VERY EFFICIENT APPROXIMATION ALGORITHMS TO EDIT DISTANCE PROBLEMS

BY TIMOTHY RYAN NAUMOVITZ

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Mathematics

Written under the direction of

Michael Saks

and approved by

____

____

____

____

New Brunswick, New Jersey

October, 2016

ABSTRACT OF THE DISSERTATION

Very Efficient Approximation Algorithms to Edit Distance Problems

by Timothy Ryan Naumovitz

Dissertation Director: Michael Saks

This thesis deals with the question of approximating *distance to monotonicity* in the streaming setting as well as the task of approximating the *ulam distance* between two permutations. Both of these problems are variants of the *edit distance* problem which, given two input sequences, is the minimum number of insertions and deletions (and in some cases, substitutions) needed to transform one sequence into the other.

The *distance to monotonicity* of a sequence of n numbers is the minimum number of entries whose deletion leaves an increasing sequence. We give the first deterministic streaming algorithm that approximates the distance to monotonicity within a $1+\varepsilon$ factor for any fixed $\varepsilon > 0$ and runs in space polylogarithmic in the length of the sequence and the range of the numbers. The best previous deterministic algorithm achieving the same approximation factor required space $\Omega(\sqrt{n})$ [13]. Previous polylogarithmic space algorithms were either randomized [22], or had approximation factor no better than 2 [9]. We also give polylogarithmic space lower bounds for this problem: Any deterministic streaming algorithm that gets a $1+\varepsilon$ approximation requires space $\Omega(\frac{1}{\varepsilon}\log^2(n))$ and any randomized algorithm requires space $\Omega(\frac{1}{\varepsilon}\frac{\log^2(n)}{\log\log(n)})$.

The *Ulam distance* between two permutations of length n is the minimum number of insertions and deletions needed to transform one sequence into the other. We provide an algorithm which, for any fixed $\varepsilon > 0$, gives a $(1+\varepsilon)$-multiplicative approximation for the Ulam distance d in $\tilde{O}_\varepsilon(n/d + \sqrt{n})$ time, which has been shown to be optimal up to polylogarithmic factors. This is the first sublinear time algorithm (provided that $d = (\log n)^{\omega(1)}$) that obtains arbitrarily good multiplicative approximations to the Ulam distance. The previous best bound is an $O(1)$-approximation (with a large constant) by Andoni and Nguyen [4] with the same running time bound (ignoring polylogarithmic factors).

Acknowledgements

Completing this thesis was a difficult task and, for better or for worse, could not have been accomplished if it were not for many of the people around me and the things they have done to help me throughout the process. I would like to recognize a small subset of them. I have grouped them according to the way in which they have helped me.

Firstly, I would like to thank a number of the teachers I had before starting graduate school, who layed the foundations for my development as a mathematician, most notably Sue Allen, Frank Forte, Carol Skidmore, John Mackey, and Tom Bohman.

Secondly, I would like to thank several fellow graduate students at Rutgers for their mathematical assistance at various points of my studies, most notably Justin Gilmer, Matthew Russell, Pat Devlin, and Francis Seuffert.

Thirdly, I would like to thank my advisor, Michael Saks, who provided a large number of insights without which I would not have been able to complete this work. Additionally, I would like to thank our coauthor C. Seshadhri for his insights on one of the projects in this thesis. I would also like to thank the entire Rutgers mathematics faculty for creating an environment capable of allowing graduate students to succeed.

Finally, I would like to thank many friends and family members for being consistently willing to offer their support whether it was asked for or not. In particular, friends such as Jennifer Chu, Megan Cox, Daniel Naylor, and Mason and Becca Compton have provided assistance in my efforts to overcome various obstacles that I have faced. My sisters Heather, Jennifer, and Stefanie have always been supportive, wishing the best for me. My parents Joe and Elke have done everything in their power to make it as easy as possible for me to succeed in many areas of my life. Without these people, it is likely that my efforts would have fallen short.

Table of Contents

Chapter 1

Introduction

In this thesis, we discuss two problems which both fall under the category of what we'll call *Longest Increasing Point Sequence* (LIPS) problems. In an LIPS problem, we are given a set of points inside a box, and the goal is to find the size of the largest subset of these points which form an increasing sequence. It is intuitive what we mean by an increasing sequence, but one can define it more formally by saying that a sequence of points is an increasing sequence if there does not exist a pair of points P_1, P_2 in the sequence such that P_1 lies above and to the left of P_2.

In both of the problems we consider, we will make use of various "divide and conquer" approaches. To illustrate a main idea, if we have a box $\mathcal{B}$ and we choose two subboxes $\mathcal{B}_1, \mathcal{B}_2$ of $\mathcal{B}$ such that $\mathcal{B}_1$ lies entirely below and to the left of $\mathcal{B}_2$, then if S_1 is an increasing point sequence in $\mathcal{B}_1$ and S_2 is an increasing point sequence in $\mathcal{B}_2$, the concatenation of S_1 and S_2 is an increasing point sequence in $\mathcal{B}$. With this idea, our approach in both problems is to break the input box up into a sequence of subboxes where each subbox lies entirely below and to the left of its successor, and attempt to find an increasing point sequence in each subbox. However, our goal is to ensure not only that the sequence we find is increasing, but that its length is close to that of the longest increasing point sequence of the input box. Satisfying this constraint is one of the major difficulties that we need to overcome in both problems, and is the source of the bulk of the technical subtleties that we introduce.

The other major constraint that we need to worry about is efficiency. In both problems, a trivial approach can yield an exact result, but the point of our approach is to show that significant efficiency gains can be obtained if we are willing to tolerate a small amount of error in the output. This gives rise to two potential frontiers for improvement over previous results: (1) improving the running time without significant losses in the error of the output, and (2) decreasing the error of the output without significant blowup of the running time. We will discuss the gains we make in both of these respects in the following sections of the introduction, where we consider each of the two problems we tackle.

The two problems we address are the *Distance to Monotonicity* problem, which derives from the *Longest Increasing Subsequence* (LIS) problem, and the *Ulam Distance* problem, which derives from the *Longest Common Subsequence* (LCS) problem. Both the LIS and LCS problems can be viewed as LIPS problems. We will discuss each of these in further detail in the following sections.

1.1 The Distance to Monotonicity Problem

In the Longest Increasing Subsequence (LIS) problem the input is a function (array) $f : [n] \to [m]$ (where $[n] = \{1, \ldots, n\}$) and the problem is to determine $\mathtt{lis}_f$, the size of the largest $I \subseteq [n]$ such that the restriction of f to I is an increasing function.

The distance to monotonicity of f, $\mathtt{dm}_f$ is defined to be $n - \mathtt{lis}_f$, which is the number of entries of f that must be changed to make f an increasing function. Clearly the algorithmic problems of computing $\mathtt{dm}_f$ and $\mathtt{lis}_f$ are essentially equivalent as are the problems of approximating these quantities within a specified additive error. However, there is no such obvious correspondence between the problems of approximating $\mathtt{dm}_f$ and $\mathtt{lis}_f$ to within a constant *multiplicative* factor. In fact we see from this paper that there is a significant difference in the difficulty of approximating these two problems, as least in some settings.

These problems, both the exact and approximate versions, have attracted attention in several different computational models, such as sampling, streaming, and communication models. Following several recent papers, we study this problem in the streaming model, where we are allowed one sequential pass over the input sequence, and our goal is to minimize the amount of space used by the computation.

1.1.1 Previous Results

The exact computation of $\texttt{lis}_f$ and $\texttt{dm}_f$ can be done in $O(n\log(n))$ time using a clever implementation of dynamic programming [2, 11, 23], which is known to be optimal [20]. In the streaming setting, it is known that exact computation of $\texttt{lis}$ and $\texttt{dm}$ require $\Omega(n)$ space even when randomization is used [13].

The most space efficient multiplicative approximation for $\texttt{lis}_f$ is the deterministic $O(\sqrt{n})$ space algorithm [13] for computing a $(1+\varepsilon)$-multiplicative approximation. This space is essentially optimal [9, 12] for deterministic algorithms. Whether randomization helps significantly for this problem remains a very interesting open question.

In contrast, $\texttt{dm}_f$ has very space efficient approximations algorithms. A randomized multiplicative $(4+\varepsilon)$-approximation using $O(\log^2(n))$ space was found by [13]. This was improved upon by [22] with a $(1+\varepsilon)$-multiplicative approximation using $O(\frac{1}{\varepsilon}\log^2(n))$ space. In the deterministic case, [9] gave a polylogarithmic space algorithm giving a $2+o(1)$ factor approximation, but prior to the present paper the only deterministic algorithm known that gave a $(1+\varepsilon)$-factor approximation for arbitrary $\varepsilon > 0$ was an $O(\sqrt{n})$-space multiplicative approximation given by [13]. There have been no significant previous results with regard to lower bounds for this problem in either the randomized or deterministic case.

1.1.2 Our Contributions

We give the first deterministic streaming algorithm for approximating $\texttt{dm}_f$ to within an $1+\varepsilon$ factor using space polylogarithmic in n and m. More precisely, our algorithm uses space $O(\frac{1}{\varepsilon^2}\log^5(n)\log(m))$.

The improvement in the approximation factor from $2+o(1)$ to $1+\varepsilon$ is qualitatively significant because a factor 2 approximation algorithm to $\texttt{dm}_f$ can't necessarily distinguish between the case that $\texttt{lis}_f = 1$ and $\texttt{lis}_f = n/2$, while a $1+\varepsilon$ approximation can approximate $\texttt{lis}_f$ to within an additive εn term.

Our algorithm works by maintaining a small number of small *sketches* at different scales during the streaming process. The main technical challenge in the analysis is to show that the size of the sketches can be controlled while maintaining the desired approximation quality.

We also establish lower bounds for finding $1+\varepsilon$ multiplicative approximations to `dm`. Using standard communication complexity techniques we establish an $\Omega(\frac{1}{\varepsilon}\log^2(n))$ space lower bound for deterministic algorithms and an $\Omega(\frac{1}{\varepsilon}\frac{\log^2(n)}{\log\log(n)})$ space lower bound for randomized algorithms. The reduction maps the streaming problem to the one-way communication complexity of the "Greater Than" function.

1.2 The Ulam Distance Problem

Computing the Longest Common Subsequence (LCS) between two strings x and y is a fundamental algorithmic problem with numerous applications. Except for some polylogarithmic improvements, the best known algorithm is still the textbook quadratic time dynamic program. Recent results show that strongly subquadratic algorithms for LCS would violate the Strong Exponential Time Hypothesis [5].

A notable special case of LCS is when the strings x and y have no repeated characters. Equivalently, it is convenient of think of both x and y as permutations of length n. In this case, a simple reduction to the longest increasing subsequence yields an $O(n\log n)$ algorithm. We give the first sublinear time algorithm that gives arbitrarily good approximations to the LCS length.

Theorem 1.2.1 *Assume random access to two strings x and y of length n that have no repeated characters. Fix any $\delta > 0$. There is an algorithm that, with probability $> 2/3$, outputs an additive δn estimate to the LCS length in time $\widetilde{O_\delta}(\sqrt{n})$ time.*

More generally, our techniques give multiplicative approximations to the Ulam distance between x and y. This is the minimum number of insertions or deletions required to transform x to y. Alternately, the Ulam distance is n minus the LCS length [8, 2]. (We note that some later results allow substitution operations as well [7, 4]. These authors were concerned with constant factor approximations, and this distinction was unimportant for them.)

Theorem 1.2.2 *Assume random access to two strings x and y of length n that have no repeated characters. Fix any $\varepsilon > 0$ and let d be n minus the LCS length. There is an algorithm that outputs, with probability $> 2/3$, a $(1+\varepsilon)$-approximation to d in time $\widetilde{O}_{\varepsilon}(n/d + \sqrt{n})$.*

1.2.1 Connections to Previous Work

The previous best result for estimating d, as defined in Theorem 1.2.2, is the result of Andoni-Nguyen [4] (henceforth AN). They get a (large) constant factor approximation with the same running time as Theorem 1.2.2. Moreover, they prove that $\tilde{O}(n/d + \sqrt{n})$ is optimal. These bounds were improvements over previous results [6, 3]. We note that the improvement of the constant to $1+\varepsilon$ is essential for getting Theorem 1.2.1. Roughly speaking, if one can get a c-approximation to the Ulam distance, that implies (at best) an additive $(1 - 1/c)n$ approximation to the LCS.

A related result is that of Saks-Seshadhri (henceforth SS), on sublinear time algorithms to approximate the length of the Longest Increasing Subsequence (LIS) of an array [21]. This problem is (almost) equivalent to approximating the LCS length between an arbitrary input permutation and the identity permutation. In the language of Theorem 1.2.2, they give a $(1+\varepsilon)$-approximation to the Ulam distance d in time $\widetilde{O}_{\varepsilon}(\mathrm{poly}(n/d))$. At a high level, our algorithm is a combination of (suitably modified and improved versions of) AN and SS.

Chapter 2

Preliminaries

As described in Chapter 1, the problems we consider can be viewed as Longest Increasing Point Sequence (LIPS) problems. As a result, we can build a common geometric framework with which we can discuss both problems. We introduce the definitions necessary to build this framework in section 2.1 before introducing terminology specific to each problem in sections 2.3 and 2.4.

2.1 Geometric Framework

2.1.1 Input box, Input points, X-indices and Y-indices

Throughout this thesis, $\mathcal{U}$ denotes the full input box; we will use other letters ($\mathcal{B}$, $\mathcal{T}$, etc.) to denote other boxes. In both the LIS and LCS problems, the input box will be determined by the input sequences; we will discuss this in more detail in sections 2.3 and 2.4. We refer to one of the points given as input to the LIPS problem as an *input point*. For an input point $P \in \mathcal{U}$, we write $x(P)$ and $y(P)$, respectively, for its x and y coordinates. We refer to the horizontal indices of the box $\mathcal{U}$ as X-indices, and the vertical indices of $\mathcal{U}$ as Y-indices. Note that for any point P, $x(P)$ is an X-index and $y(P)$ is a Y-index, but it is not necessarily the case that for every X-index x (resp Y-index y), there is a point P with $x(P) = x$ (resp $y(P) = y$).

2.1.2 Intervals, boxes, height and width

A subinterval of X-indices is an *X-interval* and a subinterval of Y-indices is a *Y-interval*. The product of an X-interval I_X and a Y-interval I_Y is a *box* $\mathcal{B} = I_X \times I_Y$; in particular, we can think of $\mathcal{U}$ as the box given by $I_X \times I_Y$, where I_X and I_Y denote the sets of all X-indices and Y-indices, respectively. For a box $\mathcal{B}$, $X(\mathcal{B})$ and $Y(\mathcal{B})$ denote the X-interval and Y-interval such that $\mathcal{B} = X(\mathcal{B}) \times Y(\mathcal{B})$. $\mathcal{P}(\mathcal{B})$ denotes the set of input points lying in $\mathcal{B}$. A box $\mathcal{B}$ has *width* $w(B) = |X(\mathcal{B})|$ and *height* $h(B) = |Y(\mathcal{B})|$. For simplicity, we also use $d_{min}(\mathcal{B}) = \min(w(\mathcal{B}), h(\mathcal{B}))$ and $d_{max}(\mathcal{B}) = \max(w(\mathcal{B}), h(\mathcal{B}))$.

In the future we will want to refer to endpoints of intervals and corner points of boxes. We borrow the following notation from [21].

- For an X-interval I_X, we will want to refer to the left and right ends of I_X. In particular, if $I_X = (a, b]$, $x_L(I_X) = a$ and $x_R(I_x) = b$.
- For a Y-interval I_Y, we will want to refer to the bottom and top ends of I_Y. In particular, if $I_Y = [c, d]$, $y_B(I_Y) = c$ and $y_T(I_Y) = d$.
- For a box $\mathcal{B}$, we denote the bottom left and top right corners of $\mathcal{B}$ by $P_{BL}(\mathcal{B})$ and $P_{TR}(\mathcal{B})$, respectively.
- For simplicity, we write $x_L(\mathcal{B})$ for $x_L(X(\mathcal{B}))$, $x_R(\mathcal{B})$ for $x_R(X(\mathcal{B}))$, $y_B(\mathcal{B})$ for $y_B(y(\mathcal{B}))$ and $y_T(\mathcal{B})$ for $y_T(Y(\mathcal{B}))$.

2.1.3 The relations $P \sim Q$ and $P \nsim Q$

For input points $P, Q \in \mathcal{U}$, $P < Q$ means $x(P) < x(Q)$ and $y(P) < y(Q)$. If either $P < Q$ or $Q < P$, then P is *comparable* to Q, denoted $P \sim Q$. If neither hold, then P is a *violation* with Q, denoted $P \nsim Q$.

2.1.4 Increasing Point Sequences and the function `lips`

A sequence of input points $P_1, P_2, \ldots, P_k$ satisfying $P_1 < \cdots < P_k$ is called an *increasing point sequence*. If all points of an increasing point sequence lie in a box $\mathcal{B}$, this is an increasing point sequence in $\mathcal{B}$. Define $\texttt{lips}(\mathcal{B})$ to be the length of the longest increasing point sequence in $\mathcal{B}$. The goal of the LIPS problem is to output $\texttt{lips}(\mathcal{U})$. The LIPS problem will specialize to both the LIS problem and the LCS problem in their respective settings. We will discuss the details of these specializations in sections 2.3 and 2.4, respectively.

2.1.5 Loss functions

A *loss function* assigns to each box a nonnegative integer that measures the number of indices or input points that do not participate in the longest increasing point sequence. We define four loss functions.

- $d_{in}(\mathcal{B}) = |\mathcal{P}(\mathcal{B})| - \texttt{lips}(\mathcal{B})$. $d_{in}(\mathcal{B})$ measures the number of input points in $\mathcal{B}$ which do not lie on a fixed LIPS of $\mathcal{B}$.

- $\texttt{Xloss}(\mathcal{B}) = w(\mathcal{B}) - \texttt{lips}(\mathcal{B})$. $\texttt{Xloss}(\mathcal{B})$ measures the number of X-indices of $\mathcal{B}$ that are not the x coordinate of a point on a fixed LIPS of $\mathcal{B}$.

- $\texttt{Yloss}(\mathcal{B}) = h(\mathcal{B}) - \texttt{lips}(\mathcal{B})$. $\texttt{Yloss}(\mathcal{B})$ measures the number of Y-indices of $\mathcal{B}$ that are not the y coordinate of a point on a fixed LIPS of $\mathcal{B}$. It turns out that we will not need to use the loss function $\texttt{Yloss}(\mathcal{B})$ directly, but it is convenient to define, and we will use a modified version of it, which will be introduced in section 6.2.1.

- $\texttt{uloss}(\mathcal{B}) = w(\mathcal{B}) + h(\mathcal{B}) - 2 \cdot \texttt{lips}(\mathcal{B})$. Alternatively, $\texttt{uloss}(\mathcal{B}) = \texttt{Xloss}(\mathcal{B}) + \texttt{Yloss}(\mathcal{B})$. $\texttt{uloss}(\mathcal{B})$ measures the number of indices (X or Y) of $\mathcal{B}$ that do not appear as a coordinate of a point on a fixed LIPS of $\mathcal{B}$. In the LCS setting, the observation $\texttt{uloss}(\mathcal{B}) = 2 \cdot \texttt{Xloss}(\mathcal{B}) + h(\mathcal{B}) - w(\mathcal{B})$ will be useful for us.

2.1.6 Box sequences and box chains

A *box sequence* is a list of boxes such that each successive box is entirely to the right of the previous. We use the notation $\vec{\mathcal{B}}$ to denote a box sequence. We write $\mathcal{B} \in \vec{\mathcal{B}}$ to mean that the box $\mathcal{B}$ appears in $\vec{\mathcal{B}}$. A *box chain* $\vec{\mathcal{T}}$ is a box sequence $\vec{\mathcal{T}} = (\mathcal{T}_1, \cdots, \mathcal{T}_k)$ such that (for all i), $P_{TR}(\mathcal{T}_i) < P_{BL}(\mathcal{T}_{i+1})$. We say that a box chain $\vec{\mathcal{T}}$ *spans* $\mathcal{B}$ if $\mathcal{B}$ is the smallest box containing $\vec{\mathcal{T}}$.

2.1.7 Box Characterizations

We define two characterizations of boxes. These characterizations will not be used frequently, so we suggest that the reader not worry about them initially, and instead refer back to them when necessary.

- For $\lambda \in (0, 1)$, a box $\mathcal{B}$ is said to be λ-*proportional* if $d_{min}(\mathcal{B}) \geq \lambda d_{max}(\mathcal{B})$. A box is λ-*disproportional* if it is not λ-proportional. Call it λ-*skinny* if $w(\mathcal{B}) < \lambda h(\mathcal{B})$, or λ-*fat* if $h(\mathcal{B}) > \lambda w(\mathcal{B})$.

- For $\omega \in (0, 1)$, a box $\mathcal{B}$ is said to be ω-*small* if $w(\mathcal{B}) \leq \omega$.

2.2 Natural number intervals, sequences and subsequences

$\mathbb{N}$ denotes the set of nonnegative integers and $\mathbb{N}^+$ is the set of positive integers. Interval notation $[a, b]$, $(a, b]$, etc. is used both for intervals of nonnegative integers and real numbers; the meaning will be clear from context. For a positive integer r, $[r]$ denotes the set $\{1, ...r\}$. A natural number sequence of length r is represented as a function $f : [r] \longrightarrow \mathbb{N}$. The sequence is *nonrepeating* if the function is 1-1. A subsequence of f is specified by a subset $I \subseteq [r]$ and is denoted $f(I)$. Writing I as $\{i_1, \ldots, i_s\}$ of $[r]$, with $i_1 < \cdots < i_s$, we think of $f(I)$ as the subsequence $f(i_1), \ldots, f(i_s)$. When I is an interval $[a, b]$ or $(a, b]$, $f(I)$ is a *consecutive subsequence.*

2.3 LIS Notation

2.3.1 The input sequence and input points

In the LIS setting, f denotes a fixed function from $[n]$ to $[m]$, which we refer to as the *input sequence.* The domain (resp range) elements of f correspond to the X-indices (resp Y-indices) in the geometric framework. In this setting, we refer to domain elements of f as *indices* and range elements of f as *values.*

An input point is a pair (x, y) such that $f(x) = y$. Note that in this setting, each index has exactly one value associated to it.

2.3.2 Increasing subsequences and the function `lis`

An *increasing subsequence* of the input sequence f is a list of indices $x_1 < \cdots < x_k$ such that $f(x_1) < \cdots < f(x_k)$. Define $\texttt{lis}_f$ to be the length of the longest increasing subsequence of f. Viewed in the geometric framework, the input points $(x_i, f(x_i))$ form an increasing point sequence. Conversely, the list of X-indices of any increasing point sequence of $\mathcal{U} = [n] \times [m]$ is an increasing sequence of f. Therefore, $\texttt{lips}(\mathcal{U}) = \texttt{lis}_f$.

For a box $\mathcal{B}$, we can also define $\texttt{lis}(\mathcal{B})$ to be the length of the longest increasing subsequence when we restrict to the set of input points lying in $\mathcal{B}$. Alternatively, $\texttt{lis}(\mathcal{B})$ is the length of the longest list of indices $x_1 < \cdots < x_k$ such that $f(x_1) < \cdots < f(x_k)$, and $\forall i$, $(x_i, f(x_i)) \in \mathcal{B}$. Again, we have $\texttt{lips}(\mathcal{B}) = \texttt{lis}(\mathcal{B})$.

