTISET queries c and obtains 6 as final point, certifying that there is no mode at $[0,7]$.

Thus, QueryMaxSpan uses at least three queries, while MULTISET requires only one. □

We are not sure if either QueryAllMode or QueryMaxSpan has 3-competitiveness against MULTISET. As of this point, the question that is still open is whether there exists an online algorithm that can achieve better than k-competitiveness against MULTISET. Lastly, note that in both Lemma 5.1.3 and Lemma 5.1.4, we came up with examples in which MULTISET is able to figure out the mapping between the elements and final points.

Chapter 6

Conclusion

We have looked at a model in which we are given inputs which are imprecise, but we can make queries to improve the precision. In this model, a query to an element allows us to uncover more information about the element's precision, at a unit cost. Unfortunately, we cannot predict the results of a query in advance.

The model can be divided into two types, kinetic and static. In the kinetic input model, a query to an element will provide the full trajectory up to the time of the query. In this model, we want to compute some functions at a specific time in the future. It turns out that no online algorithm has good competitive ratio against the optimal algorithm for the problems we considered.

The other model is the static input model. A query to an element simply gives more precision. We explored problems such as minimum element, sorting, set of k smallest elements, convex hull and the mode problem.

In both models, we compare the cost of online algorithms against the minimum query cost to compute the function. We obtained lower bounds on the ratio of these costs for a variety of simple functions. In the static input model, we looked at algorithms with matching upper bounds for the problems of sorting, finding the set of k smallest elements, finding the k-th smallest element and finding the extreme elements. In the kinetic input model, it turns out that in the worst case, no algorithm can do significantly better than the naive algorithm, which does not take advantage of the preprocessing available in the first phase.

Since the standard model of competitive analysis may not be sufficient, we

also studied the multiset model, with focus on the mode problem. Unfortunately, the question that is still open is whether there exists an online algorithm that can achieve 2-competitiveness against the MULTISET algorithm.

6.1 Future Work

More research can be done on the multiset model. For example, we can try to apply this multiset model on the kinetic input model.

Another idea is that, in the kinetic input model, we can simply focus on the first phase and measure the amount of value the preprocessing gives. Instead of considering how many more queries are needed in the second phase to answer the question, we can consider how much ”work” has been done in the first phase. In order to do that, we define a metric to indicate how close we are to the answer. For example, in the sorting problem, one possible metric is the number of permutations consistent with the input. We compare algorithms by how much this metric is reduced in the preprocessing phase.

We can also further advance the kinetic input model. Instead of computing a function only at time t_0, perhaps we want the function to be computed at for a certain increasing sequence of times $t_0, t_1, t_2, \ldots$ etc.

We can also try to apply the idea of tolerance from related papers such as Olston and Widom [11]. In other words, we accept an answer as long as it is within some tolerance of the correct answer.

Perhaps, we can try to think of other ways to analyse the static input model. Or perhaps, instead of worst-case analysis, we can analyse the cost in terms of some probability distribution. Or perhaps, there are other ways to come up with an algorithm that knows more than an online algorithm, but less than the optimal algorithm, other than MULTISET.

There are also other variations of the mode problem that can be explored. For example, we can look at the problem in two or three dimensions. Or we can consider the problem of making the fewest number of queries to certify that the mode has modality less than k.

Bibliography

[1] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1 – 28, 1999. ISSN 0196-6774.

[2] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998. ISBN 0-521-56392-5.

[3] R. Bruce, M. Hoffmann, D. Krizanc, and R. Raman. Efficient update strategies for geometric computing with uncertainty. *Theor. Comp. Sys.*, 38: 411–423, July 2005. ISSN 1432-4350.

[4] K. Buchin, M. Löffler, P. Morin, and W. Mulzer. Preprocessing imprecise points for delaunay triangulation: Simplified and extended. *Algorithmica*, 61:674–693, 2011. ISSN 0178-4617.

[5] M. Chrobak. Sigact news online algorithms column 8. *SIGACT News*, 36(3): 67–81, Sept. 2005. ISSN 0163-5700.

[6] T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. Computing the median with uncertainty. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, STOC '00, pages 602–607, New York, NY, USA, 2000. ACM. ISBN 1-58113-184-4.

[7] T. Feder, R. Motwani, L. OCallaghan, C. Olston, and R. Panigrahy. Computing shortest paths with uncertainty. In H. Alt and M. Habib, editors, *STACS 2003*, volume 2607 of *Lecture Notes in Computer Science*, pages 367–378. Springer Berlin / Heidelberg, 2003. ISBN 978-3-540-00623-7.

[8] 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, New York, NY, USA, 1991. ACM. ISBN 0-89791-397-3.

[9] S. Khanna and W.-C. Tan. On computing functions with uncertainty. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS ’01, pages 171–182, New York, NY, USA, 2001. ACM. ISBN 1-58113-361-8.

[10] D. Kirkpatrick. Hyperbolic dovetailing. In A. Fiat and P. Sanders, editors, *Algorithms - ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 516–527. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-04127-3.

[11] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB ’00, pages 144–155, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-715-3.

[12] D. Salesin, J. Stolfi, and L. Guibas. Epsilon geometry: building robust algorithms from imprecise computations. In *Proceedings of the fifth annual symposium on Computational geometry*, SCG ’89, pages 208–217, New York, NY, USA, 1989. ACM. ISBN 0-89791-318-3.

[13] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, Feb. 1985. ISSN 0001-0782.

[14] K.-C. Tseng and D. Kirkpatrick. Input-thrifty extrema testing. In T. Asano, S.-i. Nakano, Y. Okamoto, and O. Watanabe, editors, *Algorithms and Computation*, volume 7074 of *Lecture Notes in Computer Science*, pages 554–563. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-25590-8.