2.4 LCS Notation

2.4.1 The input sequences and input points

In the LCS setting, f_X and f_Y denote nonrepeating sequences of lengths n_X and n_Y, respectively, referred to as the *horizontal sequence* and the *vertical sequence*. These are the input to the longest common subsequence problem. The domain elements of f_X (resp f_Y) correspond to the X-indices (resp Y-indices) in the geometric framework, we will maintain these names in this setting.

An input point is a pair (x, y) such that $f_X(x) = f_Y(y)$. Since the functions f_X and f_Y are nonrepeating, each X-index has at most one Y-index associated to it in this way. In the LCS setting, we will refer to input points as *matches*.

2.4.2 Common subsequences and the function `lcs`

A *common subsequence* of the input sequence f is a list of matches $(x_1, y_1), \dots (x_k, y_k)$ such that $x_1 < \cdots < x_k$ and $y_1 < \cdots < y_k$. Define $\texttt{lcs}_{f_X, f_Y}$ to be the length of the longest common subsequence of f_X, f_Y. As in the LIS setting, when viewed in the geometric framework, the matches (x_i, y_i) form an increasing point sequence. Conversely, any increasing point sequence of $\mathcal{U} = [n_X] \times [n_Y]$ is a common sequence of f_X, f_Y. Therefore, $\texttt{lips}(\mathcal{U}) = \texttt{lcs}_{f_X, f_Y}$.

As in the LIS setting, for a box $\mathcal{B}$, we can define $\texttt{lcs}(\mathcal{B})$ to be the length of the longest common subsequence when we restrict to the set of input points lying in $\mathcal{B}$. Again, we have $\texttt{lips}(\mathcal{B}) = \texttt{lcs}(\mathcal{B})$.

2.5 Tail Bounds

We recall a version of the Chernoff-Hoeffding bound from [17] for binomial random variables:

Theorem 2.5.1 (Chernoff-Hoeffding) *Let* $X \sim Bin(n,p)$*. Then if* $\mu = np$*,* $\delta \in (0,1)$*,*

- $\Pr[X < (1-\delta)\mu] \leq e^{-\delta^2\mu/2}$.
- $\Pr[X > (1+\delta)\mu] \leq e^{-\delta^2\mu/3}$.

We also make use of the following version from [18]:

Theorem 2.5.2 (Chernoff Bound) *Let* $X \sim Bin(n,p)$*. Then for* $\delta > 0$*,*

- *If* $\delta \leq 2e-1$*, then* $\Pr[|X - pn| \geq \delta pn] \leq 2 * e^{-\delta^2 pn/4}$
- *If* $\delta \geq 2e-1$*, then* $\Pr[|X - pn| \geq \delta pn] \leq 2^{-(1+\delta)pn}$.

2.6 Parameter approximation and Gap tests

Fix a real valued parameter P. A real number $\tilde{P}$ is a (τ,δ)-approximation to P if $|\tilde{P} - P| \leq \tau P + \delta$. If $\tilde{P}$ is obtained by a randomized algorithm, then $\tilde{P}$ is a (τ,δ)-approximation with failure at most κ if $\Pr[|\tilde{P} - P| > \tau P + \delta] \leq \kappa$. When $\delta = 0$, we simply call the approximation a τ-approximation to P.

A (τ,δ)-*gap test* for P takes as input two parameters, the *lower threshold* a and the *upper threshold* $b \geq (1+\tau)a$ and outputs SMALL or BIG. The gap test is said to fail if either $P < a - \delta$ and it outputs BIG or $P > b$ and it outputs SMALL. We say that the gap test operates with failure probability at most κ if on any input (a,b) with $b \geq (1+\tau)a$ the probability of failure is at most κ. When $\delta = 0$, we simply call the test a τ-gap test. We refer to the pair (τ,δ) (or the parameter τ when $\delta = 0$) as the *quality* of the gap test. The following proposition is easily proven by a geometric search over parameters to a gap test.

Proposition 2.6.1 *Consider* P *whose range is* $\{0,\ldots,r\}$*. Suppose there exists a* (τ,δ)*-gap test for* P *that runs in time* T *with failure probability at most* κ*. Then, there exists* *a* $(2\tau+\tau^2, \delta)$*-approximation algorithm for* P *that runs in time at most* $T(\log_{1+\tau}(r)+2)$ *with failure at most* $\kappa(\log_{1+\tau}(r)+2)$*.*

Proof Construct an approximation algorithm A as follows. Run the (τ, δ)-gap test for successive values of j starting from 0, where $a_j = r/(1+\tau)^{j+1}$ and $b_j = r/(1+\tau)^j$. The first time the test says BIG output a_j as the estimate. If j reaches the value $t = \log_{1+\tau} r + 1$ and the test returns SMALL with parameters a_t and b_t, output 0 as the estimate.

A consists of at most $(\log_{1+\tau}(r)+2)$ calls to the gap test, so by a union bound, the probability that none of these calls fail is at least $1 - \kappa(\log_{1+\tau}(r)+2)$. Therefore, we will assume that none of the gap test calls fail. Additionally, this shows us that the running time of A is within the stated bound.

First, suppose $P = 0$. In this case, the gap test is guaranteed to return small whenever the value of a_j is greater than δ. As a result, the output of A will either be some value of a_j which is at most δ, or 0 if every gap test call returns SMALL. In either case, the output of A differs from the value of P by at most δ.

Suppose $P > 0$. Since $b_t < 1$, the gap test will return BIG for some value of j. Let k be the smallest value of j such that the gap test returns BIG. The output of A is a_k. By the guarantee of the gap test, the value of P must be at least $a_k - \delta$, which is within an additive δ of the output of A. If $k > 0$, then the gap test returned SMALL with parameters a_{k-1} and b_{k-1}. Therefore, the value of P must be at most b_{k-1}. Since $b_{k-1} = (1+\tau)^2 a_k$, the value of P is at most $(1 + 2\tau + \tau^2)$ times the output of A, so these two quantities differ by at most a $(2\tau+\tau^2)$ multiplicative factor. If instead $k = 0$, then since the value of P is at most $r = b_k$, the same bound holds. $\square$

Chapter 3

Approximation Algorithm for Distance to Monotonicity in the Streaming Model

In this chapter, we provide an algorithm which efficiently approximates distance to monotonicity in the streaming model. In the (one pass) streaming model, the input elements are revealed to the algorithm sequentially, and the algorithm outputs an answer when all input elements have been revealed. One can trivially simulate a dynamic programming algorithm in the streaming setting using linear space by writing down every input element as it appears and then doing all of the computation after all elements have been revealed. As a result, the goal is often to be able to design a streaming algorithm which uses a sublinear amount of space.

In our case, the input sequence is revealed to us one element at a time, and our goal is to approximate the distance to monotonicity of the input sequence using a polylogarithmic amount of space. Our approach is a "divide and conquer" approach, where we keep track of $\texttt{Xloss}(\mathcal{B})$ for various subboxes $\mathcal{B}$ of the input box. The quantity $\texttt{Xloss}(\mathcal{B})$ implicitly represents an increasing sequence in $\mathcal{B}$. Since we know that the concatenation of increasing sequences of each box in a box chain is itself an increasing sequence, we can mirror this concatenation by adding $\texttt{Xloss}(\mathcal{B}_1)$ and $\texttt{Xloss}(\mathcal{B}_2)$ whenever $\mathcal{B}_1$ lies entirely below and to the left of $\mathcal{B}_2$. This idea allows us to piece together the estimates we get for $\texttt{Xloss}(\mathcal{B})$ for various subboxes $\mathcal{B}$ into an estimate for $\texttt{Xloss}(\mathcal{U})$.

Building off this idea, for an X-interval I_1, we will keep track of an estimate of $\texttt{Xloss}(I_1 \times J)$ for various Y-intervals J. If we then do the same for an X-interval I_2 lying entirely to the right of I_1, then for J_1 lying entirely below J_2, $\texttt{Xloss}(I_1 \times J_1) + \texttt{Xloss}(I_2 \times J_2) \geq \texttt{Xloss}((x_L(I_1), x_R(I_2)) \times (y_B(J_1), y_T(J_2)))$. The tightness of this inequality will depend on the distribution of pairs of Y-intervals J_1, J_2 for which we kept track of $\texttt{Xloss}(I_1 \times J_1)$ and $\texttt{Xloss}(I_2 \times J_2)$. As a result, for an X-interval I, we will want to keep track of an estimate of $\texttt{Xloss}(I \times J)$ for each J in an appropriately distributed set of Y-intervals. It will be convenient to store these estimates in a matrix indexed by the left and right endpoints of each Y-interval J. These matrices, together with their row and column index sets form what we will call *DM-sketches*. We will define these sketches more formally in section 3.1. The main technical difficulties we face will be ensuring that the estimates given by these sketches are sufficiently accurate, and ensuring that these sketches don't get too big. To handle these difficulties, we note that since our goal is a multiplicative approximation to distance to monotonicity, we can afford more error when we know that $\texttt{Xloss}(\mathcal{U})$ is large. Our sketches can make use of this fact by keeping track of fewer Y-intervals whose size differs significantly from the current X-interval. It turns out that with this idea, we can keep track of enough information to get an ε-approximation to $\texttt{Xloss}(\mathcal{U})$ using only polylogarithmically many Y-intervals in each sketch. Several technical subtleties do come up in the process of verifying this fact, and these will be laid out in sections 3.3 and 3.4. We now proceed to our formulation of DM-sketches

3.1 DM sketches

For a fixed $I \subseteq [n]$, we will want to be able to talk about $\texttt{Xloss}(I \times J)$ for various Y-intervals J. We define DM_I to be the $(m+1) \times (m+1)$ matrix where $DM_I(l,r) = \texttt{Xloss}(I \times (l,r])$. Observe that if $l \geq r$, then $DM_I(l,r) = |I|$.

Our streaming algorithm will try to approximate $DM_{[n]}(0,m)$ (i.e. the distance to monotonicity of the entire sequence). To do this, it will maintain a small set of small matrices that each provide some approximate information about the matrices DM_I for various choices of I. This motivates the next definitions:

- A *DM-sketch* is a triple (L,R,D) where $L,R \subseteq [m] \cup \{0\}$ and D is a nonnegative matrix with rows indexed by L and columns indexed by R. We sometimes refer to the matrix D as a DM-sketch, leaving L and R implicit.

- A DM-sketch (L,R,D) is *well behaved* if for any $l,l' \in L$ and $r,r' \in R$ with $l \leq l'$

and $r' \leq r$, it holds that $D(l,r) \leq D(l',r')$.

- A DM-sketch is said to be *valid for interval* I if $|I| \geq D(l,r) \geq DM_I(l,r)$ for all $l \in L$ and $r \in R$.

- For $i \in [n]$, the *trivial sketch* for i is the DM-sketch with $L = \{f(i)-1\}, R = \{f(i)\}$, and $D = [0]$. Note that the trivial sketch for i is trivially well behaved and valid for i.

- The size of a DM sketch (L,R,D) is $\max(|L|,|R|)$.

Given a valid DM-sketch (L,R,D) for I, we want to obtain an estimate for the $(m+1) \times (m+1)$ matrix DM_I. Observe that, for any I, $([0,m],[0,m],DM_I)$ is a well behaved and valid DM sketch for I. For $l,r \in [m] \cup \{0\}$ and $l' \in L$ and $r' \in R$ with $l \leq l'$ and $r' \leq r$, we have $DM_I(l,r) \leq DM_I(l',r') \leq D(l',r')$. This motivates the following definitions:

- For $l,r \in [m] \cup \{0\}$, the *L-ceiling* of l, denoted by $\bar{l}^L$ is the smallest element $l' \in L \cup \{m\}$ such that $l \leq l'$. Similarly, the *R-floor* of r, denoted by $\underline{r}_R$ is the largest element $r' \in R \cup \{0\}$ such that $r \geq r'$.

- Given the DM-sketch (L,R,D) for I, the *natural estimator of DM_I induced by D* is the matrix D^* given by:

  $$D^*(l,r) = D(\bar{l}^L, \underline{r}_R)$$

  Observe that $([m] \cup \{0\}, [m] \cup \{0\}, D^*)$ is a DM-sketch.

- (L,R,D) is $(1+\delta)$-*accurate for interval* I if for every $l,r \in [m] \cup \{0\}$, $D^*(l,r) \leq (1+\delta)DM_I(l,r)$.

Proposition 3.1.1 *Let D be a DM-sketch, I an interval, and D^* be the natural estimator of DM_I induced by D. If D is well behaved and valid for I, then so is D^*.*

Proof First, note that the well-behavedness of D^* follows from the fact that if $l \leq l'$, $r' \leq r$, then $\bar{l}^L \leq \bar{l'}^L$ and $\underline{r'}_R \leq \underline{r}_R$. Let $l,r \in [m] \cup \{0\}$. The fact that $D^*(l,r) \leq |I|$

follows from the validity of (L, R, D), so it remains to show $D^*(l, r) \geq DM_I(l, r)$. We know that $\bar{l}^L \geq l$ and $\underline{r}_R \leq r$ by definition. As a result, any $(\bar{l}^L, \underline{r}_R)$-monotone subset of I is an (l, r)-monotone subset of I, so we have $DM_I(l, r) \leq DM_I(\bar{l}^L, \underline{r}_R)$. Since $D^*(l, r) = D(\bar{l}^L, \underline{r}_R) \geq DM_I(\bar{l}^L, \underline{r}_R)$ by the validity of (L, R, D), we are done. ☐

3.2 A Polylogarithmic Space Streaming Algorithm

As mentioned in section 3.1, at each step j, our streaming algorithm will maintain a small number of small sketches for various subintervals of $[1, j]$. Our algorithm involves the repeated use of two main building blocks: an algorithm MERGE and an algorithm SHRINK.

The algorithm MERGE takes as input an interval I of even size split into its two halves I_1 and I_2 and DM-sketches (L_1, R_1, D_1) for I_1 and (L_2, R_2, D_2) for I_2 and outputs a DM-sketch (L, R, D) for I. It does this in the following very simple way:

- $L = L_1 \cup L_2$
- $R = R_1 \cup R_2$
- D is defined, for $l \in L$ and $r \in R$ by:

$$D(l, r) = \min_{l \leq z \leq r} D_1^*(l, z) + D_2^*(z, r),$$

  where D_1^* is the natural estimator for $DM_{I_1}(\cdot, \cdot)$ induced by D_1 and D_2^* is the natural estimator for $DM_{I_2}(\cdot, \cdot)$ induced by D_2.

The algorithm SHRINK takes as input a DM-sketch (L, R, D) and outputs a DM-sketch (L', R', D') where $L' \subseteq L$, $R' \subseteq R$ and D' is the restriction of D to $L' \times R'$. It takes a parameter $\gamma > 0$.

The goal of the algorithm SHRINK is to choose (L', R', D') as small as possible while ensuring that, for any $l, r \in [m] \cup \{0\}$, $D'^*(l, r)$ is not too much bigger than $D^*(l, r)$. To find $L' \subseteq L$ and $R' \subseteq R$, our algorithm greedily omits values from L and R without destroying the property

$$\forall l, r \in [m] \cup \{0\}, D^*(l, r) \leq D'^*(l, r) \leq (1 + \gamma)^2 D^*(l, r).$$

The algorithm SHRINK first determines L' and then determines R'. Let $l_1 < \cdots < l_{|L|}$ be the values in L. We construct a sequence $x_1 \leq \hat{x}_1 \leq x_2 \leq \hat{x}_2 \leq x_3, ..., \hat{x}_{s-1} \leq x_s$ iteratively as follows. Let $x_1 = l_1$. For $k \geq 1$, having defined $x_1, \hat{x}_1, ..., \hat{x}_{k-1}, x_k$, if $x_k = l_{|L|}$, stop. Otherwise, let $\hat{x}_k = l_i$ where i is the largest index less than $|L|$ such that

$$\forall r \in R, D(l_i, r) \leq (1+\gamma)D(x_k, r) \tag{3.1}$$

and let $x_{k+1} = l_{i+1}$. Set $L' = \{x_1, \hat{x}_1, x_2, \hat{x}_2, x_3, ..., \hat{x}_{s-1}, x_s\}$. Now let D'' be the submatrix of D induced by the rows of L', giving us an intermediate sketch (L', R, D''). Starting from D'', we perform an analogous construction for R', defining y_1 to be the largest value of R, and working our way downwards (so y_t will be the smallest value of R). We get $R' = \{y_1, \hat{y}_1, y_2, \hat{y}_2, y_3, ..., \hat{y}_{t-1}, y_t\}$, and let D' be the submatrix of D'' induced by the columns labeled by R'. This yields another DM sketch (L', R', D') for I. The DM sketch (L', R', D') will be the sketch that SHRINK outputs.

Armed with the procedures MERGE and SHRINK, we can now describe our deterministic streaming algorithm DMAPPROX for approximating distance to monotonicity. DMAPPROX requires a parameter $\gamma > 0$. (The choice of γ will be $\ln(1+\varepsilon)/(2\log(n))$ where ε is the desired approximation factor.)

We first describe a version of our algorithm that is not in the streaming model, and then convert it into a streaming algorithm, which will be called DMAPPROX. Assume without loss of generality that $n = 2^d$ for an integer d. Consider the rooted binary tree whose nodes are subintervals of $[n]$ with $[n]$ at the root, and for each interval I of length greater than 1, its left and right children will be the first and second halves of I, respectively. This will yield a full binary tree of depth $\log(n)$, where the i^{th} leaf (read from left to right) is the singleton $\{i\}$.

Our algorithm assigns to every node I a DM sketch for I as follows. To each leaf $\{i\}$ we assign the trivial sketch for i. For a non-leaf I with children I_1 and I_2, we take the DM-sketches (L_1, R_1, D_1) and (L_2, R_2, D_2) for I_1 and I_2 respectively, and apply MERGE followed by SHRINK with parameter γ to these sketches to get a DM-sketch (L', R', D') for I. We assign these DM-sketches inductively until we reach the root, yielding a DM sketch (L, R, D) for $[n]$. The output of the algorithm is $D^*(0, m)$.

We now convert this bottom up procedure into a streaming algorithm. We say that a node (interval) I is *completed* if we have reached the end of I in our stream, and we call a node (interval) *complemented* if its parent's other child is also completed. At any point during the stream, we maintain a DM sketch for every completed uncomplemented node I, creating a trivial DM sketch for each leaf as it is streamed. At step i, we look at the i^{th} value in the stream, and we find the largest interval in the binary tree for which i is the right endpoint of that interval. Call this interval I_k, where k is such that the size of this interval is 2^k. Define a sequence of intervals $I_k, I_{k-1}, ..., I_0$, where I_j is the right child of I_{j+1}. Note that i is the right endpoint of each I_j, so each I_j becomes completed at step i. As a result, our algorithm first creates the trivial sketch for i (Note that $I_0 = \{i\}$) and then performs a (possibly empty) sequence of merges and shrinks as follows. For $0 \leq j < k$, given a DM sketch for I_j, the algorithm applies MERGE to the sketch for I_j and the sketch stored for its sibling, and then applies SHRINK with parameter γ to the output of MERGE to get a DM sketch for I_{j+1} (at which point it forgets the sketches for the children of I_{j+1}). The algorithm repeats this process k times, obtaining a sketch for I_k which it stores, as I_k is not yet complemented at step i. Once we reach the end of the stream, we will have our DM sketch for the root. We will prove:

Theorem 3.2.1 (Main Theorem) *Let* $\varepsilon > 0$ *and consider the algorithm* DMAPPROX *with parameter* $\gamma = \ln(1+\varepsilon)/(2\log(n))$. *On input a sequence* f *of* n *integers,* DMAPPROX *outputs an approximation to the distance to monotonicity that is between* $\mathtt{dm}_f$ *and* $(1+\varepsilon)\mathtt{dm}_f$. *The algorithm uses* $O(\frac{1}{\varepsilon^2}\log^5(n)\log(m))$ *space and runs in* $O(\frac{1}{\varepsilon^3}n\log^6(n))$ *time.*

When accounting for time, we assume that arithmetic operations (additions and comparisons) can be done in unit time.

3.3 Proof of the Main Theorem

In this section we state some basic properties about the procedures MERGE and SHRINK, and use them to prove the main theorem. Some of these properties of MERGE and

SHRINK are proved in this section, and others are proved in the next section.

Lemma 3.3.1 (MERGE) *Suppose* MERGE *is run on input* I,I_1,I_2, D_1,D_2 *as described above and let* (L, R, D) *be the output DM-sketch.*

1. *The size of* D *is at most the sum of the sizes of* D_1 *and* D_2.
2. *If* D_i *is well-behaved for* $i \in \{1, 2\}$ *then so is* D.
3. *If* D_i *is valid for* I_i *for* $i \in \{1, 2\}$ *then* D *is valid for* I.
4. *If* D_i *is* $(1 + \delta)$*-accurate for* $i \in \{1, 2\}$ *then* D *is* $(1 + \delta)$*-accurate.*
5. *The algorithm* MERGE *runs in space* $O(\log(m)|L||R|)$ *and time* $O(|L||R|(|L| + |R|))$.

The proof of this lemma is routine and unsurprising.

Proof We prove each item of the claim sequentially.

First, we need to show that the size of D is at most the sum of the sizes of D_1 and D_2. The size of (L, R, D) is given by

$$\begin{aligned} max(|L|, |R|) &= max(|L_1 \cup L_2|, |R_1 \cup R_2|) \\ &\leq max(|L_1| + |L_2|, |R_1| + |R_2|) \\ &\leq max(|L_1|, |R_1|) + max(|L_2|, |R_2|). \end{aligned}$$

which is the sum of the sizes of (L_1, R_1, D_1) and (L_2, R_2, D_2).

Next, to show that D is well-behaved, we need to show that for $l, l' \in L$ and $r, r' \in R$ with $l \leq l'$ and $r' \leq r$, $D(l, r) \leq D(l', r')$. According to the definition of D, let z be such that $D(l', r') = D_1^*(l', z) + D_2^*(z, r')$. Since D_1 and D_2 are well-behaved, D_1^* and D_2^* are well-behaved by Prop. 3.1.1. This gives:

$$\begin{aligned} D_1^*(l', z) + D_2^*(z, r') &\geq D_1^*(l, z) + D_2^*(z, r) \\ &\geq D(l, r). \end{aligned}$$

where the last inequality follows from the definition of D. This shows that D is well-behaved.

Next, to show that (L, R, D) is valid, we need to show that for $x \in L$, $y \in R$,

(1) $D(x,y) \le |I|$

(2) $D(x,y) \ge DM_I(x,y)$

Let z be such that $D(x,y) = D_1^*(x,z) + D_2^*(z,y)$. By Prop. 3.1.1,

$$|I| = |I_1| + |I_2| \ge D_1^*(x,z) + D_2^*(z,y) = D(x,y)$$

establishing (1).

For (2), let z be such that $D(x,y) = D_1^*(x,z) + D_2^*(z,y)$. We have $D_1^*(x,z) = D_1(\bar{x}^{L_1}, \underline{z}_{R_1})$ and $D_2^*(z,y) = D_2(\bar{z}^{L_2}, \underline{y}_{R_2})$ (Note that $\underline{z}_{R_1} \le z \le \bar{z}^{L_2}$). By the validity of (L_1, R_1, D_1) and (L_2, R_2, D_2),

$$\begin{aligned} D(x,y) &= D_1(\bar{x}^{L_1}, \underline{z}_{R_1}) + D_2(\bar{z}^{L_2}, \underline{y}_{R_2}) \\ &\ge DM_{I_1}(\bar{x}^{L_1}, \underline{z}_{R_1}) + DM_{I_2}(\bar{z}^{L_2}, \underline{y}_{R_2}) \\ &\ge DM_I(x,y) \end{aligned}$$

the last inequality following from the definition of DM. This shows that (L,R,D) is valid.

To prove the $(1+\delta)$-accuracy of D, let $l, r \in I$ and let J be the set of indices of an LIS of $I \times (l, r]$. We need to show that $D^*(l,r) \le (1+\delta)DM_I(l,r)$. Let h be the value associated to the largest index of $J \cap I_1$. We see that $DM_{I_1}(l,h) + DM_{I_2}(h,r) = DM_I(l,r)$, so for the (L,R,D) sketch for I,

$$\begin{aligned} D^*(l,r) &= min_{l \le k \le r}(D_1^*(l,k) + D_2^*(k,r)) \\ &\le D_1^*(l,h) + D_2^*(h,r) \\ &\le (1+\delta)(DM_{I_1}(l,h) + DM_{I_2}(h,r)) \\ &\le (1+\delta)DM_I(l,r). \end{aligned}$$

by the $(1+\delta)$-accuracy of (L_1, R_1, D_1) and (L_2, R_2, D_2). This shows that (L,R,D) is $(1+\delta)$-accurate.

We now analyze the amount of time that MERGE takes. Getting L and R from (L_1, R_1, D_1) and (L_2, R_2, D_2) is trivial, and getting $D(x,y)$ for each pair $(x,y) \in L \times R$ requires taking a minimum over at most $|L| + |R|$ choices of z (values of z outside of

$L \cup R$ will not be helpful). Since the D_1^* and D_2^* values here can be computed in constant time (by looking at appropriate values in D_1 and D_2), each of these $|L| + |R|$ choices takes time $O(1)$. This yields the desired time bound of $O(|L||R|(|L| + |R|))$.

Finally, the amount of space that this algorithm uses is just the amount of space required to store L, R, and D. Since each element uses $\log(m)$ bits, this yields the desired space bound of $O(\log(m)|L||R|)$. This completes the proof of Lemma 3.3.1. $\square$

Lemma 3.3.2 (SHRINK) *On input an a sketch* (L, R, D) *that is valid for* I *and* $(1+\delta)$*-accurate,* SHRINK *with parameter* γ *outputs a sketch* (L', R', D') *that is well behaved and valid for* I *and is* $(1+\gamma)^2(1+\delta)$*-accurate. This algorithm runs in space* $O(\log(m)|L||R|)$ *and time* $O(|L||R|)$*.*

Proof First, we see that SHRINK produces a matrix D' which is a submatrix of D for the same interval I, and as a result, the well behavedness and validity of (L', R', D') follows trivially from the definitions.

Next, we need to show that for $l, r \in [m] \cup \{0\}$, $D'^*(l, r) \leq (1+\gamma)^2(1+\delta)DM_I(l, r)$. We do this by showing that $D''^*(l, r) \leq (1+\gamma)D^*(l, r)$, and $D'^*(l, r) \leq (1+\gamma)D''^*(l, r)$. The two arguments are analogous, so we show the proof for the first case only. If $\bar{l}^{L'} = m$ (with $m \notin L'$), then since the largest value of L is in L' by construction of L', $\bar{l}^L = m$ also, and $D''^*(l, r) = D^*(l, r) \leq (1 + \delta)DM_I(l, r)$ by hypothesis. Otherwise, $\bar{l}^{L'} = x_k$ or $\bar{l}^{L'} = \hat{x}_k$ for some k. If $\bar{l}^{L'} = x_k$, then for $x_k = l_{i+1}$ as in the description of SHRINK, $l > l_i$, so $\bar{l}^L = l_{i+1} = x_k$. This means that again, $D''^*(l, r) = D^*(l, r) \leq (1+\delta)DM_I(l, r)$ by hypothesis.

If instead, $\bar{l}^{L'} = \hat{x}_k$,

$$\begin{aligned} D''^*(l, r) &= D''(\hat{x}_k, \underline{r}_R) \\ &= D(\hat{x}_k, \underline{r}_R) \\ &\leq (1+\gamma)D(x_k, \underline{r}_R) \\ &\leq (1+\gamma)D(\bar{l}^L, \underline{r}_R) \\ &= (1+\gamma)D^*(l, r) \\ &\leq (1+\gamma)(1+\delta)DM_I(l, r) \end{aligned}$$

where the third line follows from the definition of SHRINK, and the fourth line follows from the well behavedness of D, as $x_k \leq l \leq \bar{l}^L$. This shows that (L', R, D'') is $(1+\gamma)(1+\delta)$-accurate. As mentioned earlier, an analogous argument with the shrinking of R shows that (L', R', D') is within a $(1+\gamma)$ factor of (L', R, D''), so (L', R', D') is $(1+\gamma)^2(1+\delta)$-accurate.

To analyze the amount of time this algorithm takes, we see that our algorithm involves constructing the sequence $x_1, \hat{x}_1, x_2, \hat{x}_2, x_3, ..., \hat{x}_{s-1}, x_s$. Recall that the elements of L are enumerated as $l_1 < l_2 < \cdots < l_{|L|}$. To determine, $x_{k+1} = l_{i'}$ from $x_k = l_i$, we need to compute the difference between rows l_j and l_i of D starting with $j = i+1$ and continuing until we reach $j = i'$ for which some entry of the difference vector exceeds $(1+\gamma)$ times the corresponding entry of row l_i (or l_j reaches $l_{|L|}$). When this happens we set $x_{k+1} = l_{i'}$ and $\hat{x}^k = l_{i'-1}$. If $x_{k+1} = l_{|L|}$ we stop otherwise we continue to determine x_{k+2} in the same way. Notice that throughout the algorithm, we consider each row only once as l_j so we compute the difference of at most $|L|$ pairs of rows. Each such difference is computed in $O(|R|)$ arithmetic operations so the overall running time is $O(|L||R|)$.

Looking at the amount of space used, we see that since (L', R', D') is not larger than (L, R, D) and none of the intermediate computations require any significant amount of space, we will need at most the space required to store the (L, R, D) sketch, which will be at most $O(\log(m)|L||R|)$ for the D matrix, as it consists of $|L||R|$ elements, each using at most $\log(m)$ bits. Note that if (L, R, D) has size $O(\log_{1+\gamma}(n))$, then this becomes space $O(\frac{1}{\gamma^2}\log^2(n)\log(m))$. □

We will also crucially need to control the size of the sketch that is output by SHRINK. Without an additional hypothesis on the input sketch (L, R, D) we can't bound the size of the sketch (L', R', D') (better than the trivial bound given by the size of (L, R, D)). To obtain the desired bound we will impose a technical condition called *coherence* on (L, R, D). We defer the definition of coherence until Section 5, but the reader can understand the structure of the argument in this section without knowing this definition. In section 5, we'll prove two Lemmas:

Lemma 3.3.3 *If (L, R, D) is coherent, then the output (L', R', D') of* SHRINK *with parameter γ is coherent and satisfies* $\max(|L'|, |R'|) \leq 2\log_{1+\gamma} n + 3$.

In order to carry out the appropriate induction argument we'll need:

Lemma 3.3.4 *For $i \in [n]$, the trivial sketch for i is coherent. Furthermore in* MERGE *if (L_1, R_1, D_1) and (L_2, R_2, D_2) are both coherent then so is (L, R, D).*

Using these pieces, we can now prove Theorem 3.2.1.

Proof First, we aim to show that DMAPPROX approximates $\mathtt{dm}_f$ to within a $1+\varepsilon$ factor. To do this, it suffices to show that the DM sketch for $[n]$ computed by DMAPPROX is valid and $1+\varepsilon$-accurate. Let $\gamma = \ln(1+\varepsilon)/(2\log(n))$. If we run DMAPPROX on f, we have a binary tree of depth $\log(n)$, with a DM sketch for each node. Using Lemma 3.3.1 and Lemma 3.3.2, for a node I with children I_1 and I_2, if the DM sketches for I_1 and I_2 are $(1+\delta)$-accurate, then the DM sketch for I is $(1+\gamma)^2(1+\delta)$-accurate. Furthermore, it is trivial to see that the trivial sketch for i is 1-accurate. By a simple induction on the depth of the tree, our final DM sketch (L, R, D) for $[n]$ is $(1+\gamma)^{2\log(n)}$-accurate. In addition, since the trivial sketch is valid and MERGE and SHRINK preserve validity, the DM sketch for $[n]$ is valid. We see that $(1+\gamma)^{2\log(n)} \leq (e^{2\gamma})^{\log(n)} = 1+\varepsilon$.

Next, we need to show that, at any point during the stream, the algorithm DMAPPROX uses $O(\frac{1}{\varepsilon^2}\log^5(n)\log(m))$ space. First, we note that the trivial sketch is coherent by Lemma 3.3.4, and since MERGE and SHRINK preserve coherence by Lemma 3.3.4 and Lemma 3.3.3, every sketch computed by DMAPPROX is coherent by induction. Now, it is clear that the trivial sketch has size 1, and by Lemma 3.3.3, for any interval I, the DM sketch for I has size $O(\log_{1+\gamma}(n))$. As a result, the intermediate sketches resulting from applications of MERGE will also have size $O(\log_{1+\gamma}(n))$. Note also that for each of these sketches, the constant out in front is uniformly bounded by a small, fixed constant. As a result, it remains to show that the number of sketches stored by DMAPPROX at any given time is sufficiently small.

According to our algorithm description, we maintain DM sketches only for nodes which are both completed and uncomplemented. Since, for any given level of the tree, the sketches for the nodes of this level are obtained sequentially from left to right, at most one node from any level can be both completed and uncomplemented at any point during the stream. As a result, our algorithm stores $O(\log(n))$ sketches at any point in time. This means that the total amount of space needed to store these sketches is $O(\frac{1}{\gamma^2}\log^3(n)\log(m)) = O(\frac{1}{\varepsilon^2}\log^5(n)\log(m))$. Since none of the intermediate computations require more space than this, the desired result is achieved.

Lastly, we need to show that DMAPPROX runs in time $O(\frac{1}{\varepsilon^3}n\log^6(n))$. We start with n intervals of size 1 and we finish with 1 interval of size n, so our procedure performs $n-1$ applications of MERGE and SHRINK, each of which take $O(\frac{1}{\gamma^3}\log^3(n))$ time. Since our entire procedure consists of constructing our DM-sketches for the leaves (each of which takes $O(1)$ time), performing these applications of MERGE and SHRINK, and outputting a value from our final D matrix, the entire procedure runs in time $O(\frac{1}{\gamma^3}n\log^3(n)) = O(\frac{1}{\varepsilon^3}n\log^6(n))$. □

3.4 Sequence Matrices

In this section, we give the definition of the term *coherence* that appears in Lemma 3.3.3 and Lemma 3.3.4, and we prove the lemmas.

At a high level, the goal of this section is to show that our SHRINK procedure yields a sketch which is sufficiently small. In order to do so, it will be necessary to keep track of not only the lengths of the increasing sequences represented by our D matrices, but also the sequences themselves. We have the following definitions:

- A *sequence matrix* S of an interval I is a matrix with rows and columns indexed by values of f, whose entries are subsets of I which are index sets of increasing subsequences.

- A sequence matrix is said to *represent* a DM sketch (L, R, D) if the rows of S are indexed by L, the columns of S are indexed by R, and for each $l \in L$ and $r \in R$ the entry $S(l, r)$ is an index set of an increasing subsequence of $I \times (l, r]$ of length $|I| - D(l, r)$.

Looking at SHRINK, we see that an element is added to L' each time condition (3.1) in the shrinking procedure is violated. We would like to show that each violation of this condition can be associated to a set of witnesses to the violation (which we call *irrelevant elements* below) of sufficient size. To illustrate the idea, consider $l_1, l_2 \in L$, $r \in R$ such that $D(l_2, r) > (1+\gamma)D(l_1, r)$ (a violation of condition (3.1)). If we set $k = D(l_2, r) - D(l_1, r)$, then if S represents (L, R, D), $S(l_1, r)$ has k more elements than $S(l_2, r)$, so it is clear that $S(l_1, r)$ contains at least k elements which do not appear in $S(l_2, r)$. We need for our argument that none of these elements appear in any entry of S in any row at or above l_2, but unfortunately this is not true for an arbitrary sequence matrix representative of (L, R, D). However, it will be possible for us to guarantee that this condition (which we call *coherence*) is satisfied by the sequence matrices that we consider. This motivates the following definitions.

- Given a sequence matrix S, an index i is said to be *left irrelevant* (henceforth we will refer to this simply as *irrelevant*) to $l \in [m] \cup \{0\}$ if for all $l' \in L, r \in R$ such that $l' \geq l$, $S(l', r)$ does not contain i. Analogously, an index i is said to be *right irrelevant* to $r \in [m] \cup \{0\}$ if for all $r' \in R, l \in L$ such that $r' \leq r$, $S(l, r')$ does not contain i.

- A DM sketch (L, R, D) is said to be *left-coherent* for I if there exists a representative sequence matrix S for this sketch such that for any two values $l_1, l_2 \in L$, $r \in R$, $S(l_1, r)$ contains at least $D(l_2, r) - D(l_1, r)$ indices which are left irrelevant (irrelevant) to l_2. Analogously, a DM sketch is said to be *right-coherent* for I if there exists a representative sequence matrix S for this sketch such that for any two values $r_1, r_2 \in R$, $l \in L$, $S(l, r_2)$ contains at least $D(l, r_1) - D(l, r_2)$ indices which are right irrelevant to r_1. Call (L, R, D) *coherent* if it is both left-coherent and right-coherent.

- For S a sequence matrix which represents a DM sketch (L, R, D), the *sequence estimator induced by* S is the $(m+1) \times (m+1)$ sequence matrix S^* given by:

$$S^*(l, r) = S(\bar{l}^L, \underline{r}_R)$$

For the purposes of our analysis, we will build up these sequence matrices in the same way we build up our distance matrices. For $i \in [n]$, the *trivial sequence matrix* for i is the 1×1 matrix $[\{f(i)\}]$. Note that the trivial sequence matrix for i represents the trivial sketch for i.

Let (L_1, R_1, D_1) and (L_2, R_2, D_2) be valid DM sketches for consecutive intervals I_1 and I_2 respectively, and let (L, R, D) be the output sketch obtained by applying MERGE to these two sketches. Given sequence matrices S_1 and S_2 which represent (L_1, R_1, D_1) and (L_2, R_2, D_2) respectively, we construct a sequence matrix S which represents (L, R, D) as follows. Recall that the matrix D constructed in our algorithm had entries $D(l, r)$, where $D(l, r) = min_{l \leq z \leq r}(D_1^*(l, z) + D_2^*(z, r))$. Let z_0 be the smallest z value achieving this minimum. Now let $S(l, r) = S_1^*(l, z_0) \cup S_2^*(z_0, r)$. It is clear that this union is an index set of an increasing subsequence of $I \times (l, r)$. Furthermore, its size by the representativity of S_1 and S_2 is

$$\begin{aligned}
&(|I_1| - D_1^*(l, z_0)) + (|I_2| - D_2^*(z_0, r)) \\
&= |I| - (D_1^*(l, z_0) + D_2^*(z_0, r)) \\
&= |I| - D(l, r)
\end{aligned}$$

This shows that S is representative of (L, R, D). Call S the *merged sequence matrix* of S_1 and S_2.

We now state and prove a proposition which will help us prove Lemma 3.3.4.

Proposition 3.4.1 *Let I be an interval split into two halves I_1 and I_2, and let S_1, S_2 be sequence matrices which represent I_1, I_2 respectively. Let S be the merged sequence matrix of S_1 and S_2. For $l \in L$ and any index $i \in I_1$, if i is irrelevant to l in S_1, then i is irrelevant to l in S. Similarly, for $r \in R$ and any index $i \in I_2$, if i is right irrelevant to r in S_2, then i is right irrelevant to r in S.*

Proof We prove the first statement, the proof of the second part of the proposition is analogous. Let $l' \in L$ such that $l' \geq l$, and let $r \in R$. We aim to show that $S(l', r)$ does not contain i. We have that

$$\begin{aligned} S(l', r) &= S_1^*(l', z_0) \cup S_2^*(z_0, r) \\ &= S_1(\bar{l}'^{L_1}, \underline{z_0}_{R_1}) \cup S_2(\bar{z_0}^{L_2}, \underline{r}_{R_2}) \end{aligned}$$

Since i is irrelevant to l in S_1, $S_1(\bar{l}'^{L_1}, \underline{z_0}_{R_1})$ does not contain i. Furthermore, $S_2(\bar{z_0}^{L_2}, \underline{r}_{R_2})$ does not contain i, as i lies in I_1, and $S_2(\bar{z_0}^{L_2}, \underline{r}_{R_2}) \subseteq I_2$. This shows that $S(l', r)$ does not contain i, proving the claim. □

Using this tool, we now prove Lemma 3.3.4.

Proof First, it is clear that the trivial sequence matrix for i exhibits the coherence of the trivial sketch for i, as L and R both contain 1 element, making the condition for coherence trivially satisfied.

It remains to show that the resultant sketch (L, R, D) from the algorithm MERGE is coherent, given that the input sketches are coherent. We prove that (L, R, D) is left-coherent, the proof that it is right-coherent is analogous and left to the reader. Let I_1, I_2, I be as defined in Lemma 3.3.1, and let (L_1, R_1, D_1) and (L_2, R_2, D_2) be coherent DM sketches for I_1 and I_2 respectively. Let S_1 and S_2 be the representative sequence matrices for these sketches given by the left-coherent condition, and let S be the merged sequence matrix of S_1 and S_2. Let $l_1 < l_2$ be values in L, and let $r \in R$ (Note that the statement is trivial if $l_1 = l_2$, so we only consider $l_1 \neq l_2$). Our goal will be to find $D(l_2, r) - D(l_1, r)$ elements in $S(l_1, r)$ which are irrelevant to l_2. Let z_0 be the minimum value such that

$$D(l_1, r) = D_1^*(l_1, z_0) + D_2^*(z_0, r)$$

We break the argument into two cases:

Case 1: $z_0 \geq l_2$

In this case, we have

$$\begin{aligned} D(l_2, r) &= \min_{l_2 \leq z \leq r} (D_1^*(l_2, z) + D_2^*(z, r)) \\ &\leq D_1^*(l_2, z_0) + D_2^*(z_0, r) \end{aligned}$$

so defining $k = D_1^*(l_2, z_0) - D_1^*(l_1, z_0)$, we have

$$D(l_2, r) - D(l_1, r) \leq D_1^*(l_2, z_0) - D_1^*(l_1, z_0) = k$$

Since (L_1, R_1, D_1) is left-coherent, $S_1^*(l_1, z_0)$ contains at least k indices which are irrelevant to l_2. These indices are in $S(l_1, r)$ by definition of the merged sequence matrix, and they are irrelevant to l_2 in S by Prop. 3.4.1. As such, we find k indices in $S(l_1, r)$ which are irrelevant to l_2 proving the claim in this case.

Case 2: $z_0 < l_2$

In this case, we have

$$\begin{aligned} D(l_2, r) &= \min_{l_2 \leq z \leq r} (D_1^*(l_2, z) + D_2^*(z, r)) \\ &\leq D_1^*(l_2, l_2) + D_2^*(l_2, r) \end{aligned}$$

$$\begin{aligned} D(l_2, r) - D(l_1, r) \leq D_1^*(l_2, l_2) - D_1^*(l_1, z_0) \\ + D_2^*(l_2, r) - D_2^*(z_0, r) \end{aligned}$$

Let k_1, k_2 be such that

$$\begin{aligned} D_1^*(l_2, l_2) - D_1^*(l_1, z_0) &= k_1 \\ D_2^*(l_2, r) - D_2^*(z_0, r) &= k_2 \end{aligned}$$

By definition of D^*, we have $D_1^*(l_2, l_2) = |I_1| = D_1^*(l_2, z_0)$, so $k_1 = D_1^*(l_2, z_0) - D_1^*(l_1, z_0)$. Again, since (L_1, R_1, D_1) is left-coherent, $S_1^*(l_1, z_0)$ contains at least k_1 indices which are irrelevant to l_2. These indices are in $S(l_1, r)$ by definition of the merged sequence matrix, and they are irrelevant to l_2 in S by Prop. 3.4.1.

Furthermore, since (L_2, R_2, D_2) is left-coherent, $S_2^*(z_0, r)$ contains at least k_2 indices which are irrelevant to l_2. These indices are in $S(l_1, r)$ by definition of the merged sequence matrix, and they are irrelevant to l_2 in S by Prop. 3.4.1. Lastly, note that $S_1^*(l_1, z_0) \subseteq I_1$ and $S_2^*(z_0, r) \subseteq I_2$, so these two sets of ($k_1$ and k_2) indices are disjoint. As such, we find $k_1 + k_2$ indices in $S(l_1, r)$ which are irrelevant to l_2 proving the claim in this case as well.

This exhausts all cases, proving the lemma. $\square$

We now prove Lemma 3.3.3.

Proof First, the reader should note that if (L, R, D) is a coherent sketch (with sequence matrix S) and (L', R', D') is any shrinking of (L, R, D) (i.e. $L' \subseteq L$, $R' \subseteq R$, and D' is the associated submatrix of D induced by L' and R'), then it is clear that (L', R', D') is also coherent, as we can just take S' to be the appropriate submatrix of S. As a result, we have that SHRINK preserves coherence.

Consider the sequence $x_1, x_2, x_3, ..., x_s = l_{|L|}$ (without the $\hat{x}$'s) described in the SHRINK procedure. For each i, let r_i be an element $r \in R$ that maximizes $D(x_{i+1}, r) - D(x_i, r)$ and let $k_i = D(x_{i+1}, r_i) - D(x_i, r_i)$. Let S be a coherent sequence matrix representative of (L, R, D). By the definition of left-coherent, for each i between 1 and $s - 1$ there are k_i elements of $S(x_i, r_i)$ that are irrelevant to x_{i+1} (and thus also irrelevant to $x_{i+2}, \ldots, x_s$). Thus these sets of irrelevant elements are disjoint and so $k_1 + \ldots + k_{s-1} \leq n$.

We now prove by induction on j between 1 and $s-1$ that $k_1 + \ldots + k_j \geq (1+\gamma)^{j-1}$. For the basis, $k_1 \geq 1$, and for the induction step suppose $j > 1$. There are $k_1 + \cdots + k_{j-1}$ indices that are irrelevant to x_j so all entries of row x_j are at least this sum which is at least $(1+\gamma)^{j-2}$ by the inductive hypothesis. Since k_j is at least γ times the smallest entry of row x_j by condition (3.1) in SHRINK, we have $k_1 + \cdots + k_j \geq (1+\gamma)(k_1 + \cdots + k_{j-1}) \geq (1+\gamma)^{j-1}$.

On the other hand, $k_1 + \cdots + k_{s-1} \leq n$ which implies $s \leq \log_{1+\gamma}(n) + 2$ so $|L'| \leq 2\log_{1+\gamma}(n) + 3$. Similarly $|R'|$ is bounded above by the same quantity. □

3.5 Algorithm for unknown input length

In streaming algorithms, the question of what values are known to the algorithm is frequently asked. The reader should note that, in our previous algorithm, m was not needed, however the algorithm did require a priori knowledge of the value of n. A closer look at the algorithm shows that, apart from the value of γ, knowledge of n was not needed (note that the way we progress through the binary tree allows us to build to it as we go, continuing the procedure in the same way regardless of the size of n).

Seeking this, we look at the role of γ in our approximation, and we see that one property it had was that $\prod_{i=1}^{\log(n)}(1+\gamma)^2 \leq 1+\varepsilon$. We replace γ with a quantity that depends on the current level of the binary tree, call it $a(i)$ (Here level is counted from the bottom up, i.e. the leaves are at level 1, the parents of the leaves are at level 2, etc). If n is not known beforehand then in principle it could be arbitrarily large, meaning that if we replace γ with $a(i)$, we require $\prod_{i=1}^{\infty}(1+a(i))^2 \leq 1+\varepsilon$. Taking $a(i) = \frac{c}{i^{1+\beta}}$ for any fixed $\beta > 0$ we can choose $c = c(\beta)$ so that this product is at most $1+\varepsilon$. This will yield the desired accuracy of approximation, so it remains to determine the amount of space that this modified algorithm would require.

This modification will result in DM sketches of size $O(\log_{1+a(i)}(n))$ after i merges. We see that $a(i) \leq \frac{c}{\log^{1+\beta}(n)}$, since we have $\log(n)$ levels in our tree, so this yields DM sketches of size at most $O(\frac{1}{\varepsilon}\log^{2+\beta}(n))$. As a result, our D matrices have at most $O(\frac{1}{\varepsilon^2}\log^{4+2\beta}(n))$ entries, resulting in an algorithm that runs in space $O(\frac{1}{\varepsilon^2}\log^{5+2\beta}(n)\log(m))$, for any $\beta > 0$.

Chapter 4

Lower Bounds for Approximating Distance to Monotonicity in the Streaming Model

In this chapter we use standard communication complexity arguments to prove lower bounds for the space complexity of approximating distance to monotonicity for both randomized and deterministic algorithms. We apply a reduction from an appropriate one-way communication problem, a common technique which has been used frequently to establish streaming lower bounds [24].

4.1 Reduction

Let $A(n, \varepsilon)$ be the problem of approximating the distance to monotonicity of n integers taking on values in $[m]$ (where $m = poly(n)$) to within a factor of $(1+\varepsilon)$. Now consider the one-way communication problem where Alice is given a list of k r-bit integers $x_1, x_2, ..., x_k$, Bob is given an index i between 1 and k, as well as an r-bit integer y, and the goal is to compute $GT(x_i, y)$, where GT is the "greater than" function ($GT(x, y) = 1$ iff $x > y$). Denoting this problem by $B(k, r)$, we show that for appropriate choices of parameters, $B(k, r)$ can be reduced to $A(n, \varepsilon)$.

Theorem 4.1.1 *Let* $k = \lfloor \frac{1}{2} \log_{1+\varepsilon}(\frac{\varepsilon n}{2\lceil 1/\varepsilon \rceil}) - \frac{1}{2} \rfloor$, $r = \lceil \log(n) \rceil$, *and assume there exists a protocol to solve* $A(n, \varepsilon)$ *using* $S(n, m, \varepsilon)$ *bits of space. Then there is a protocol for* $B(k, r)$ *using* $O(S(n, m, \varepsilon))$ *bits.*

In order to prove this theorem, we will need the following proposition.

Proposition 4.1.2 *Let* $\varepsilon > 0$, $n \in \mathbb{N}$, $k = \lfloor \frac{1}{2} \log_{1+\varepsilon}(\frac{\varepsilon n}{2\lceil 1/\varepsilon \rceil}) - \frac{1}{2} \rfloor$. *There exists a sequence of positive integers* $a_1, a_2, ..., a_k$ *satisfying the following properties:*

1. $\forall j < k, a_j \geq \varepsilon \sum_{i=j+1}^{k} a_i$
2. $\sum_{i=1}^{k} a_i \leq \frac{n}{2}$

Proof We construct such a sequence $a_1, a_2, ..., a_k$ as follows. Let $a_k = \lceil \frac{1}{\varepsilon} \rceil$. For $j < k$, set $a_j = \lceil (1+\varepsilon) a_{j+1} \rceil$. To establish property 1, we see inductively

$$\begin{aligned} a_j &\geq (1+\varepsilon) a_{j+1} \\ &= \varepsilon a_{j+1} + a_{j+1} \\ &\geq \varepsilon a_{j+1} + \varepsilon \sum_{i=j+2}^{k} a_i \\ &= \varepsilon \sum_{i=j+1}^{k} a_i \end{aligned}$$

For property 2, we first note trivially that for any real number $x \geq \frac{1}{\varepsilon}$, we have $(1+\varepsilon)x \geq x + 1 \geq \lceil x \rceil$. As a result, for any $j < k$, $a_j = \lceil (1+\varepsilon) a_{j+1} \rceil \leq (1+\varepsilon)^2 a_{j+1}$. This yields the following:

$$\begin{aligned} \sum_{i=1}^{k} a_i &\leq a_1 + \sum_{i=2}^{k} a_i \\ &\leq a_1 + \frac{1}{\varepsilon} a_1 \\ &= \frac{1+\varepsilon}{\varepsilon} a_1 \\ &\leq \frac{1+\varepsilon}{\varepsilon} (1+\varepsilon)^{2k} \lceil 1/\varepsilon \rceil \\ &\leq \frac{n}{2} \end{aligned}$$

□

Using this, we prove Theorem 4.1.1.

Proof Assume that we have a streaming protocol P for $A(n, \varepsilon)$ using $S(n, m, \varepsilon)$ bits, and consider an instance of $B(k, r)$ where Alice receives $x_1, x_2, ..., x_k$ as input, and Bob receives i, y as input. Consider the sequence of integers $T(x_1, x_2, ..., x_k, i, y)$ defined as follows. Let $a_1, a_2, ..., a_k$ be a sequence of integers satisfying Prop. 4.1.2, and for any

j let $g(x_j, l) = n^2(l-1) + nx_j$. $T(x_1, x_2, ..., x_k, i, y)$ will consist of $k+1$ blocks, where for $j \leq k$, the j^{th} block consists of a_j consecutive integers ending at $g(x_j, j)$, and the $(k+1)^{th}$ block will consist of $n - \sum_{j=1}^{k} a_j$ consecutive integers beginning at $g(y, i) + 1$.

Under this construction, if $x_i \leq y$, then the first i blocks along with the last block form an increasing subsequence of length greater than $n/2$, and any increasing subsequence containing any element from blocks $i+1$ through k cannot contain any element from the last block, so it will have length at most $n/2$. As a result, the increasing subsequence of the first i blocks and the last block are a longest increasing subsequence, so the distance to monotonicity of $T(x_1, x_2, ..., x_k, i, y)$ is $\sum_{j=i+1}^{k} a_j$. On the other hand, if $x_i > y$, then the same is true for the first $i-1$ blocks along with the last block, so the distance to monotonicity of $T(x_1, x_2, ..., x_k, i, y)$ is $\sum_{j=i}^{k} a_j$ in this case. By condition 1 of Prop. 4.1.2, these values differ by a factor of at least $(1+\varepsilon)$, so P must be able to separate these two cases. As a result, Alice can construct the first k blocks of $T(x_1, x_2, ..., x_k, i, y)$ using her input and run P on this part of the sequence. She can then communicate the current bits stored by P to Bob, at which point Bob can construct the last block of $T(x_1, x_2, ..., x_k, i, y)$ using his input and run the remainder of P to get its result. At this point, Bob can use the result of P to determine whether or not $x_i > y$, and output the result. This is a protocol for $B(k, r)$ using $O(S(n, m, \varepsilon))$ bits, so $B(k, r)$ requires $O(S(n, m, \varepsilon))$ bits. □

4.2 Results

Using the reduction from the previous section, any deterministic (resp. randomized) lower bound for $B(k, r)$ will translate to a deterministic (resp. randomized) lower bound for $A(n, \varepsilon)$.

Lemma 4.2.1 *Given $B(k, r)$ as defined above,*

1. *Any deterministic protocol for $B(k, r)$ requires $\Omega(kr)$ bits.*

2. *Any randomized protocol for $B(k, r)$ requires $\Omega(\frac{kr}{\log(r)})$ bits.*

Before proving this lemma, we note that it along with Theorem 4.1.1 immediately implies the following two results.

Theorem 4.2.2 *Any deterministic streaming algorithm which approximates the distance to monotonicity of a sequence of* n *nonnegative integers to within a factor of* $1+\varepsilon$ *requires space* $\Omega(\frac{1}{\varepsilon}\log^2(n))$.

Theorem 4.2.3 *Any randomized streaming algorithm which approximates the distance to monotonicity of a sequence of* n *nonnegative integers to within a factor of* $1+\varepsilon$ *requires space* $\Omega(\frac{1}{\varepsilon}\frac{\log^2(n)}{\log\log(n)})$.

We now prove Lemma 4.2.1

Proof Starting with the first claim, it is a well known fact that the deterministic one-way communication complexity of a function $D(x, y)$ is just $\log(w)$, where w is the number of distinct rows in the communication matrix for D. Since any two rows of the matrix for $B(k, r)$ corresponding to distinct k-tuples $(x_1, x_2, ..., x_k)$ are distinct, it remains to count the number of such possible k-tuples. Each x_i can take on any of 2^r values, giving us 2^{kr} such k-tuples. The claim follows.

Before addressing the second claim, we first note that $B(1, r)$ is just the "Greater Than" function, $GT(r)$. It has been shown that a lower bound for the one-way communication complexity of $GT(r)$ is $\Omega(r)$ [16]. It seems plausible that this would translate to a $\Omega(kr)$ lower bound for the one-way communication complexity of $B(k, r)$, however we are unable to adapt this argument. [15] gives a simpler argument achieving a lower bound of $\Omega(\frac{r}{\log(r)})$ for the one-way communication complexity of $GT(r)$, which we are able to adapt to achieve a lower bound of $\Omega(\frac{kr}{\log(r)})$ for $B(k, r)$. Applying this technique, we show that running a randomized protocol for $B(k, r)$ $O(\log(r))$ times will yield a randomized one way protocol capable of computing the indexing function where Alice is given a kr bit string $x_1, x_2, ..., x_{kr}$, Bob is given an index $i \in [kr]$, and the goal is to output x_i, a problem that is known to require $\Omega(kr)$ bits [15].

Let P be a randomized protocol for $B(k, r)$ achieving the optimal complexity. Fix inputs $x_1, x_2, ..., x_k, i, y$ for Alice and Bob. If Alice and Bob run P on this input, they will err with probability at most 1/3. If instead Alice and Bob run P $c\log(r)$ times for some constant c and Bob outputs the majority result, this protocol will err with probability at most $r^{-\Omega(1)}$. Note that the message sent by this protocol does not depend on Bob's input, meaning Bob can compute the output for several different choices of his input without any additional communication (though it will increase the probability of error). This means that for the set $\{y_1, y_2, ..., y_r\}$, Bob can use Alice's message to compute $GT(x_i, y_j)$ for each j. Furthermore, since this set has only r elements, the probability that all of these computations are correct is at least $1 - r^{-\Omega(1)}$. Choosing the y_j's accordingly, Bob can essentially run a binary search to determine x_i exactly with high probability. To see this, we first note that, given a fixed x_i, running a binary search to determine x_i will use a fixed sequence $y_1, y_2, ..., y_r$, assuming each output of $GT(x_i, y_j)$ is correct. Therefore, for any j, if $GT(x_i, y_t)$ gave the correct output for each $t < j$, then Bob's choice of y_j will be determined by x_i (i.e. by the previous values of $GT(x_i, y_t)$). Since the value of x_i uniquely determines the sequence $y_1, y_2, ..., y_r$ that will yield a correct binary search for x_i, we can use a union bound to bound the probability that $GT(x_i, y_j)$ outputs the correct value for all indices j. Since for any fixed j, the probability that the output for $GT(x_i, y_j)$ is incorrect is at most $r^{-\Omega(1)}$, the probability that at least one of these is incorrect is at most $r^{1-\Omega(1)} = r^{-\Omega(1)}$. This shows that the probability that all of these outputs are correct (i.e. the probability that Bob correctly computes x_i) is at least $1 - r^{-\Omega(1)}$.

As a result, this is a randomized protocol P' for the problem where Alice is given $x_1, x_2, ..., x_k$, Bob is given i, and the goal is to compute every bit of x_i. This protocol can be used as a protocol for the indexing problem mentioned earlier as follows. For an instance of this aforementioned problem, Alice is given $x'_1, x'_2, ..., x'_{kr}$, Bob is given an index $i \in [kr]$, and the goal is to output x'_i. Alice can view her input as k strings of length r and run P'. Bob can run P' using $\lfloor \frac{i-1}{n} + 1 \rfloor$, which will give him the value of x'_i (in addition to several other values x'_j) with probability at least 2/3. This shows that c $\log(r)$ iterations of P can be used to simulate a computation known to require $\Omega(kr)$ bits [15]. The result follows. $\square$

Chapter 5

Approximation Algorithm for Ulam Distance

In this chapter, we outline our algorithm for efficiently approximating ulam distance. Here we are working in the random access model where we are allowed query access to the input sequences, and our goal is to minimize the amount of time our algorithm takes in the worst case. To do this, we again employ a "divide and conquer" approach, making use of and building on ideas from [4] and [21]. The framework we start with is similar to the framework of [4]. We first break the input box $\mathcal{U}$ up into a box chain with the property that (with probability at least 2/3) the LCS's of the boxes in the box chain together form an LCS of the input box $\mathcal{U}$. Again following the track of [4], we approximate the ulam distance of each of these boxes by considering further subboxes of them. However, the way in which we do this deviates somewhat from the approach of [4]. Additionally, we modify the terminology of this approach, reformulating these steps in terms of gap tests, as well as a class of procedures which we will define, called *loss indicators.*

After all this, we end up with the problem of approximating $\texttt{Xloss}(\mathcal{T})$ for $\mathcal{T}$ a subbox of $\mathcal{B}$, which is itself a subbox of the input box $\mathcal{U}$. In [4], this task was done by standard inversion counting techniques. Our goal is a more accurate approximation algorithm, for which such techniques will no longer suffice. As a result, we implement the more subtle approach of approximating $\texttt{Xloss}(\mathcal{B})$ outlined in [21]. Unfortunately, the algorithm of [21] is an approximation algorithm for distance to monotonicity, where everything is done in the LIS setting. Since we are working in the LCS setting, we need to find ways to simulate the tasks of this algorithm in the LCS setting. In particular, for an X-index x, the value $f(x)$ can be queried in constant time in the LIS setting, but it requires a linear amount of time to find a y with $f_X(x) = f_Y(y)$ in the LCS setting.

Since we desire a sublinear complexity for our algorithm, it is necessary for us to find a more clever simulation of this task. We are able to do this, but it requires us delving a little deeper into the nature of these samples, and also requires us to make several other modifications of the algorithm of [21].

The remainder of this chapter addresses the task of providing a more detailed overview of our approach. At the end of this chapter, we will outline the layout of the chapters which deal with the details of this algorithm, providing references to the sections which address the tasks employed by our algorithm.

5.1 Overview of the algorithm

Our results use ideas as well as the high level structure of the AN algorithm [4]. This algorithm runs in time $\tilde{O}(\frac{n}{d} + \sqrt{n})$, and with high probability, provides a $(C, 0)$-approximation to the Ulam distance for some large C. We carefully analyze AN to identify the places that contribute significantly to the multiplicative loss. Along the way we abstract and simplify the key ideas of the algorithm.

Prop. 2.6.1 allows us to focus on gap tests. We also need gap tests for the Ulam distance within boxes. The AN algorithm can be viewed as constructing a hierarchy of three gap tests, each with successively better run times.

- A slow gap test whose run time is $\tilde{O}(\frac{w(\mathcal{B})^{3/2}}{b})$.
- A medium gap test whose run time is $\tilde{O}(\frac{w(\mathcal{B})\sqrt{a}}{b})$.
- A fast gap test whose run time is $\tilde{O}(\frac{w(\mathcal{B})}{b} + \sqrt{a})$.

The slow gap test is built using inversion/violation counting techniques, common to property testing algorithms for LIS [10, 19, 1]. The medium gap test is built from the slow gap test and the fast gap test is built from the medium gap test. (Both are done through related, but different techniques.) We refer to the methods that do this as the *speed-up procedures*. The drawback of their algorithm is the multiplicative approximation guarantee (quality) of the gap test. The slow gap test has quality 36, the medium gap test has quality 512, and the fast gap test has quality about 1400.

Our improvement has two components:

- **Higher quality speed-up:** By a careful analysis and modification of the AN framework, we avoid the explosion of the quality factor from the slow test to the medium, and from the medium to the fast. Thus, the fast gap test has quality only slightly worse than the slow gap test.

- **Higher quality slow gap test:** The main bottleneck in the quality comes from the slow test. Their slow test relies on a standard technique from monotonicity testing called *inversion counting* and this method cannot yield approximation factors better than 2. The SS algorithm [21] overcomes this limitation to get a polylogarithmic time algorithm for approximating distance to monotonicity with an additive error δn.

We adapt the SS algorithm to get a slow gap test with quality $(\varepsilon, 0)$. The SS algorithm is an incredibly complex beast. Our main insight is that a gap test can be constructed by a careful modification to SS that avoids getting into the intricacies of SS.

We describe some of the key ideas behind these two components.

5.1.1 Getting the speed-up routines

Our speed-up routines are directly inspired by those in the AN algorithm, but differ in several respects: firstly, we provide a conceptual simplification of AN for a cleaner exposition; secondly, the requirement of a higher quality speed-up introduces new technical ideas.

Switching between loss functions: We begin with the latter. Routine calculations give tighter bounds for many of the AN arguments. But the transformation from the slow gap test to the medium test (at the very best) takes a β-gap test and produces a $(\beta+1)$-gap test. (Note that the latter gives a $(1+\beta, 0)$-approximation.) This suffices for a constant factor approximation to the Ulam distance, but cannot yield an (overall) $(\varepsilon, 0)$-approximation. Avoiding this requires getting into the subtlety of defining losses.

All the AN gap tests (and proofs) deal with $\texttt{Xloss}(\mathcal{B}) = w(\mathcal{B}) - \texttt{lcs}(\mathcal{B})$. Note that this is potentially smaller that the Ulam distance $\texttt{uloss}(\mathcal{B})$, which is $\texttt{Xloss}(\mathcal{B}) + h(\mathcal{B}) - \texttt{lcs}(\mathcal{B}) \geq \texttt{Xloss}(\mathcal{B})$. Thus, the AN gap tests are dealing with a more stringent loss function. Our slow gap test also yields an $\texttt{Xloss}(\cdot)$ bound, but our medium gap test only yields a guarantee for `uloss`$(\mathcal{B})$. Of course, this suffices for the final algorithm, but it is crucial for avoiding the quality loss from a β-gap test to a $(\beta+1)$-gap test.

Loss indicators: One key conceptual simplification we introduce in building the medium and fast gap test is an *indicator for a parameter P*. Suppose we are trying to estimate a nonnegative parameter P, which has an upper bound r. A *(β_0, β_1)-quality P-indicator* is a 0-1 valued random variable Z_P, satisfying $\Pr[Z_P = 1] \in [\frac{P}{r} - \beta_0, \frac{P}{r} + \beta_1]$. In particular Z_P is an estimator for P having bias at most $\max(\beta_0, \beta_1)$. We will consider indicators for `uloss`$(\mathcal{B})$ and `Xloss`$(\mathcal{B})$, which we refer to as *loss indicators*. Here we take the trivial upper bound for `uloss`$(\mathcal{B})$ to be $w(\mathcal{B}) + h(\mathcal{B})$ and for `Xloss`$(\mathcal{B})$ to be $w(\mathcal{B})$.

Given such an indicator, we can easily construct a gap test for P with thresholds a and b, where $b > a + (\beta_0 + \beta_1)r$.

Proposition 5.1.1 *Suppose for a nonnegative parameter P with upper bound r, we have a P-indicator with (β_0, β_1)-quality, then we can construct a $(\tau, (\beta_0+\beta_1)r)$-gap test for P with error at most $\kappa > 0$ which, for any a, b with $a \geq (\beta_0 + \beta_1)r$, $b > (1+\tau)a$, runs in time $O(\frac{(1+\tau)r}{b\tau^3}\log(\frac{1}{\kappa}))$ times the running time of the P-indicator.*

Proof Suppose that the probability that the P-indicator outputs 1 is q. If we take m samples of the estimator, the sum of the outputs is a random variable $Z \sim Bin(m, q)$. By Theorem 2.5.1, for $\delta > 0$, $\Pr[Z < (1-\delta)mq] \leq e^{-\delta^2 mq/2}$. Therefore, if $m \geq \frac{2\log(\frac{1}{\kappa})}{\delta^2 q}$, this is at most κ. Similarly, if $m \geq \frac{3\log(\frac{1}{\kappa})}{\delta^2 q}$, then $\Pr[Z > (1+\delta)mq]$ is at most κ.

Our construction is simply to take m samples of the estimator and declare SMALL iff the fraction of 1's is less than $(1-\frac{\tau}{3})(\frac{b}{r} - \beta_0)$. If the value of P is at least b, then $q \geq \frac{b}{r} - \beta_0$. Therefore, taking $\delta = \frac{\tau}{3}$, if $m \geq \frac{18\log(\frac{1}{\kappa})}{\tau^2(\frac{b}{r}-\beta_0)}$, the probability that we declare SMALL is at most κ.

If on the other hand, the value of P is at most $a-(\beta_0+\beta_1)r$, then $q \leq a - \beta_0 r \leq \frac{b-\beta_0 r}{1+\tau}$. Therefore, taking $\delta = \frac{\tau}{3}$, if $m \geq \frac{27\log(\frac{1}{\kappa})}{\tau^2(\frac{b}{r}-\beta_0)}$, the probability that we declare BIG is at most κ.

Since $b > (1+\tau)a \geq (1+\tau)(\beta_0+\beta_1)r$, $\beta_0 \leq \frac{b}{(1+\tau)r}$. Therefore, $\frac{b}{r} - \beta_0 \geq \frac{\tau}{1+\tau}\frac{b}{r}$, so in either case, taking $m = O(\frac{(1+\tau)r}{b\tau^3}\log(\frac{1}{\kappa}))$ samples of the estimator suffices, establishing the claim. □

The above procedure constructs a $(\tau, (\beta_0 + \beta_1)r)$-gap test for P, however we will often want to obtain a purely multiplicative gap test for P. It turns out that if β_0, β_1 are (approximate) multiples of the parameter P, then we can treat these additive terms as multiplicative terms, meaning that the gap test we have for P can be stated as a purely multiplicative gap test.

Proposition 5.1.2 *Suppose G is a $(\tau, \varepsilon(P+1))$-gap test for an integer parameter P. Then for input thresholds a, b with $a \geq 1$, G is a $(\tau + 2\varepsilon)$-gap test for P.*

Proof Let a, b be input thresholds with $b \geq (1 + \tau + 2\varepsilon)a$, $a \geq 1$. We know that G is a $(\tau, \varepsilon(P+1))$-gap test for P, so if we have input parameters a', b' with $b' \geq (1+\tau)a'$, the test will return BIG if the value of P is at least b' and SMALL if the value of P is at most $a' - \varepsilon(P+1)$. Set $b' = b$, $a' = \frac{b}{1+\tau}$. If the value of P is at least b, the test will return BIG. If the value of P is at most $a \leq a' - 2\varepsilon b$, then if P is at least 1, $a' - 2\varepsilon b \leq a' - \varepsilon(2P) \leq a' - \varepsilon(P+1)$, so the test returns SMALL. If P is 0, then since $a \geq 1 = P + 1$ and $2b \geq a$, $a' - 2\varepsilon b \leq a' - \varepsilon a \leq a' - \varepsilon(P+1)$, so the test again returns SMALL. □

It turns out that we can frame most of AN in this language, reformulating it as algorithms that generate indicators for various parameters and run in very fast expected time. This allows for a cleaner description of AN, and also eases the way to plug in better slow gap tests.

5.1.2 Getting a better slow gap test

Our improved slow gap test is obtained by adapting the fast LIS (longest increasing subsequence) algorithm of SS. In the LIS problem, the input is a single sequence f and the problem is to find the longest monotonically increasing subsequence of f. We can represent the problem geometrically in the plane by the set of points $P(x) = (x, f(x))$ for $x \in [n]$. Just as with the LCS problem, the LIS problem is to find the longest increasing chain of points. Let $\texttt{loss}_f$ denote the number of points not on the LIS.

The SS algorithm obtains an approximation having error $\varepsilon\texttt{loss}_f$ that runs in time $\widetilde{O}((\frac{w(\mathcal{B})}{\texttt{loss}_f})^{O(1)})$. We want to adapt the SS algorithm to get the base gap test. The first step in the adaptation is to improve the running time of the SS algorithm to $\widetilde{O}(\frac{w(\mathcal{B})}{\texttt{loss}_f})$. We sketch that improvement at the end of this section.

Even though LCS and LIS have the same geometric representation, we cannot adapt the LIS algorithm directly to LCS. The reason is that the LIS algorithm has the ability to query the point $P(x)$ associated to a particular X-index x at unit cost. The LCS algorithm can't do this because, given x, the algorithm can read the symbol $f_X(x)$, but doesn't know which Y-index it's matched to in f_Y.

Examining the SS algorithm, one sees that the key primitive that it needs is, given a box of width w and a time bound $\widetilde{O}(t)$, in time $\widetilde{O}(t)$ either generate a random point in the box, or determine that there are at most $\frac{w}{t}$ points in the box. This primitive is implemented in the LIS algorithm by sampling $\widetilde{O}(t)$ random indices and returning the first point in the box or if none, saying that there are at most $\frac{w}{t}$ points. So to adapt this algorithm to LCS, we just need to simulate that primitive.

To simulate this primitive, we select $\widetilde{O}(t\sqrt{w(\mathcal{B})})$ random X-indices from $X(\mathcal{B})$ and $\widetilde{O}(t\sqrt{w(\mathcal{B})})$ random Y-indices from $Y(\mathcal{B})$ and select a random match from among them if there is a match (The full details appear in section 7.3). Provided that $h(\mathcal{B})$ and $w(\mathcal{B})$ are not too far apart, a "birthday paradox" calculation shows that if the number of points is at least $\frac{w(\mathcal{B})}{t}$, then this procedure will almost certainly find a match. Thus, we are able to simulate the primitive at a multiplicative cost of $\widetilde{O}(\sqrt{w(\mathcal{B})})$. This simulation fails on unbalanced boxes, but it turns out that these boxes are unimportant for getting an accurate estimate. Therefore the overall running time of the resulting LCS algorithm is $\widetilde{O}(\frac{w(\mathcal{B})}{\texttt{Xloss}(\mathcal{B})}\sqrt{w(\mathcal{B})})$.

Now we describe our improvement to the LIS algorithm. While the run time bound given by [21] for a multiplicative $(1+\varepsilon)$ approximation to LIS distance is $\widetilde{O_\varepsilon}(\text{poly}(w(\mathcal{B})/\texttt{Xloss}(\mathcal{B})))$, [21] also proves a $\widetilde{O_\delta}(w(\mathcal{B})/\texttt{Xloss}(\mathcal{B}))$ bound for approximating LIS distance to within an additive δn term. We observe that though the full LIS algorithm of [21] is iterative with potentially many levels of recursion, when $\texttt{Xloss}(\mathcal{B})$ is small, most runs of the algorithm end in just one level. At a high level, we can combine a version of SS that runs for only one level of recursion with the additive δn version, to get an overall improved running time. This requires some careful parameter settings, and yields an $(\varepsilon, 0)$-approximation algorithm for $\texttt{Xloss}(\mathcal{B})$ that runs in $\widetilde{O_{\varepsilon}}(w(\mathcal{B})/\texttt{Xloss}(\mathcal{B}))$ time.

5.1.3 The high level structure and main lemmas

The construction consists of eight algorithms:

- **XLI**$_0$ is a point classification procedure such that, when run on a uniformly random X-index of the input box $\mathcal{T}$, is an `Xloss`-indicator for $\mathcal{T}$ running in time $\widetilde{O}_{\varepsilon_3}(\sqrt{w(\mathcal{T})})$.
- **BaseGapTest** is the slow gap test for $\texttt{Xloss}(\mathcal{T})$, running in time $\tilde{O}(\frac{w(\mathcal{T})}{b}\sqrt{w(\mathcal{T})})$, using **XLI**$_0$ as a subroutine.
- **XLI**$_1$ generates an `Xloss`-indicator for a box $\mathcal{T}$ in time $\tilde{O}(\sqrt{w(\mathcal{T})})$, using **BaseGapTest** as a subroutine.
- **XLI**$_2$ generates an `Xloss`-indicator for a box $\mathcal{B}$ in time $\tilde{O}(\sqrt{r})$ by running **XLI**$_1$ on a box of width r.
- **XGapTest** is the medium gap test for $\texttt{uloss}(\mathcal{B})$, using **XLI**$_2$ as a subroutine.
- **ULI**$_1$ generates a `uloss`-indicator for a box $\mathcal{B}$ that runs in time $\tilde{O}(\sqrt{\texttt{uloss}(\mathcal{B})}+1)$, using **XGapTest** as a subroutine.
- **ULI**$_2$ generates a `uloss`-indicator for a box $\mathcal{U}$ that runs in time $\tilde{O}(\frac{b}{\sqrt{n}} + 1)$ by running **ULI**$_1$ on a subbox of width $\tilde{O}(b)$
- **UGapTest** is the fast gap test for $\texttt{uloss}(\mathcal{B})$, using **ULI**$_2$ as a subroutine.

The reader should note that the medium gap test for $\texttt{uloss}(\mathcal{B})$ uses an `Xloss`-indicator rather than a `uloss`-indicator. This is one of the technical subtleties of the algorithm and the proof. We state the eight main lemmas, corresponding to eight steps listed above.

Tab. 5.1 lists each of the 8 procedures along with the running time, quality, failure probability and any constraints on the input that are assumed in establishing the behavior. For each procedure there is a lemma that establishes the guarantees for that

Table 5.1: Procedure Guarantees

	Procedures			
	XLI$_0(x, \mathcal{T})$	**BaseGapTest** $(\mathcal{T}, a, b)$	**XLI**$_1(\mathcal{T})$	**XLI**$_2(\mathcal{B}, a', r)$
Running Time	$\widetilde{O}_{\varepsilon_3}(\sqrt{w(\mathcal{T})})$	$\widetilde{O}_{\varepsilon_3}(\frac{w(\mathcal{T})}{b}\sqrt{w(\mathcal{T})})$	$\widetilde{O}_{\varepsilon_3}(\sqrt{w(\mathcal{T})})$	$\widetilde{O}_{\varepsilon_3}(\sqrt{r})$
Quality	$(0, \varepsilon_4 \frac{\texttt{Xloss}(\mathcal{T})}{w(\mathcal{T})} + (1 + \varepsilon_4 \frac{\texttt{terr}_{\varepsilon_3}(\mathcal{T})}{w(\mathcal{T})})$	$(\varepsilon_3, \texttt{terr}_{\varepsilon_3}(\mathcal{T}))$	$(\frac{2\varepsilon_3}{1+2\varepsilon_3} \frac{\texttt{Xloss}(\mathcal{T})}{w(\mathcal{T})}, \varepsilon_3 \frac{\texttt{Xloss}(\mathcal{T})+1/n}{w(\mathcal{T})} + (1 + \varepsilon_3) \frac{\texttt{terr}_{\varepsilon_3}(\mathcal{T})}{w(\mathcal{T})})$	$(4\varepsilon_3 \frac{\texttt{Xloss}(\mathcal{B})}{w(\mathcal{B})}, 5\varepsilon_3 \frac{\texttt{Xloss}(\mathcal{B})+1}{w(\mathcal{B})} + 2\varepsilon_3 \frac{h(\mathcal{B})-w(\mathcal{B})}{w(\mathcal{B})})$
Failure Probability	1/3	1/3	—	—
Input Constraints	$h(\mathcal{T}) \leq 2w(\mathcal{T})$	$h(\mathcal{T}) \leq 2w(\mathcal{T})$	$h(\mathcal{T}) \leq 2w(\mathcal{T})$	$r \geq \lceil \frac{100a'}{\varepsilon_3^2} \rceil$

procedure, assuming the guarantees for the procedure listed immediately before it in the table.

By adapting the SS algorithm, we obtain the following procedure **XLI**$_0$. **XLI**$_0$ has a multiplicative quality term ε_4, as well as an additive term $\mathtt{terr}_{\varepsilon_3}(\mathcal{T})$, which (as mentioned earlier) is a somewhat technical term whose definition we defer to section 6.2.

Lemma 5.1.3 *Suppose* $\mathcal{T}$ *satisfies the input constraints for* **XLI**$_0(x, \mathcal{T})$ *in Tab. 5.1. Then for* x *chosen uniformly at random from* $X(\mathcal{T})$*,* **XLI**$_0(x, \mathcal{T})$ *is an* `Xloss`*-indicator for* $\mathcal{T}$ *achieving the guarantees listed in Tab. 5.1.*

Lemma 5.1.4 *Assume that for any* $\mathcal{T}$ *satisfying the input constraints for* **XLI**$_0$*, running* **XLI**$_0(x, \mathcal{T})$ *for* x *chosen uniformly at random from* $X(\mathcal{T})$ *is an* `Xloss`*-indicator for* $\mathcal{T}$ *achieving the guarantees listed in Tab. 5.1. Then* **BaseGapTest**$(\mathcal{T}, a, b)$ *is a gap test for* $\mathtt{Xloss}(\mathcal{T})$ *achieving the guarantees listed in Tab. 5.1.*

Lemma 5.1.5 *Assume that for any box* $\mathcal{T}$ *satisfying the input constraints of* **BaseGapTest** *in Tab. 5.1,* **BaseGapTest**$(\mathcal{T}, a, b)$ *is a gap test for* $\mathtt{Xloss}(\mathcal{T})$ *achieving the guarantees listed in Tab. 5.1. Then* **XLI**$_1(\mathcal{T})$ *is an* `Xloss`*-indicator for* $\mathcal{T}$ *achieving the guarantees listed in Tab. 5.1.*

Lemma 5.1.6 *Suppose that* $\mathcal{B}, a', r$ *satisfy the input constraints for* **XLI**$_2(\mathcal{B}, a', r)$ *in Tab. 5.1. Assume that for any box* $\mathcal{T}$ *satisfying the input constraints of* **XLI**$_1$ *in Tab. 5.1,* **XLI**$_1(\mathcal{T})$ *is an* `Xloss`*-indicator achieving the guarantees in Tab. 5.1. Then* **XLI**$_2(\mathcal{B}, a', r)$ *is an* `Xloss`*-indicator achieving the guarantees in Tab. 5.1.*

Lemma 5.1.7 *Assume that for* $\mathcal{B}, a', r$ *satisfying the input constraints for* **XLI**$_2$ *in Tab. 5.1,* **XLI**$_2(\mathcal{B}, a', r)$ *is an* `Xloss`*-indicator achieving the guarantees in Tab. 5.1. Then* **XGapTest**$(\mathcal{B}, a, b)$ *is a gap test for* $\mathtt{uloss}(\mathcal{B})$ *achieving the guarantees in Tab. 5.1.*

Lemma 5.1.8 *Suppose that for a box* $\mathcal{B}$*,* **XGapTest**$(\mathcal{B}, a, b)$ *is a gap test for* $\mathtt{uloss}(\mathcal{B})$ *achieving the guarantees Tab. 5.1. Then* **ULI**$_1(\mathcal{B})$ *is a* `uloss`*-indicator achieving the guarantees Tab. 5.1.*

Lemma 5.1.9 *Suppose that* $\mathcal{U}, a, b, \mathcal{B}_0, \ldots, \mathcal{B}_k$ *satisfy the input constraints for* **ULI**$_2$ *in Tab. 5.1. Assume that for any box* $\mathcal{B}$*,* **ULI**$_1(\mathcal{B})$ *is a* `uloss`*-indicator achieving the*

Table 5.1: Continued

	Procedures			
	XGapTest $(\mathcal{B}, a, b)$	**ULI**$_1(\mathcal{B})$	**ULI**$_2$ $(\mathcal{U}, a, b, \mathcal{B}_0, \ldots, \mathcal{B}_k)$	**UGapTest** $(\mathcal{U}, a, b)$
Running Time	$\widetilde{O}_{\varepsilon_2}(\frac{w(\mathcal{B})}{b}\sqrt{a})$	$\widetilde{O}_{\varepsilon_1}(\sqrt{\texttt{uloss}(\mathcal{B})}+1)$	$\widetilde{O}_{\varepsilon_1}(\sqrt{\frac{\texttt{buloss}(\mathcal{U})}{n}}+1)$	$\widetilde{O}_{\varepsilon}(\frac{n}{b}+\sqrt{n})$
Quality	ε_2	$(\frac{\varepsilon_1}{1+\varepsilon_1}\frac{\texttt{uloss}(\mathcal{B})}{w(\mathcal{B})+h(\mathcal{B})}, \varepsilon_1\frac{\texttt{uloss}(\mathcal{B})+1/n}{w(\mathcal{B})+h(\mathcal{B})})$	$(\frac{\varepsilon_1}{1+\varepsilon_1}\frac{\texttt{uloss}(\mathcal{U})}{w(\mathcal{U})+h(\mathcal{U})}, \varepsilon_1\frac{\texttt{uloss}(\mathcal{U})+1}{w(\mathcal{U})+h(\mathcal{U})})$	ε
Failure Probability	1/3	—	—	2/5
Input Constraints	—	—	$\mathcal{U}$ is an $n \times n$ box, $\mathcal{B}_0, \ldots, \mathcal{B}_k$ is a box chain spanning $\mathcal{U}$ with $\sum_{i=0}^{k} \texttt{uloss}(\mathcal{B}_i) = \texttt{uloss}(\mathcal{U})$ and $\forall i \in [k] \cup \{0\}$, $w(\mathcal{B}_i), h(\mathcal{B}_i) = \widetilde{O}(b)$	$\texttt{uloss}(\mathcal{U}) \leq (1+\varepsilon)b$

guarantees in Tab. 5.1. Then $\mathbf{ULI}_2(\mathcal{U}, a, b, \mathcal{B}_0, \ldots, \mathcal{B}_k)$ *is a* `uloss`*-indicator achieving the guarantees in Tab. 5.1.*

Lemma 5.1.10 *Suppose that* $\mathcal{U}, a, b$ *satisfy the input constraints for* UGap *in Tab. 5.1. Assume that, for* $\mathcal{U}, a, b, \mathcal{B}_0, \ldots, \mathcal{B}_k$ *satisfying the input constraints for* $\mathbf{ULI}_2$ *in Tab. 5.1,* $\mathbf{ULI}_2(\mathcal{U}, a, b, \mathcal{B}_0, \ldots, \mathcal{B}_k)$ *is a* `uloss`*-indicator achieving the guarantees in Tab. 5.1. Then* $\mathbf{UGapTest}(\mathcal{U}, a, b)$ *is a gap test for* $\texttt{uloss}(\mathcal{U})$ *achieving the guarantees in Tab. 5.1.*

5.2 Constructing loss indicators from gap tests

We sketch the construction and proof for the loss indicator procedures $\mathbf{XLI}_1$ and $\mathbf{ULI}_1$ (Lemma 5.1.5 and Lemma 5.1.8). The full proofs are in section 6.2. In both procedures, the loss indicator is built from a gap test. The procedures $\mathbf{XLI}_2$ and $\mathbf{ULI}_2$ are loss indicators constructed from loss indicators as opposed to gap tests; we will discuss them in the next section along with the procedures **XGapTest** and **UGapTest**. $\mathbf{XLI}_2$ and **XGapTest** together form a process which constructs a gap test from the loss indicator given by $\mathbf{XLI}_1$; $\mathbf{XLI}_2$ both improves the running time of the indicator (so that **XGapTest** will be more efficient than **BaseGapTest**) and alters the quality of the indicator in a way that makes it compatible with the analysis of **XGapTest**. $\mathbf{ULI}_2$ and **UGapTest** have a similar relationship. For these reasons, we choose to group the exposition of these four procedures together in the next section as opposed to grouping any of them with $\mathbf{XLI}_1$ and $\mathbf{ULI}_1$ in this section.

The constructions of $\mathbf{XLI}_1$ and $\mathbf{ULI}_1$ are elementary and generic; they are applicable to general parameter indicators and not just loss indicators. For convenience, we will assume that all randomized procedures make no error. (In general, we can ensure this with high probability and take union bounds.)

We begin with the setting of Lemma 5.1.5, but simplify parameters for this discussion. Fix a box $\mathcal{B}$. Let $\tau > 0$ be an arbitrarily small, but fixed constant. We have a procedure **BaseGapTest** for $\texttt{Xloss}(\mathcal{B})$ that runs in time $\widetilde{O}((w(\mathcal{B})/b)\sqrt{w(\mathcal{B})})$. This means that given arbitrary $a, \delta > 0$ and $b > (1+\tau)a$, **BaseGapTest** correctly determines if $\texttt{Xloss}(\mathcal{B}) \geq b$ (BIG) or $\texttt{Xloss}(\mathcal{B}) \leq a - \delta$ (SMALL). More abstractly, we have a (τ, δ)-gap test for the parameter $P = \texttt{Xloss}(\mathcal{B})$, with a maximum value $w(\mathcal{B})$. We wish to construct a loss indicator for P with quality $(\Theta(\tau P/w(\mathcal{B})), \Theta((\tau P + \delta)/w(\mathcal{B})))$.

A direct approach would be to simply estimate P using the gap test, and finally flip an appropriate biased coin. Define $c_i = w(\mathcal{B})/(1+\tau)^i$. For increasing values of i, we can run the gap test with $a = c_i$ and $b = c_{i-1}$, and stop when the test reports BIG. As shown by Prop. 2.6.1, the corresponding value of a is a $(2\tau + \tau^2, \delta)$-approximation to P. The run time of a gap test is inversely proportional to b, and thus the very last test dominates the run time of the indicator. The last value of b is $\Theta(P) = \Theta(\texttt{Xloss}(\mathcal{B}))$. Thus, the run time of the indicator is $\widetilde{O}((w(\mathcal{B})/\texttt{Xloss}(\mathcal{B}))\sqrt{w(\mathcal{B})})$.

Inspired by the analysis in [4], we can design a faster indicator whose run time is only $\widetilde{O}(\sqrt{w(\mathcal{B})})$. The procedure $\mathbf{XLI}_1$ is constructed using this indicator.

We use $G_{i,j}$ (for $i > j$) to denote the gap test with $a = c_i$ and $b = c_j$. The simple indicator described above runs gap test $G_{1,0}$, $G_{2,1}$, $G_{3,2}$, $\ldots$ until one of them returns BIG. The indicator described below uses a probabilistic stopping rule to get a better running time.

1. Initialize h to 0.
2. While $c_h > 1$
   (a) Perform $G_{h+1,h}$.
   (b) If gap test returns BIG, output 1.
   (c) Else (gap test returned SMALL):
      i. With probability $1 - 1/(1+\tau)$, output 0.
      ii. (With remaining probability) Increment h.

For simplicity of exposition, let us suppose that $P = c_k = w(\mathcal{B})/(1+\tau)^k$. The probability that $G_{h+1,h}$ is run is exactly $(1+\tau)^{-h}$. The only way the indicator outputs 1 is when $G_{k+1,k}$ is run, which happens with probability $(1+\tau)^{-k} = P/w(\mathcal{B})$. Thus, the indicator has the right expectation. The expected running time is $\widetilde{O}(\sum_{h \leq \log_{1+\tau} w(\mathcal{B})} (1+\tau)^{-h}(w(\mathcal{B})/c_h)\sqrt{w(\mathcal{B})})$. Plugging in $c_h = w(\mathcal{B})/(1+\tau)^h$, the expected running time is $\widetilde{O}(\sqrt{w(\mathcal{B})})$, as desired.

To design $\mathbf{ULI}_1$, we require an indicator that is even faster. Suppose we have a gap test for a parameter P that runs in time $\widetilde{O}((w(\mathcal{B})/b)\sqrt{a})$. Then, we can get an indicator with an expected running time of $\widetilde{O}(\sqrt{P})$, significantly better than $\widetilde{O}(\sqrt{w(\mathcal{B})})$. (This is precisely the statement of Lemma 5.1.8.)

The key is to run tests of the form $G_{i,j}$ where $i > j+1$. Depending on whether it outputs BIG or SMALL, we can modify the i, j to make the gap smaller. The description of the indicator follows.

1. Initialize i to $\lceil \log_{1+\tau} w(\mathcal{B}) \rceil$ (maximum possible index value) and j to 0
2. While indicator is not assigned 0 or 1
   (a) Perform $G_{i,j}$.
   (b) If test returns BIG then:
      i. If $i = j+1$, output 1.
      ii. Else ($i > j+1$): decrement i.
   (c) Else (test returns SMALL):
      i. If $i = j+1$, output 0.
      ii. With prob. $1 - 1/(1+\tau)$, output 0. With remaining prob., increment j.

It is not hard to show that this procedure outputs 1 with the correct probability (The details of the proof appear in section 6.2). The only probabilistic operation is the increment of j (just as in the previous indicator), and the analysis is almost identical.

The running time analysis crucially uses the improved gap test run time of $\widetilde{O}((w(\mathcal{B})/b)\sqrt{a})$. Observe that when $b = \Theta(w(\mathcal{B}))$, the gap test runs in time $\widetilde{O}(\sqrt{a})$. The choice of a in all the gap tests above is never larger than P. The observation together with the run time analysis of the previous indicator yields an expected run time of $\widetilde{O}(\sqrt{P})$.

5.3 Constructing gap tests from loss indicators

Our method for constructing a gap test from a loss indicator involves first constructing an improved loss indicator using the input loss indicator, and then converting this improved loss indicator to a gap test. In each case, we break this process into two procedures implementing these two steps. We now sketch the construction and correctness proof for the construction of the improved loss indicators $\mathbf{XLI}_2$ and $\mathbf{ULI}_2$ and gap test procedures **XGapTest** and **UGapTest**. The full proofs are in section 6.2.

In both procedures, a gap test with thresholds a, b is built from a loss indicator. As mentioned in section 5.1.1, we can naively construct a gap test from a loss indicator by simply running the loss indicator on the input box a sufficient number of times and using the average value of the loss indicator as an estimate for the value of the parameter. However, this process would not yield any speed-up from the previous gap test because the loss indicator run on the input box is too expensive. So instead, we construct a loss indicator which consists of choosing a random subbox of width $O(b)$ (according to a carefully chosen distribution) and applying the initial loss indicator to that subbox. This new loss indicator is much faster and can be used to build a gap test (via Prop. 5.1.1) which runs in the desired time.

In the case of **UGapTest**, the approach is straightforward (Again, the full details of the proofs appear in section 6.2). Part of the AN algorithm is a procedure **PartialAlign**, which constructs a box chain $\mathcal{T}_1, \cdots \mathcal{T}_M$ for a box $\mathcal{B}$ with $M = \tilde{O}(\frac{n}{b})$ boxes each of width $\tilde{O}(b)$ such that if $\texttt{uloss}(\mathcal{B}) < b$, then with high probability the ulam distance of $\mathcal{B}$ is the sum of the ulam distances of the boxes in the box chain. The running time of **PartialAlign** is $\tilde{O}(\frac{n}{b} + \sqrt{n})$, which is within the cost we're aiming at for our gap test. Let I_j be a $\texttt{uloss}$-indicator for $\mathcal{T}_j$. The procedure $\mathbf{ULI_2}$ constructs a $\texttt{uloss}$-indicator for $\mathcal{B}$ by simply selecting one of the boxes in the chain so that $\mathcal{T}_j$ is selected with probability $\frac{w(\mathcal{T}_j)}{w(\mathcal{B})}$. It is easy to check that $\mathbf{ULI_2}$ outputs a $\texttt{uloss}$-indicator for $\mathcal{B}$ whose expected running time is the (weighted) average running time of I_j. The running time of I_j is $\tilde{O}(\sqrt{\texttt{uloss}(\mathcal{T}_j)} + 1)$. By concavity, the weighted average (which is the running time of $\mathbf{ULI_2}$) can be bounded by $\tilde{O}(\sqrt{\frac{b \cdot \texttt{uloss}(\mathcal{B})}{n}} + 1)$. Now applying Prop. 5.1.1, we get a gap test for $\mathcal{B}$ by running this indicator $\tilde{O}(\frac{n}{b})$ times, so the cost of the gap test is $\tilde{O}(\sqrt{\frac{n \cdot \texttt{uloss}(\mathcal{B})}{b}} + \frac{n}{b})$. By hypothesis of Lemma 5.1.10, $b \geq \Omega(\texttt{uloss}(\mathcal{B}))$, so the running time can be bounded by $\tilde{O}(\sqrt{n} + \frac{n}{b})$.

In the case of **XGapTest**, the argument is significantly more subtle. Again, we would like to construct a loss indicator for $\mathcal{B}$ obtained by running a loss indicator on a randomly chosen subbox of much narrower width. Unlike in **UGapTest**, in this case we can't afford the running time of **PartialAlign**. For some width parameter r, consider boxes which are of the form $[i, i+r] \times [i-a, i+r+a]$. Given access to an $\texttt{Xloss}$-indicator for each of these boxes (which comes from $\mathbf{XLI}_1$), the procedure $\mathbf{XLI}_2$ constructs an $\texttt{Xloss}$-indicator by randomly sampling one of these boxes. (For technical reasons, we actually work on an enlarged box obtained from $\mathcal{B}$ by extending $\mathcal{B}$ r units in all directions and adding an increasing sequence of points of length r below and to the left and also above and to the right.) By choosing r to be a sufficiently large multiple of a, we ensure that this indicator can't be much smaller than $\frac{\texttt{Xloss}(\mathcal{B})}{w(\mathcal{B})}$. Unfortunately, when we use this indicator to get a gap test for $\texttt{Xloss}(\mathcal{B})$ using Prop. 5.1.1, the additive $\texttt{terr}_\varepsilon(\mathcal{B})$ term introduces an unacceptable additive error for $\texttt{Xloss}(\mathcal{B})$. Fortunately, when we do the straightforward conversion of the $\texttt{Xloss}(\mathcal{B})$ gap test into a $\texttt{uloss}(\mathcal{B})$ gap test, the additive error gets converted to an (acceptable) small multiplicative error in $\texttt{uloss}(\mathcal{B})$ (Again, the full details of this proof appear in section 6.2).

The running time analyses for $\mathbf{XLI}_2$ and **XGapTest** are trivial. The procedure $\mathbf{XLI}_2$ consists of one call to $\mathbf{XLI}_1$ on a box of width $O_{\varepsilon_3}(a)$. For input $\mathcal{B}, a, b$, the procedure **XGapTest** consists of $\widetilde{O}_{\varepsilon_3}(\frac{w(\mathcal{B})}{b})$ calls to $\mathbf{XLI}_2$. Using the bound on the running time of $\mathbf{XLI}_1$ established previously, we get the desired running time bound for **XGapTest**.

5.4 Designing BaseGapTest

We pick up the discussion from section 5.1.2. We delve a little deeper into the ideas behind designing **BaseGapTest**. As mentioned earlier, we adapt the SS algorithm by implementing the specific kind of box queries. Unfortunately, the queries are approximate and we have to ensure that the SS guarantees are not affected (too much). We remind the reader of the definitions $d_{min}(\mathcal{B}) = \min(w(\mathcal{B}), h(\mathcal{B}))$ and $d_{max}(\mathcal{B}) = \max(w(\mathcal{B}), h(\mathcal{B}))$.

Given a box $\mathcal{B}$ and a budget t, we need to find t random points/matches inside this box, or certify that the box has at most $w(\mathcal{B})/t$ points in it. As mentioned earlier, we can choose $\widetilde{O}(t\sqrt{\mathcal{B}})$ uniform random samples from $x(\mathcal{B})$ and $y(\mathcal{B})$, count the matches among these, and scale up appropriately to estimate the actual number of matches.

A direct birthday paradox argument shows that if $d_{min}(\mathcal{B}) = \Omega(d_{max}(\mathcal{B}))$, then this succeeds with high probability. Thus, when the box is "square", everything works with an additional running time factor of $\sqrt{\mathcal{B}}$.

We need to deal with the case when $d_{min}(\mathcal{B})$ is significantly smaller than $d_{max}(\mathcal{B})$, so the box is "thin". Since n is an upper bound on the dimensions of the box, we can choose to sample $\sqrt{n}$ samples from $x(\mathcal{B})$ and $y(\mathcal{B})$. By going through the birthday paradox calculation in section 7.3, we show that if $d_{min}(\mathcal{B}) = \Omega(\sqrt{n})$, then the procedure succeeds. If $d_{max}(\mathcal{B}) = O(\sqrt{n})$, then both dimensions of the box are small. We can exactly compute the number of matches in $O(\sqrt{n})$ time. (In terms of the input strings, we are provided two substrings of size $O(\sqrt{n})$ and wish to find the number of matches.)

This leaves us with the main problematic case: $d_{min}(\mathcal{B}) = o(\sqrt{n})$, $d_{max}(\mathcal{B}) = \omega(\sqrt{n})$. To deal with these boxes, we observe that the number of matches can be at most $d_{min}(\mathcal{B})$. If $d_{max}(\mathcal{B}) = w(\mathcal{B})$, then we can use this trivial upper bound. If $t \leq w(\mathcal{B})/h(\mathcal{B})$, we simply declare the box has few points in it. When $t \geq w(\mathcal{B})/h(\mathcal{B})$, we explicitly enumerate $y(\mathcal{B})$ and random sample $x(\mathcal{B})$ to discover matches.

Thus, the hard case is when $d_{min} = w(\mathcal{B})$. In an extreme case ($w(\mathcal{B}) = O(1)$, $h(\mathcal{B}) = \Omega(n)$), all x-indices in $\mathcal{B}$ could correspond to matches, yet random sampling cannot find a match. One may wonder why this is a problem, since the x and y axes of the points are completely symmetric. The problem is in the accounting of SS: losses can be measured with respect to $w(\mathcal{B})$ or $h(\mathcal{B})$, but it has to be done consistently for all boxes. Thus, if we commit to performing the SS analysis with respect to `Xloss`, we cannot "switch" the axes.

It requires some detailed understanding of SS to resolve this situation. We can consider square subboxes $\mathcal{B}'$ of $\mathcal{B}$ where $Y(\mathcal{B}')$ starts either at the beginning of $Y(\mathcal{B})$ or ends at the end of $Y(\mathcal{B})$ (as described in section 7.4). We can brute force within these subboxes. If we do not find any matches here, it turns out that we can afford to not find any matches in $\mathcal{B}$. In some sense, we can assert that $\mathcal{B}$ cannot have too many points on the final LCS, so erroneous answers for queries in $\mathcal{B}$ do not affect the final answer.

But all these approximations lead to a further technical difficulty. The final guarantee of **BaseGapTest** has an additive error as well, instead of a purely multiplicative error on the Ulam distance in $\mathcal{B}$. This additive error must now percolate through the entire speed-up process down to the final answer. This is why we define (τ, δ)-gap tests as opposed to (purely multiplicative) τ-gap tests. This requires generalizing all the speed-up analysis of AN to additive errors, introducing much extra notation and complications.

5.5 Road Map

Since this algorithm for ulam distance is very lengthy and technical with many moving parts, it may be helpful to the reader to have an outline of what pieces of the algorithm are covered in which sections and how they all fit together. In section 6.1, we state the "outer" procedures of the algorithm, consisting of the seven procedures in Tab. 5.1 (excluding $\mathbf{XLI}_0$) from section 5.1.3. These procedures are stated in the order in which they are called, and are proved in the same order in section 6.2.

From here, we move on to the discussion of the procedure $\mathbf{XLI}_0$ and the subprocedures it calls. As described in section 5.4, $\mathbf{XLI}_0$ is a modification of the LIS algorithm presented in [21]. Our first task is a modification of that LIS algorithm which improves the running time: we cover this in section 7.1. We then describe the modifications to this algorithm that we apply in order to efficiently simulate it in the LCS setting: this is done in section 7.2. Two of these modifications involve constructing the procedures **BoxSample** and **FindLCS**, which we address in sections 7.3 and 7.4, respectively. We then take a brief detour to provide two motivating examples in section 7.5 before analyzing the correctness and running time of $\mathbf{XLI}_0$ in section 7.6. Lastly, (in section 7.7) we provide the code for a less essential subprocedure mentioned in section 7.2, as well as (in section 7.8) the code for $\mathbf{XLI}_0$ and the rest of its subprocedures (which is similar to the code in [21]) before providing the proof of a straightforward running time claim from section 7.6, which requires the statements of the procedures in section 7.8.

Chapter 6

Reducing the Ulam Distance Problem

6.1 Algorithm

In this section, we state the seven procedures (excluding $\mathbf{XLI}_0$) referred to in section 5.1.3. Again, the procedure **PartialAlign** is restated from [4]. We first introduce the following definition:

Box Extension The a-extension of box $\mathcal{B}$, denoted by $\texttt{ext}_a(\mathcal{B})$ is the box obtained by concatenating two common subsequences of length a to the bottom left and top right corners of $\mathcal{B}$. Similarly, $\texttt{extL}_a(\mathcal{B})$ and $\texttt{extR}_a(\mathcal{B})$ are the boxes obtained by concatenating one common subsequence of length a to the bottom left corner and top right corner of $\mathcal{B}$ respectively. These boxes have the property $\texttt{uloss}(\mathcal{B}) = \texttt{uloss}(\texttt{extL}_a(\mathcal{B})) = \texttt{uloss}(\texttt{extR}_a(\mathcal{B})) = \texttt{uloss}(\texttt{ext}_a(\mathcal{B}))$.

We also introduce the primitive **Sample**(S, p). The goal of **Sample**(S, p) is to simulate choosing a subset of a set S by selecting each element independently with probability p. It is known that this can be done in $\widetilde{O}(p|S|)$ time by choosing a value k subject to the binomial distribution with parameters $|S|$ and p and then selecting a random subset of S of size k. We will make use of this primitive in the procedure **BaseGapTest**.

For **PartialAlign**, $\beta = C_1 \log^3 n$ for some sufficiently large constant $C_1 > 0$, and $\gamma = C_2 \log n$ for some sufficiently large constant $C_2 > 0$.

```
**Main**$(\mathcal{U}, \varepsilon)$
Output: Approximation to uloss$(\mathcal{U})$.
1. Fix global parameters (unchanged throughout algorithm).
```

Symbol	Value
ε_1	$\varepsilon/14$
ε_2	$\varepsilon_1/4$
ε_3	$\varepsilon_2/44$
ε_4	$\varepsilon_3/10$

```
2. For $i \leftarrow 0$ to $\log_{1+\varepsilon} n - 1$
    (a) Run UGapTest$(\mathcal{U}, \frac{n}{(1+\varepsilon)^{i+1}}, \frac{n}{(1+\varepsilon)^i})$ $O(\log n)$ times and take the majority answer.
    (b) If the majority answer is BIG, stop the $i$ loop and return $\frac{n}{(1+\varepsilon)^i}$.
3. If the $i$ loop never stopped (uloss$(\mathcal{U}) \leq 1$), return 0.
```

```
UGapTest$(\mathcal{U}, a, b)$
Output: BIG or SMALL
1. $\mathcal{B}_0, \ldots, \mathcal{B}_k \longleftarrow$ PartialAlign$(X(\mathcal{U}), Y(\mathcal{U}), (1+\varepsilon)b)$.
2. Run ULI$_2(\mathcal{U}, a, b, \mathcal{B}_0, \ldots, \mathcal{B}_k)$ $m = O(k \log^3 n)$ times. Let $z$ be the sum of the outputs.
3. If $z(w(\mathcal{U}) + h(\mathcal{U})) > (1 - \frac{\varepsilon_1}{3})\frac{mb}{(1+\varepsilon_1)}$, return BIG. Otherwise, return SMALL.
```

```
ULI$_2(\mathcal{U}, a, b, \mathcal{B}_0, \ldots, \mathcal{B}_k)$
Output: 0 or 1
1. Choose $i \in [k] \cup \{0\}$ at random with probability $\frac{w(\mathcal{B}_i)+h(\mathcal{B}_i)}{w(\mathcal{U})+h(\mathcal{U})}$.
2. Return ULI$_1(\mathcal{B}_i)$.
```

```
$\textbf{PartialAlign}(A, B, d)$
Output: Boxes $\mathcal{B}_0, \ldots, \mathcal{B}_k$
1. Split $A$ and $B$ into blocks of size $\beta d$. Set $m_0 = 0$, $k = \lceil \frac{n}{\beta d} \rceil$.
2. For $i \leftarrow 1$ to $k$
  (a) For $j \leftarrow 4$ to $\log 4n$
    (1) Pick a random location $p$ in $[(i-1) \cdot \beta d + 4d, i \cdot \beta d - 4d]$ of the $i^{th}$ block.
    (2) Pick $\gamma \cdot 2^{j/2}$ random positions from each of $A[p, p + 2^j]$ and $B[p + m_{i-1} - 2^j, p + m_{i-1} + 2 \cdot 2^j]$.
    (3) If there is at least one collision $A[u] = B[v]$, then do the following.
      (i) Choose any such collision $u_i, v_i$. Set $m_i \leftarrow v_i - u_i, a_i \leftarrow u_i, b_i \leftarrow v_i$.
      (ii) Stop the $j$ loop and jump to the next $i$.
  (b) If the $j$-loop did not stop, then fail.
3. Set $a_0 = b_0 = 0$, $a_{k+1} = |A|$, $b_{k+1} = |B|$.
4. Return the boxes $\mathcal{B}_0, \ldots, \mathcal{B}_k$, where $\mathcal{B}_i = (a_i, a_{i+1}] \times [b_i + 1, b_{i+1}]$.
```

```
$\textbf{ULI}_1(\mathcal{B})$
Output: 0 or 1
1. Set $a = \max(|w(\mathcal{B}) - h(\mathcal{B})|, 1), b = w(\mathcal{B}) + h(\mathcal{B})$.
2. while $\frac{b}{a} > 1 + \varepsilon_2$ do
  (a) Run $\textbf{XGapTest}(\mathcal{B}, a, b)$ $O(\log n)$ times and take the majority answer.
  (b) If the majority answer is BIG, set $a = a(1 + \varepsilon_2)$.
  (c) Else with probability $\frac{\varepsilon_2}{1+\varepsilon_2}$ return 0 and halt, else set $b = \frac{b}{1+\varepsilon_2}$ and continue.
3. If $a > 1$ or $w(\mathcal{B}) \neq h(\mathcal{B})$, return 1.
4. Else return 0.
```

```
$\textbf{XGapTest}(\mathcal{B}, a, b)$
Output: BIG or SMALL
1. Set $a' = \lceil a \rceil$, $r = \lceil \frac{100a'}{\varepsilon_3^2} \rceil$.
2. Run $\textbf{XLI}_2(\mathcal{B}, a', r)$ $k = \widetilde{O}(\frac{w(\mathcal{B})}{b})$ times and let $z$ be the sum of the outputs.
3. If $z > (1 - 4\varepsilon_3) \frac{k}{w(\texttt{extL}_r(\mathcal{B}))} \frac{b + w(\mathcal{B}) - h(\mathcal{B})}{2}$, return BIG. Otherwise, return SMALL.
```

```
$\mathbf{XLI}_2(\mathcal{B}, a', r)$
Output: 0 or 1
1. Set $x = 0$.
2. Let $p_1$ be a uniformly random element of $X(\texttt{extL}_r(\mathcal{B}))$, and let $p_2 = p_1 + r$.
3. Let $\mathcal{T}$ be the subbox of $\texttt{ext}_r(\mathcal{B})$ given by $(p_1, p_2] \times [p_1 - a', p_2 + a']$.
4. Return $\mathbf{XLI}_1(\mathcal{T})$.
```

```
$\mathbf{XLI}_1(\mathcal{T})$
Output: 0 or 1
1. Set $i = 0$
2. While $i \leq \log_{1+\varepsilon_3}(w(\mathcal{T})) + 1$ do
    (a) Run **BaseGapTest**$(\mathcal{T}, \frac{w(\mathcal{T})}{(1+\varepsilon_3)^{i+1}}, \frac{w(\mathcal{T})}{(1+\varepsilon_3)^{i}})$ $O(\log n)$ times and take the majority.
    (b) If the majority answer is BIG, return 1 and halt.
    (c) Else with probability $\frac{\varepsilon_3}{1+\varepsilon_3}$ return 0 and halt. (Otherwise continue)
    (d) Set $i = i + 1$.
3. Return 0.
```

```
BaseGapTest($\mathcal{T}, a, b$)
Output: BIG or SMALL
1. Approximate the number of $X$-indices of $\mathcal{T}$ whose match lies outside $\mathcal{T}$ as follows:
    (a) If $b \leq \frac{16(2e-1)}{\varepsilon_4^2}\sqrt{w(\mathcal{T})}\log n$
        • Let $V$ be the number of characters in $f_X(X(\mathcal{T}))$ not contained in $f_Y(Y(\mathcal{T}))$, obtained by reading the intervals entirely.
    (b) If $b > \frac{16(2e-1)}{\varepsilon_4^2}\sqrt{w(\mathcal{T})}\log n$
        • Set $p = \min(O(\frac{\sqrt{w(\mathcal{T})}\log n}{b}), 1)$.
        • Set $\hat{X} = \mathbf{Sample}(X(\mathcal{T}), p)$, $\hat{Y} = \mathbf{Sample}(Y(\mathcal{T}), p)$
        • Let $C$ be the number of collisions between $\hat{X}$ and $\hat{Y}$.
        • Let $V = w(\mathcal{T}) - \frac{C}{p^2}$.
2. Set $r = \min(O(\frac{\log n}{\sqrt{b}}), 1)$. Run $\mathbf{Sample}(X(\mathcal{T}), r)$ and $\mathbf{Sample}(Y(\mathcal{T}), r)$ and find the collisions.
3. Run $\mathbf{XLI}_0(x, \mathcal{T})$ $O(\log n)$ times on each point $x$ and take the majority outcome. Let $D$ be the number of points classified as bad, and let $W = \frac{D}{r^2}$.
4. If $V + W \geq b(1 - \varepsilon_4)$, return BIG, else return SMALL.
```

6.2 Analysis

In this section, we provide proofs of seven of the eight lemmas stated in section 5.1.3 (Lemma 5.1.3 will be proved in section 7.6). We also state and prove several other lemmas, most of which are tools that help us prove these seven lemmas. In order to do so, we first formally define the function `terr`, as well as several other bits of necessary notation.

6.2.1 The function terr

In section 2.1.5, we introduced two loss functions, `uloss` and `Xloss`. For the analysis, we will need to extend the definition of `Xloss` to sequences, and define a third loss indicator, $\texttt{Yloss}_{\texttt{trim}}$.

As stated in section 2.1.5, $\texttt{Xloss}(\mathcal{B}) = w(\mathcal{B}) - \texttt{lcs}(\mathcal{B})$, which is the number of X-indices of $\mathcal{B}$ that are not matched by the LCS. We extend this function to a sequence

S, where $\texttt{Xloss}(S, \mathcal{B}) = w(\mathcal{B}) - |S|$.

The function $\texttt{Yloss}_{\texttt{trim}}$. Let S be an increasing point sequence in a box $\mathcal{B}$ with $|S| = k$. For P_1, P_k the first and last points of S respectively, $\texttt{Yloss}_{\texttt{trim}}(S, \mathcal{B}) = (y(P_k) - y(P_1)) - (k-1)$. Intuitively, $\texttt{Yloss}_{\texttt{trim}}(S, \mathcal{B})$ represents the minimum number of Y-indices that can be missed by any increasing sequence formed by concatenating S with other increasing sequences on the left or right. Note that if S is an LCS of $\mathcal{B}$, then $\texttt{Xloss}(\mathcal{B}) + \texttt{Yloss}_{\texttt{trim}}(S, \mathcal{B}) \leq \texttt{uloss}(\mathcal{B})$.

With these tools, we now provide the notion of an *error function.* An error function for a sequence S and box $\mathcal{B}$ outputs a nonnegative real number. Error functions are used to measure differences between loss functions of different common subsequences in $\mathcal{B}$. Intuitively, we use error functions to compare the length of the sequences our algorithm finds with the length of the LCS. This is how we keep track of and control the quality of our algorithm.

We will use the following error functions:

- **The function $\texttt{herr}$.** $\texttt{herr}(S, \mathcal{B}) = \texttt{Xloss}(S, \mathcal{B}) - \texttt{Xloss}(\mathcal{B}) = \texttt{lcs}(\mathcal{B}) - |S|$.
- **The function $\texttt{serr}$.** $\texttt{serr}_\varepsilon(S, \mathcal{B}) = \min_{S' \subseteq S}(\texttt{herr}(S', \mathcal{B}) + \varepsilon \cdot \texttt{Yloss}_{\texttt{trim}}(S', \mathcal{B}))$.
- **The function $\texttt{terr}$.** $\texttt{terr}_\varepsilon(\mathcal{B}) = \max_S(\texttt{serr}_\varepsilon(S, \mathcal{B}))$, where the max is taken over common subsequences S of $\mathcal{B}$ with length at least $\texttt{lcs}(\mathcal{B}) - \varepsilon \cdot \texttt{Xloss}(\mathcal{B})$.

It will also be helpful for us to be able to talk about the sequence S' achieving the minimum in $\texttt{serr}_\varepsilon(S, \mathcal{B})$. We'll denote this sequence by $\widetilde{S}_\varepsilon(\mathcal{B})$.

6.2.2 Proofs

We begin with a lemma which establishes the desired properties of **Main**.

Lemma 6.2.1 *Suppose that for $\mathcal{U}, a, b$ satisfying the input constraints of* **UGapTest** *in Tab. 5.1,* **UGapTest**$(\mathcal{U}, a, b)$ *is a gap test for* $\texttt{uloss}(\mathcal{U})$ *achieving the guarantees in Tab. 5.1. Then with probability at least* $2/3$*,* **Main**$(\mathcal{U}, \varepsilon)$ *outputs an* $(\varepsilon, 0)$ *approximation to* $\texttt{uloss}(\mathcal{U})$ *and runs in time* $\widetilde{O}_\varepsilon(\frac{n}{\texttt{uloss}(\mathcal{U})} + \sqrt{n})$.

Proof First, we note that by the Chernoff Bound, since **UGapTest** is correct with probability at least $3/5$. the probability that the majority of $O(\log n)$ runs of **UGapTest** is wrong is bounded by $\frac{1}{n^6}$. Since this task of taking the majority answer of $O(\log n)$ runs of **UGapTest** is done at most $O(\frac{1}{\varepsilon}\log n)$ times, by the union bound all of these are correct with probability at least $1 - \frac{1}{\varepsilon n^5}$. As a result, we may assume that all majority answers are correct.

We continue by showing the correctness of **Main**. First suppose that $\texttt{uloss}(\mathcal{U}) > 1$. Let k be such that $\frac{n}{(1+\varepsilon)^{k+1}} < \texttt{uloss}(\mathcal{U}) \leq \frac{n}{(1+\varepsilon)^k}$. If $\texttt{uloss}(\mathcal{U}) = \frac{n}{(1+\varepsilon)^k}$, **UGapTest** $(\mathcal{U}, \frac{n}{(1+\varepsilon)^{j+1}}, \frac{n}{(1+\varepsilon)^j})$ will return SMALL for $j \leq k-2$ and BIG for $j \geq k+1$. As a result, the output will be between $\frac{n}{(1+\varepsilon)^{k+1}}$ and $\frac{n}{(1+\varepsilon)^{k-1}}$. Therefore, the output is at least $\frac{\texttt{uloss}(\mathcal{U})}{1+\varepsilon}$ and at most $(1+\varepsilon)\texttt{uloss}(\mathcal{U})$, which is an $(\varepsilon, 0)$ approximation to $\texttt{uloss}(\mathcal{U})$. If instead, $\texttt{uloss}(\mathcal{U}) < \frac{n}{(1+\varepsilon)^k}$, **UGapTest**$(\mathcal{U}, \frac{n}{(1+\varepsilon)^{j+1}}, \frac{n}{(1+\varepsilon)^j})$ will return SMALL for $j \leq k-1$ and BIG for $j \geq k+1$. As a result, the output will be between $\frac{n}{(1+\varepsilon)^{k+1}}$ and $\frac{n}{(1+\varepsilon)^k}$. Therefore, the output is again at least $\frac{\texttt{uloss}(\mathcal{U})}{1+\varepsilon}$ and at most $(1+\varepsilon)\texttt{uloss}(\mathcal{U})$, which is an $(\varepsilon, 0)$ approximation to $\texttt{uloss}(\mathcal{U})$.

If instead $\texttt{uloss}(\mathcal{U}) = 1$, then if line 2a returns BIG for some value of i, $\frac{n}{(1+\varepsilon)^{i+1}} \leq 1$, so $\frac{n}{(1+\varepsilon)^i} \leq 1+\varepsilon$, which is at most $(1+\varepsilon)\texttt{uloss}(\mathcal{U})$. Otherwise, the exact value of $\texttt{uloss}(\mathcal{U})$ will be returned by line 3. If $\texttt{uloss}(\mathcal{U}) = 0$, then the test on line 2a will always return SMALL, and the exact value of $\texttt{uloss}(\mathcal{U})$ will again be returned by line 3. Therefore, in each case, the output of **Main** is an $(\varepsilon, 0)$ approximation to $\texttt{uloss}(\mathcal{U})$.

To analyze the running time, we simply see that each iteration of the loop involves running **UGapTest** $O(\log n)$ times, which is in $\widetilde{O}_\varepsilon(\frac{n}{n/(1+\varepsilon)^i} + \sqrt{n})$ time for each i. Since the loop stops when $\frac{n}{(1+\varepsilon)^i} = \widetilde{O}_\varepsilon(\texttt{uloss}(\mathcal{U}))$, the loop runs in time $\widetilde{O}_\varepsilon(\frac{n}{\texttt{uloss}(\mathcal{U})} + \sqrt{n})$ The greedy algorithm on line 3 runs in $\tilde{O}(n)$ time, and since $\texttt{uloss}(\mathcal{U}) \leq 1$ if the procedure reaches this line, it is in $\widetilde{O}(\frac{n}{\texttt{uloss}(\mathcal{U})})$ time. Putting these terms together yields the desired result. □

We restate the following lemma (Lemma 3.1) from [4], which establishes the necessary properties of the procedure **PartialAlign**.

Lemma 6.2.2 *Suppose $\mathcal{U}$ is an $n \times n$ box with $\texttt{uloss}(\mathcal{U}) \leq d$. Let $\mathcal{B}_0, \ldots, \mathcal{B}_k$ be* *the output of* **PartialAlign**$(X(\mathcal{U}), Y(\mathcal{U}), d)$. *Then with probability at least* $2/3$*, the following all hold.*

1. *The boxes* $\mathcal{B}_0, \ldots, \mathcal{B}_k$ *form a box chain spanning* $\mathcal{U}$ *with* $\sum_{i=0}^{k} \texttt{uloss}(\mathcal{B}_i) = \texttt{uloss}(\mathcal{U})$.
2. *For any* i, $w(\mathcal{B}_i), h(\mathcal{B}_i) = O(d \log^3 n)$.
3. *The running time of* **PartialAlign**$(X(\mathcal{U}), Y(\mathcal{U}), d)$ *is* $\tilde{O}(\frac{n}{d} + \sqrt{n})$.

Using Lemma 6.2.2, we prove Lemma 5.1.10.

Proof First, we note that Lemma 6.2.2 ensures that with probability at least $2/3$, $\mathcal{B}_0, \ldots, \mathcal{B}_k$ satisfy the input constraints for $\mathbf{ULI}_2$ in Tab. 5.1. Therefore by assumption, $\mathbf{ULI}_2(\mathcal{U}, a, b, \mathcal{B}_0, \ldots, \mathcal{B}_k)$ is a $(\frac{\varepsilon_1}{1+\varepsilon_1} \frac{\texttt{uloss}(\mathcal{U})}{w(\mathcal{U})+h(\mathcal{U})}, \varepsilon_1 \frac{\texttt{uloss}(\mathcal{U})+1}{w(\mathcal{U})+h(\mathcal{U})})$-quality $\texttt{uloss}(\mathcal{U})$ indicator for $\mathcal{U}$. Applying Prop. 5.1.1, we use this to construct a $(\varepsilon_1, 2\varepsilon_1(\texttt{uloss}(\mathcal{U}) + 1))$-gap test, which by Prop. 5.1.2 is a $5\varepsilon_1$-gap test for $\texttt{uloss}(\mathcal{U})$. If we choose $\kappa = 1/15$ in Prop. 5.1.1, then this gap test has failure probability at most $1/15$. Combining this via a union bound with the probability that **PartialAlign** fails, **UGapTest** succeeds with probability at least $3/5$.

To analyze the running time, we see that **UGapTest**$(\mathcal{U}, a, b)$ consists of first running **PartialAlign** and making $\widetilde{O}_{\varepsilon_1}(\frac{n}{b})$ calls to $\mathbf{ULI}_2$. **PartialAlign** runs in time $\widetilde{O}(\frac{n}{b} + \sqrt{n})$, and by assumption, $\mathbf{ULI}_2$ runs in time $\widetilde{O}_{\varepsilon_1}(\sqrt{\frac{b\texttt{uloss}(\mathcal{U})}{n}} + 1)$. Since $b = \Omega(\texttt{uloss}(\mathcal{U}))$ by assumption, this is at most $\widetilde{O}_{\varepsilon_1}(\frac{b}{\sqrt{n}} + 1)$. Therefore, the total running time of all $\widetilde{O}_{\varepsilon_1}(\frac{n}{b})$ calls to $\mathbf{ULI}_2$ is $\widetilde{O}_{\varepsilon_1}(\frac{n}{b} + \sqrt{n})$. Combining this with the running time of **PartialAlign** yields the running time guarantee for **UGapTest** stated in Tab. 5.1.

We now prove Lemma 5.1.9.

Proof Assume that $\mathcal{U}, a, b, \mathcal{B}_0, \ldots, \mathcal{B}_k$ satisfy the input constraints for $\mathbf{ULI_2}$ in Tab. 5.1. By assumption, for any $\mathcal{B}_i$, the probability that $\mathbf{ULI}_1(\mathcal{B}_i)$ outputs 1 is at least $\frac{1}{1+\varepsilon_1} \frac{\texttt{uloss}(\mathcal{B}_i)}{w(\mathcal{B}_i)+h(\mathcal{B}_i)}$. $\mathcal{B}_i$ is chosen on line 1 of $\mathbf{ULI}_2$ with probability $\frac{w(\mathcal{B}_i)+h(\mathcal{B}_i)}{w(\mathcal{U})+h(\mathcal{U})}$. Therefore, the probability that $\mathbf{ULI}_2(\mathcal{U}, a, b, \mathcal{B}_0, \ldots, \mathcal{B}_k)$ outputs 1 is at least

$$\sum_{i=0}^{k} \frac{1}{1+\varepsilon_1} \frac{\texttt{uloss}(\mathcal{B}_i)}{w(\mathcal{B}_i)+h(\mathcal{B}_i)} \frac{w(\mathcal{B}_i)+h(\mathcal{B}_i)}{w(\mathcal{U})+h(\mathcal{U})} = \frac{1}{1+\varepsilon_1} \frac{\texttt{uloss}(\mathcal{U})}{w(\mathcal{U})+h(\mathcal{U})}.$$

On the other hand, the probability that $\mathbf{ULI}_1(\mathcal{B}_i)$ outputs 1 is at most $(1 + \varepsilon_1)\frac{\texttt{uloss}(\mathcal{B}_i)}{w(\mathcal{B}_i)+h(\mathcal{B}_i)} + \frac{\varepsilon_1}{n(w(\mathcal{B}_i)+h(\mathcal{B}_i))}$. Therefore, the probability that $\mathbf{ULI}_2(\mathcal{U}, a, b, \mathcal{B}_0, \ldots, \mathcal{B}_k)$

outputs 1 is at most
$\sum_{i=0}^{k}((1+\varepsilon_1)\frac{\texttt{uloss}(\mathcal{B}_i)}{w(\mathcal{B}_i)+h(\mathcal{B}_i)}+\frac{\varepsilon_1}{n(w(\mathcal{B}_i)+h(\mathcal{B}_i))})\frac{w(\mathcal{B}_i)+h(\mathcal{B}_i)}{w(\mathcal{U})+h(\mathcal{U})} \le (1+\varepsilon_1)\frac{\texttt{uloss}(\mathcal{U})}{w(\mathcal{U})+h(\mathcal{U})}+\frac{\varepsilon_1}{w(\mathcal{U})+h(\mathcal{U})}$. This establishes the quality guarantee of $\mathbf{ULI}_2$ in Tab. 5.1.

The running time of $\mathbf{ULI}_2$ is given by the cost of running $\mathbf{ULI}_1(\mathcal{B}_i)$ on the chosen input $\mathcal{B}_i$. By assumption, this cost is $\widetilde{O}_{\varepsilon_1}(\sqrt{\texttt{uloss}(\mathcal{B}_i)}+1)$. Since each $\mathcal{B}_i$ is chosen with probability $\frac{w(\mathcal{B}_i)+h(\mathcal{B}_i)}{w(\mathcal{U})+h(\mathcal{U})}$, the expected running time of $\mathbf{ULI}_2$ is
$\sum_{i=0}^{k}\frac{w(\mathcal{B}_i)+h(\mathcal{B}_i)}{w(\mathcal{U})+h(\mathcal{U})}\widetilde{O}_{\varepsilon_1}(\sqrt{\texttt{uloss}(\mathcal{B}_i)}+1)$. By assumption, $w(\mathcal{B}_i)+h(\mathcal{B}_i) = \widetilde{O}(b)$, so this quantity is at most $\frac{1}{w(\mathcal{U})+h(\mathcal{U})}\sum_{i=0}^{k}\widetilde{O}_{\varepsilon_1}(b(\sqrt{\texttt{uloss}(\mathcal{B}_i)}+1))$. Using the concavity of the square root funtion and noting that the average value of $\texttt{uloss}(\mathcal{B}_i)$ is $\texttt{uloss}(\mathcal{U})/k$, $\widetilde{O}(\sum_{i=0}^{k}(\sqrt{\texttt{uloss}(\mathcal{B}_i)}+1)) \le \widetilde{O}(k(\sqrt{\texttt{uloss}(\mathcal{U})/k}+1))$. Since $k = \widetilde{\Omega}(\frac{n}{b})$, $k\sqrt{\texttt{uloss}(\mathcal{U})/k} = \sqrt{k\texttt{uloss}(\mathcal{U})} = \widetilde{O}(\sqrt{\frac{n\texttt{uloss}(\mathcal{U})}{b}})$. Therefore,

$$\begin{aligned}\frac{1}{w(\mathcal{U})+h(\mathcal{U})}\sum_{i=0}^{k}\widetilde{O}_{\varepsilon_1}(b(\sqrt{\texttt{uloss}(\mathcal{B}_i)}+1)) &\le \widetilde{O}_{\varepsilon_1}(\frac{b}{n}\sum_{i=0}^{k}(\sqrt{\texttt{uloss}(\mathcal{B}_i)}+1))\\ &\le \widetilde{O}_{\varepsilon_1}(\frac{b}{n}(\sqrt{\frac{n\texttt{uloss}(\mathcal{U})}{b}}+\frac{n}{b}))\\ &\le \widetilde{O}_{\varepsilon_1}(\sqrt{\frac{b\texttt{uloss}(\mathcal{U})}{n}}+1).\end{aligned}$$

This establishes the running time guarantee of $\mathbf{ULI}_2$ in Tab. 5.1. □

We now prove Lemma 5.1.8.

Proof First, we note that by the Chernoff Bound, since **XGapTest** succeeds with probability at least $2/3$. the probability that the majority of $O(\log n)$ runs of **XGapTest** is wrong is bounded by $\frac{1}{n^6}$. Since this task of taking the majority answer of $O(\log n)$ runs of **XGapTest** is done at most $O(\frac{1}{\varepsilon_2}\log n)$ times, by the union bound all of these succeed with probability at least $1-\frac{1}{\varepsilon_2 n^5}$. For $\varepsilon_2 \ge \frac{1}{n^2}$, this is at least $1-\frac{1}{n^3}$.

Now assume that all majority answers are correct. First suppose that $\texttt{uloss}(\mathcal{B}) > 1$. Let j be such that $\frac{w(\mathcal{B})+h(\mathcal{B})}{(1+\varepsilon_2)^{j+1}} \le \texttt{uloss}(\mathcal{T}) \le \frac{w(\mathcal{B})+h(\mathcal{B})}{(1+\varepsilon_2)^{j}}$. If $b \le \frac{w(\mathcal{B})+h(\mathcal{B})}{(1+\varepsilon_2)^{j+1}}$, then the test on line 2a will return BIG. If $a \ge \frac{w(\mathcal{B})+h(\mathcal{B})}{(1+\varepsilon_2)^{j}}$, then the test on line 2a will return SMALL. Each time the test on line 2a returns SMALL. b decreases by a factor of $(1+\varepsilon_2)$. If this happens $j+1$ times and each time the procedure does not return 0, the test on line 2a will continue to return BIG until $\frac{b}{a} \le 1+\varepsilon_2$, at which point it will return 1 from line 3. If at some point $a \ge \frac{w(\mathcal{B})+h(\mathcal{B})}{(1+\varepsilon_2)^j}$, then the test on line 2a will continue to return SMALL until it returns 0 or b reaches $\frac{w(\mathcal{B})+h(\mathcal{B})}{(1+\varepsilon_2)^{j-1}}$, at which point the loop will stop and the procedure will return 1 from line 3. Therefore, if $\texttt{uloss}(\mathcal{B}) > 1$, $\mathbf{ULI}_1(\mathcal{B})$ will return 1 iff line 2c never returns 0, which happens with probability $\frac{\varepsilon_2}{1+\varepsilon_2}$ independently each of k times, where k is between $j-1$ and $j+1$. Therefore, the probability that it returns 1 (assuming all majority answers are correct) is between $\frac{\texttt{uloss}(\mathcal{B})}{(1+\varepsilon_2)(w(\mathcal{B})+h(\mathcal{B}))}$ and $\frac{(1+\varepsilon_2)^2\texttt{uloss}(\mathcal{B})}{w(\mathcal{B})+h(\mathcal{B})}$.

If $\texttt{uloss}(\mathcal{B}) = 1$, then if line 2c never returns 0, when the loops ends, a will be either 1 or $1+\varepsilon_2$. In either case, the procedure will return 1 via either line 3 or line 4. This will happen with probability between $\frac{1}{w(\mathcal{B})+h(\mathcal{B})}$ and $\frac{(1+\varepsilon_2)^2}{w(\mathcal{B})+h(\mathcal{B})}$ (assuming all majority answers are correct), which is again between $\frac{\texttt{uloss}(\mathcal{B})}{(1+\varepsilon_2)(w(\mathcal{B})+h(\mathcal{B}))}$ and $\frac{(1+\varepsilon_2)^2\texttt{uloss}(\mathcal{B})}{w(\mathcal{B})+h(\mathcal{B})}$.

Finally, if $\texttt{uloss}(\mathcal{B}) = 0$, then, assuming all majority answers are correct, line 2a will never return BIG, and the procedure will return 0, either on line 2c or on line 4, meaning the probability it returns 1 is $\frac{\texttt{uloss}(\mathcal{B})}{w(\mathcal{B})+h(\mathcal{B})}$.

Combining these bounds with the chance of any majority answer being incorrect via a union bound, the probability that $\mathbf{ULI}_1(\mathcal{B})$ returns 1 is between $\frac{\texttt{uloss}(\mathcal{B})}{(1+\varepsilon_2)(w(\mathcal{B})+h(\mathcal{B}))} - \frac{1}{n^3}$ and $\frac{(1+\varepsilon_2)^2\texttt{uloss}(\mathcal{B})}{w(\mathcal{B})+h(\mathcal{B})} + \frac{1}{n^3}$. If $\texttt{uloss}(\mathcal{B}) = 0$, then since the probability of returning 1 is at least 0, we can replace the lower bound with $\frac{\texttt{uloss}(\mathcal{B})}{(1+2\varepsilon_2)(w(\mathcal{B})+h(\mathcal{B}))}$. If $\texttt{uloss}(\mathcal{B}) \ge 1$, then since $\frac{1}{n^3} \ge \frac{\varepsilon_2}{(w(\mathcal{B})+h(\mathcal{B}))(1+\varepsilon_2)(1+2\varepsilon_2)}$, we can again replace the lower bound with $\frac{\texttt{uloss}(\mathcal{B})}{(1+2\varepsilon_2)(w(\mathcal{B})+h(\mathcal{B}))}$. For the upper bound, we know $\frac{1}{n^3} \le \frac{\varepsilon_1}{n(w(\mathcal{B})+h(\mathcal{B}))}$, and since $(1+\varepsilon_2)^2 \le 1+\varepsilon_1$, we can replace the upper bound with $\frac{(1+\varepsilon_1)\texttt{uloss}(\mathcal{B})+\varepsilon_1/n}{w(\mathcal{B})+h(\mathcal{B})}$. Therefore, $\mathbf{ULI}_1(\mathcal{B})$ is a $(\frac{\varepsilon_1}{1+\varepsilon_1}\frac{\texttt{uloss}(\mathcal{B})}{w(\mathcal{B})+h(\mathcal{B})}, \varepsilon_1\frac{\texttt{uloss}(\mathcal{B})+1/n}{w(\mathcal{B})+h(\mathcal{B})})$-quality `uloss`-indicator for $\mathcal{B}$.

To analyze the running time, we see that $\mathbf{ULI}_1(\mathcal{B})$ consists of running $\mathbf{XGapTest}(\mathcal{B}, a, b)$ for different values of a, b until it outputs 0 or the loop stops. For any iteration of the loop, a will be $(1+\varepsilon_2)^i$ for some i and b will be $\frac{w(\mathcal{B})+h(\mathcal{B})}{(1+\varepsilon_2)^j}$ for some j. When $b = \frac{w(\mathcal{B})+h(\mathcal{B})}{(1+\varepsilon_2)^j}$, the loop will have reached this step iff it has not returned 0 at any previous point when the majority answer on line 2a was SMALL. This will have happened with probability $\frac{1}{(1+\varepsilon_2)^j}$, and if it did happen, the cost of the iteration would be $\widetilde{O}_{\varepsilon_2}(\frac{w(\mathcal{B})\sqrt{(1+\varepsilon_2)^i}}{\frac{w(\mathcal{B})+h(\mathcal{B})}{(1+\varepsilon_2)^j}})$. This yields a total of $\widetilde{O}_{\varepsilon_2}(\sum_{i,j}\frac{1}{(1+\varepsilon_2)^j}\frac{(w(\mathcal{B})+h(\mathcal{B}))\sqrt{(1+\varepsilon_2)^i}}{\frac{w(\mathcal{B})+h(\mathcal{B})}{(1+\varepsilon_2)^j}})$, where the sum is taken over all iterations of the loop.

Looking more closely at the procedure, if a ever exceeds $\mathtt{uloss}(\mathcal{B})$, the calls to **XGapTest** will continue to return SMALL, meaning that i will never increase beyond $\log_{1+\varepsilon_2}(\mathtt{uloss}(\mathcal{B}))$. Additionally, each run of the loop makes a and b closer by a factor of $1+\varepsilon_2$, so the number of iterations of the loop will be at most polylogarithmic in $w(\mathcal{B})+h(\mathcal{B})$. Furthermore, we note that if $\mathtt{uloss}(\mathcal{B})=0$, the loop will still run with $a=1$, so for this case, it is necessary to use 0 as the upper bound for the value of i instead of $\log_{1+\varepsilon_2}(\mathtt{uloss}(\mathcal{B}))$. Simplifying, we get that the expected running time is $\tilde{O}(\sqrt{\mathtt{uloss}(\mathcal{B})})$ in the case $\mathtt{uloss}(\mathcal{B})>0$ and $\tilde{O}(1)$ in the case $\mathtt{uloss}(\mathcal{B})=0$. Therefore, we can bound the running time of $\mathbf{ULI}_1(\mathcal{B})$ by $\tilde{O}(\sqrt{\mathtt{uloss}(\mathcal{B})}+1)$. □

In order to prove Lemma 5.1.7, we will need the following proposition.

Proposition 6.2.3 *Let $\mathcal{B}$ be a box, and let $\langle x,y\rangle$ be a point on the LCS of $\mathcal{B}$. Then $|(y-y_B(\mathcal{B})+1)-(x-x_L(\mathcal{B}))|\leq\mathtt{uloss}(\mathcal{B})$.*

Proof Let $\mathcal{B}_1$ be the box $(x_L(\mathcal{B}),x]\times[y_B(\mathcal{B}),y]$, and let $\mathcal{B}_2$ be the box $(x,x_R(\mathcal{B})]\times[y,y_T(\mathcal{B})]$. Since (x,y) is on the LCS of $\mathcal{B}$, $\mathtt{lcs}(\mathcal{B}_1)+\mathtt{lcs}(\mathcal{B}_2)=\mathtt{lcs}(\mathcal{B})$, so $\mathtt{uloss}(\mathcal{B}_1)+\mathtt{uloss}(\mathcal{B}_2)=\mathtt{uloss}(\mathcal{B})$, in particular, $\mathtt{uloss}(\mathcal{B}_1)\leq\mathtt{uloss}(\mathcal{B})$. Furthermore, $\mathtt{lcs}(\mathcal{B}_1)\leq\min(w(\mathcal{B}_1),h(\mathcal{B}_1))$, so $h(\mathcal{B}_1)-w(\mathcal{B}_1)\leq w(\mathcal{B}_1)+h(\mathcal{B}_1)-2\cdot\mathtt{lcs}(\mathcal{B}_1)\leq\mathtt{uloss}(\mathcal{B})$. As a result, $(y-y_B(\mathcal{B})+1)-(x-x_L(\mathcal{B}))\leq\mathtt{uloss}(\mathcal{B})$. Similarly, applying $\mathtt{lcs}(\mathcal{B}_1)\leq h(\mathcal{B}_1)$ yields $(x-x_L(\mathcal{B}))-(y-y_B(\mathcal{B})+1)\leq\mathtt{uloss}(\mathcal{B})$. □

We now prove Lemma 5.1.7.

Proof First, note that $\mathcal{B},a',r$ satisfy the input constraints of $\mathbf{XLI}_2(\mathcal{B},a',r)$ in every call to $\mathbf{XLI}_2$ made by $\mathbf{XGapTest}(\mathcal{B},a,b)$. Therefore, $\mathbf{XLI}_2(\mathcal{B},a',r)$ is a $(4\varepsilon_3\frac{\mathtt{Xloss}(\mathcal{B})}{w(\mathcal{B})},5\varepsilon_3\frac{\mathtt{Xloss}(\mathcal{B})+1}{w(\mathcal{B})}+2\varepsilon_3\frac{h(\mathcal{B})-w(\mathcal{B})}{w(\mathcal{B})})$-quality $\mathtt{Xloss}$ indicator for $\mathcal{B}$.

Applying Prop. 5.1.1 (with $\tau=\varepsilon_3$), we get a $(\varepsilon_3,9\varepsilon_3(\mathtt{Xloss}(\mathcal{B})+1)+2\varepsilon_3(h(\mathcal{B})-w(\mathcal{B})))$-gap test for $\mathtt{Xloss}(\mathcal{B})$ (which we'll call G). We can choose κ so that the failure probability of G is at most 1/3. Unfortunately, the $2\varepsilon_3(h(\mathcal{B})-w(\mathcal{B}))$ term prevents us from applying Prop. 5.1.2 directly to get a purely multiplicative gap test. As a result, we aim to convert this to a gap test for $\texttt{uloss}(\mathcal{B})$.

For parameters a, b with $b > (1+44\varepsilon_3)a$, consider running G with parameters $b' = \frac{b+w(\mathcal{B})-h(\mathcal{B})}{2}$, and $a' = (1+9\varepsilon_3)\frac{a+w(\mathcal{B})-h(\mathcal{B})}{2} + 11\varepsilon_3 a$. If $a \geq w(\mathcal{B}) - h(\mathcal{B})$, then

$$\begin{aligned} b &> (1+\varepsilon_3)(1+42\varepsilon_3)a \\ &\geq (1+\varepsilon_3)(1+31\varepsilon_3)a + 11\varepsilon_3 a \\ &\geq (1+\varepsilon_3)(1+31\varepsilon_3)a + 11\varepsilon_3(w(\mathcal{B}) - h(\mathcal{B})) \end{aligned}$$

so

$$\begin{aligned} b + w(\mathcal{B}) - h(\mathcal{B}) &> (1+\varepsilon_3)((1+9\varepsilon_3)a + 22\varepsilon_3 a + 11\varepsilon_3(w(\mathcal{B}) - h(\mathcal{B}))) + w(\mathcal{B}) - h(\mathcal{B}) \\ &\geq (1+\varepsilon_3)((1+9\varepsilon_3)(a + w(\mathcal{B}) - h(\mathcal{B})) + 22\varepsilon_3 a) \end{aligned}$$

This yields, $\frac{b+w(\mathcal{B})-h(\mathcal{B})}{2} > (1+\varepsilon_3)((1+9\varepsilon_3)\frac{a+w(\mathcal{B})-h(\mathcal{B})}{2} + 11\varepsilon_3 a)$, so $b' > (1+\varepsilon_3)a'$. Therefore (with probability of failure at most κ), if $\texttt{Xloss}(\mathcal{B}) > b'$, G will return BIG, and if $\texttt{Xloss}(\mathcal{B}) < a' - 9\varepsilon_3(\texttt{Xloss}(\mathcal{B}) + 1) + 2\varepsilon_3(h(\mathcal{B}) - w(\mathcal{B}))$, G will return SMALL. Suppose that $\texttt{uloss}(\mathcal{B}) > b$. Then $\texttt{Xloss}(\mathcal{B}) > \frac{b+w(\mathcal{B})-h(\mathcal{B})}{2} = b'$, so G will return BIG. Suppose that $\texttt{uloss}(\mathcal{B}) < a$. Then $\texttt{Xloss}(\mathcal{B}) < \frac{a+w(\mathcal{B})-h(\mathcal{B})}{2}$. Since

$$\begin{aligned} &\frac{a+w(\mathcal{B})-h(\mathcal{B})}{2} + 9\varepsilon_3(\texttt{Xloss}(\mathcal{B}) + 1) + 2\varepsilon_3(h(\mathcal{B}) - w(\mathcal{B})) \\ &\leq \frac{a+w(\mathcal{B})-h(\mathcal{B})}{2} + 9\varepsilon_3(\frac{a+w(\mathcal{B})-h(\mathcal{B})}{2} + 1) + 2\varepsilon_3 a \\ &\leq (1+9\varepsilon_3)\frac{a+w(\mathcal{B})-h(\mathcal{B})}{2} + 11\varepsilon_3 a \\ &\leq a' \end{aligned}$$

$\texttt{Xloss}(\mathcal{B})$ is at most $a' - 9\varepsilon_3(\texttt{Xloss}(\mathcal{B}) + 1) + 2\varepsilon_3(h(\mathcal{B}) - w(\mathcal{B}))$, so G will return SMALL. Therefore, for input thresholds a, b with $a \geq \max(|w(\mathcal{B}) - h(\mathcal{B})|, 1)$, we can apply G to a', b' to get a $44\varepsilon_3$-gap test for $\texttt{uloss}(\mathcal{B})$ with error at most 1/3.

To analyze the running time, **XGapTest**$(\mathcal{B}, a, b)$ makes $\widetilde{O}(\frac{w(\mathcal{B})}{b})$ calls to $\mathbf{XLI}_2(\mathcal{B}, a', r)$ with $r = \widetilde{O}_{\varepsilon_2}(a)$. By assumption, $\mathbf{XLI}_2(\mathcal{B}, a', r)$ runs in time $\widetilde{O}_{\varepsilon_3}(\sqrt{r})$. Therefore, **XGapTest**$(\mathcal{B}, a, b)$ runs in time $\widetilde{O}_{\varepsilon_2}(\frac{w(\mathcal{B})}{b}\sqrt{a})$. □

We now prove Lemma 5.1.6.

Proof Suppose that $\mathcal{B}, a', r$ satisfy the input constraints of $\mathbf{XLI}_1$ in Tab. 5.1. Since $2a' \leq r$, for any call $\mathbf{XLI}_1(\mathcal{T})$ made by $\mathbf{XLI}_2(\mathcal{B}, a', r)$, $\mathcal{T}$ satisfies the input constraints of **BaseGapTest**. Therefore by assumption, $\mathbf{XLI}_1(\mathcal{T})$ is a $(\frac{2\varepsilon_3}{1+2\varepsilon_3}\frac{\texttt{Xloss}(\mathcal{T})}{w(\mathcal{T})}, \varepsilon_3\frac{\texttt{Xloss}(\mathcal{T})+1/n}{w(\mathcal{T})} + (1+\varepsilon_3)\frac{\texttt{terr}_{\varepsilon_3}(\mathcal{T})}{w(\mathcal{T})})$-quality `Xloss` indicator for $\mathcal{T}$. We aim to show that $\mathbf{XLI}_2(\mathcal{B}, a', r)$ is a $(4\varepsilon_3\frac{\texttt{Xloss}(\mathcal{B})}{w(\mathcal{B})}, 5\varepsilon_3\frac{\texttt{Xloss}(\mathcal{B})+1}{w(\mathcal{B})} + 2\varepsilon_3\frac{h(\mathcal{B})-w(\mathcal{B})}{w(\mathcal{B})})$-quality `Xloss` indicator for $\mathcal{B}$.

Partition $X(\texttt{extL}_r(\mathcal{B}))$ into intervals of size r, and let I_s be the set of intervals obtained by shifting this partition s indices to the right for $s \in [r]$. Let $\vec{\mathcal{T}_s} = \{I \times [x_L(I) - a', x_R(I) + a'] : I \in I_s\}$. We can think of the box $\mathcal{T}$ chosen by **XGapTest** as chosen by picking s uniformly at random, and then picking an element of $\vec{\mathcal{T}_s}$ uniformly at random.

Fix s, and let $J_1, J_2, \cdots, J_m$ be the elements of I_s. Let $\mathcal{T}_i$ be the box $J_i \times [x_L(J_i) - a', x_R(J_i) + a']$ (as in **XGapTest**), and let S_i be the sequence achieving the maximum in $\texttt{terr}_{\varepsilon_3,\varepsilon_3}(\mathcal{T}_i)$. Let $L_s = \bigcup_{\mathcal{T}_i \in \vec{\mathcal{T}_s}} S_i$. Since $y_T(\mathcal{T}_i) > y_B(\mathcal{T}_{i+1})$, it is possible that L_s is not monotone. Let L'_s be the longest common subsequence of L_s, i.e. L'_s is the resulting sequence after deleting as few points as possible from L_s to make it monotone. Let $S'_i = L'_s \cap \mathcal{T}_i$. Note that $S'_i \subseteq S_i$.

Let $\texttt{over}(L_s) = |L_s| - |L'_s|$, i.e. $\texttt{over}(L_s)$ is the number of points of L_s that must be deleted in order to make L_s monotone. We have

$$\begin{aligned}
\texttt{over}(L_s) + \sum_{\mathcal{T}_i \in \vec{\mathcal{T}_s}} \texttt{Xloss}(S_i) &= \texttt{over}(L_s) + \texttt{Xloss}(L_s) \\
&= \texttt{Xloss}(L'_s) \\
&\geq \texttt{Xloss}(\mathcal{B}) \\
&\geq \frac{\texttt{uloss}(\mathcal{B}) + w(\mathcal{B}) - h(\mathcal{B})}{2}
\end{aligned}$$

so we aim to bound $\texttt{over}(L_s)$.

We see that any point on S_i contributing to $\texttt{over}(L_s)$ will be in violation with some point on S_{i-1} or S_{i+1} (but not both). As a result, if we let $\texttt{over}_i(L_s)$ be the number of points P contributing to $\texttt{over}(L_s)$ with $y(P) \in Y(\mathcal{T}_i) \cap Y(\mathcal{T}_{i+1})$, we have $\texttt{over}(L_s) = \sum_i \texttt{over}_i(L_s)$. We now form a partition of the boxes $\mathcal{T}_i$ with $\texttt{over}_i(L_s) \neq 0$

into $\texttt{LONG}_s$ and $\texttt{SHORT}_s$. Say that $\mathcal{T}_i \in \texttt{LONG}_s$ if there are violating points P_1, P_2 on S_i, S_{i+1} respectively, with $x(P_2) - x(P_1) \geq \frac{8a'}{\varepsilon_3}$. Otherwise, $\mathcal{T}_i \in \texttt{SHORT}_s$.

First consider $\mathcal{T}_i \in \texttt{LONG}_s$. Let P_1, P_2 be violating points on S_i, S_{i+1}, respectively with $x(P_2) - x(P_1) \geq \frac{8a'}{\varepsilon_3}$. Since $|Y(\mathcal{T}_i) \cap Y(\mathcal{T}_{i+1})| = 2a' + 1$ and $y(P_1) > y_B(\mathcal{T}_{i+1}), y(P_2) < y_T(\mathcal{T}_i)$, at most $2a' - 1$ of the at least $\frac{8a'}{\varepsilon_3} - 1$ indices between P_1 and P_2 can have matches on L_s. Therefore, $\texttt{Xloss}(S_i) + \texttt{Xloss}(S_{i+1}) \geq \frac{8a'}{\varepsilon_3} - 2a' \geq \frac{6a'}{\varepsilon_3}$. Furthermore, since again $|Y(\mathcal{T}_i) \cap Y(\mathcal{T}_{i+1})| = 2a' + 1$, we can always delete at most a' points to remove all violations between $\mathcal{T}_i$ and $\mathcal{T}_{i+1}$, so $\texttt{over}_i(L_s) \leq a'$. Summing over all $\mathcal{T}_i \in \texttt{LONG}_s$, the total contribution to $\texttt{over}(L_s)$ by $\texttt{LONG}_s$ is at most $\frac{\varepsilon_3}{3} \sum_i \texttt{Xloss}(S_i) \leq \frac{\varepsilon_3}{3}(1 + \varepsilon_3)\texttt{Xloss}(\mathcal{B}) \leq \frac{2\varepsilon_3}{3}\texttt{Xloss}(\mathcal{B})$.

Now consider $\mathcal{T}_i \in \texttt{SHORT}_s$. Let K_i be the smallest X-interval containing all indices of violating pairs of points in $\mathcal{T}_i, \mathcal{T}_{i+1}$ on L_s. Since $\mathcal{T}_i \in \texttt{SHORT}_s$, $|K_i| \leq \frac{8a'}{\varepsilon_3}$. For a choice of s', say that an X-interval J is *severed* by s' if $\exists h \in \mathbb{Z} : s' + hr \in J$. We note that, if we choose s' uniformly at random, the probability that this interval K_i is severed by s' is at most $\frac{8\varepsilon_3}{100}$. Additionally, if K_i is not severed by s', then since it lies within some $\mathcal{T} \in \vec{\mathcal{T}}_{s'}$, its contribution to $\texttt{Xloss}(\mathcal{T})$ will be $\texttt{over}_i(L_s)$. Therefore, if we look at the expected contribution of $\texttt{SHORT}_s$ to $\texttt{over}(L_s)$ when choosing s uniformly at random, by applying linearity of expectation over all K_i coming from each choice of s, this expected contribution will be at most $\frac{8\varepsilon_3}{100}\texttt{Xloss}(\mathcal{B})$. Therefore, by Markov's Inequality, with probability at least $3/4$, this contribution is at most $\frac{\varepsilon_3}{3}\texttt{Xloss}(\mathcal{B})$.

Putting this together with the contribution from $\texttt{LONG}_s$, with probability at least $3/4 - n^{-\Omega(\log n)}$, $\texttt{over}(L_s)$ is at most $\varepsilon_3\texttt{Xloss}(\mathcal{B})$. Therefore, with probability at least $3/4 - n^{-\Omega(\log n)}$, $\sum_{\mathcal{T}_i \in \vec{\mathcal{T}}_s} \texttt{Xloss}(S_i) \geq (1 - \varepsilon_3)\texttt{Xloss}(\mathcal{B})$. As a result, $\mathbf{XLI}_2(\mathcal{B}, a', r)$ returns 1 with probability at least $\frac{1-\varepsilon_3}{1+2\varepsilon_3}\frac{\texttt{Xloss}(\mathcal{B})}{w(\mathcal{B})} \geq (1 - 4\varepsilon_3)\frac{\texttt{Xloss}(\mathcal{B})}{w(\mathcal{B})}$.

Next, we aim to upper bound the probability that $\mathbf{XLI}_2$ returns 1. Note that $X(\mathcal{B}) \subseteq \bigcup J_i$. By Prop. 6.2.3, for any point P on the LCS of $\mathcal{B}$, if $x(P) \in J_i$, then $y(P) \in Y(\mathcal{T}_i)$. This means that $\texttt{lcs}(\mathcal{B}) \cap J_i|\mathcal{B} \subseteq \mathcal{T}_i$, so the LCS of each $\mathcal{T}_i$ is at least as long as the portion of $\texttt{lcs}(\mathcal{B})$ lying in $J_i|\mathcal{B}$. Since the points in $\mathcal{T}_1, \mathcal{T}_m$ which lie outside of $\mathcal{B}$ form common subsequences outside the Y range of $\mathcal{B}$, they do not contribute to

$\texttt{Xloss}(\mathcal{T}_1)$, $\texttt{Xloss}(\mathcal{T}_m)$, respectively, so $\sum_{i=1}^{m} \texttt{Xloss}(\mathcal{T}_i) \leq \texttt{Xloss}(\mathcal{B})$.

Next, we aim to bound $\sum_{i=1}^{m} \texttt{terr}_{\varepsilon_3}(\mathcal{T}_i)$. Let S_i, S_i' be as defined above, so

$$\texttt{terr}_{\varepsilon_3}(\mathcal{T}_i) = \texttt{serr}_{\varepsilon_3}(S_i, \mathcal{T}_i) \leq \texttt{herr}(S_i') + \varepsilon_3 \texttt{Yloss}_{\texttt{trim}}(S_i')$$

Since L_s' is monotone, $\sum_{i=1}^{m} \texttt{Yloss}_{\texttt{trim}}(S_i')$ counts each Y-index not on L_s' at most once, so

$$\varepsilon_3 \sum_{i=1}^{m} \texttt{Yloss}_{\texttt{trim}}(S_i') \leq \varepsilon_3 \Big(\sum_{i=1}^{m} (\texttt{Xloss}(S_i')) + h(\mathcal{B}) - w(\mathcal{B})\Big)$$

This gives us

$$\begin{aligned} &\sum_{i=1}^{m} \texttt{herr}(S_i') + \varepsilon_3 \texttt{Yloss}_{\texttt{trim}}(S_i') \\ \leq &\sum_{i=1}^{m} ((1+\varepsilon_3)\texttt{Xloss}(S_i') - \texttt{Xloss}(\mathcal{T}_i)) + \varepsilon_3(h(\mathcal{B}) - w(\mathcal{B})) \end{aligned}$$

We have

$$\begin{aligned} \sum_{i=1}^{m} \texttt{Xloss}(S_i') &= \texttt{over}(L_s) + \sum_{i=1}^{m} \texttt{Xloss}(S_i) \\ &\leq \texttt{over}(L_s) + (1+\varepsilon_3) \sum_{i=1}^{m} \texttt{Xloss}(\mathcal{T}_i) \\ &\leq \texttt{over}(L_s) + (1+\varepsilon_3)\texttt{Xloss}(\mathcal{B}) \end{aligned}$$

As shown above, with probability at least $3/4 - n^{-\Omega(\log n)}$, $\texttt{over}(L_s) \leq \varepsilon_3 \texttt{Xloss}(\mathcal{B})$. Therefore,

$$\begin{aligned} &\sum_{i=1}^{m} ((1+\varepsilon_3)\texttt{Xloss}(S_i') - \texttt{Xloss}(\mathcal{T}_i)) + \varepsilon_3(h(\mathcal{B}) - w(\mathcal{B})) \\ &\qquad \leq \varepsilon_3(3 + 2\varepsilon_3)\texttt{Xloss}(\mathcal{B}) + \varepsilon_3(h(\mathcal{B}) - w(\mathcal{B})) \end{aligned}$$

Putting things together, the probability that $\mathbf{XLI}_2(\mathcal{B}, a', r)$ outputs 1 is at most $\sum_{\mathcal{T}} \frac{w(\mathcal{T})}{w(\mathcal{B})}((1+\varepsilon_3)\frac{\texttt{Xloss}(\mathcal{T})}{w(\mathcal{T})} + (1+\varepsilon_3)\frac{\texttt{terr}_{\varepsilon_3}(\mathcal{T})}{w(\mathcal{T})} + \frac{\varepsilon_3}{nw(\mathcal{T})}) \leq (1+5\varepsilon_3)\frac{\texttt{Xloss}(\mathcal{B})+1}{w(\mathcal{B})} + 2\varepsilon_3 \frac{h(\mathcal{B})-w(\mathcal{B})}{w(\mathcal{B})}$. Therefore, $\mathbf{XLI}_2(\mathcal{B}, a', r)$ is a $(4\varepsilon_3 \frac{\texttt{Xloss}(\mathcal{B})}{w(\mathcal{B})}, 5\varepsilon_3 \frac{\texttt{Xloss}(\mathcal{B})+1}{w(\mathcal{B})} + 2\varepsilon_3 \frac{h(\mathcal{B})-w(\mathcal{B})}{w(\mathcal{B})})$-quality $\texttt{Xloss}$ indicator for $\mathcal{B}$.

The running time claim follows trivially from the assumptions regarding $\mathbf{XLI}_1$, as $\mathbf{XLI}_2(\mathcal{B}, a', r)$ consists of running $\mathbf{XLI}_1$ on a box of size r. □

We now prove Lemma 5.1.5.

Proof First, we note that by the Chernoff Bound, since **BaseGapTest** succeeds with probability at least 2/3. the probability that the majority of $O(\log n)$ runs of **BaseGapTest** is wrong is bounded by $\frac{1}{n^6}$. Since this task of taking the majority answer of $O(\log n)$ runs of **BaseGapTest** is done at most $O(\frac{1}{\varepsilon_3}\log n)$ times, by the union bound all of these succeed with probability at least $1 - \frac{1}{\varepsilon_3 n^5}$. Since $\varepsilon_3 \geq \frac{1}{n^2}$, this is at least $1 - \frac{1}{n^3}$.

Now assume that all majority answers are correct. Let j_1, j_2 be the largest and smallest nonnegative integers respectively such that $\frac{w(\mathcal{T})}{(1+\varepsilon_3)^{j_2}} \leq \texttt{Xloss}(\mathcal{T}) \leq \texttt{Xloss}(\mathcal{T}) + \texttt{terr}_{\varepsilon_3}(\mathcal{T}) \leq \frac{w(\mathcal{T})}{(1+\varepsilon_3)^{j_1}}$. In $\textbf{XLI}_1(\mathcal{T})$, the test on line 2a will return SMALL for $i \leq j_1 - 1$, and it will return BIG for $i = j_2$. In order for $\textbf{XLI}_1(\mathcal{T})$ to return 1, it has to not return 0 via line 2b on each of the first j_1 iterations of the loop. This happens with probability $\frac{1}{(1+\varepsilon_3)^{j_1}}$, so $\textbf{XLI}_1(\mathcal{T})$ returns 1 with probability at most $\frac{1}{(1+\varepsilon_3)^{j_1}} \leq (1 + \varepsilon_3)\frac{\texttt{Xloss}(\mathcal{T}) + \texttt{terr}_{\varepsilon_3}(\mathcal{T})}{w(\mathcal{T})}$.

On the other hand, the test on line 2a will return BIG after at most j_2 iterations of the loop, meaning line 2b has a chance of returning 0 on at most j_2 iterations. The probability that it does not return 0 on any of these is at least $\frac{1}{(1+\varepsilon_3)^{j_2}}$, so $\textbf{XLI}_1(\mathcal{T})$ returns 1 with probability at least $\frac{1}{(1+\varepsilon_3)^{j_2}} \geq \frac{\texttt{Xloss}(\mathcal{T})}{(1+\varepsilon_3)w(\mathcal{T})}$.

Combining this with the probability that any of the majority answers are incorrect via a union bound, the probability that $\textbf{XLI}_1(\mathcal{T})$ returns 1 is at least $\frac{\texttt{Xloss}(\mathcal{T})}{(1+\varepsilon_3)w(\mathcal{T})} - \frac{1}{n^3}$ and at most $(1 + \varepsilon_3)\frac{\texttt{Xloss}(\mathcal{T}) + \texttt{terr}_{\varepsilon_3}(\mathcal{T})}{w(\mathcal{T})} + \frac{1}{n^3}$. Looking more closely at the lower bound, if $\texttt{Xloss}(\mathcal{T}) = 0$, then since the probability of returing 1 is at least 0, it must be at least $\frac{\texttt{Xloss}(\mathcal{T})}{(1+2\varepsilon_3)w(\mathcal{T})}$. If instead $\texttt{Xloss}(\mathcal{T}) \geq 1$, then since $n \geq w(\mathcal{T})$, $\frac{1}{n^2} \leq \frac{\varepsilon_3}{(1+\varepsilon_3)(1+2\varepsilon_3)}$, $\frac{\texttt{Xloss}(\mathcal{T})}{(1+\varepsilon_3)w(\mathcal{T})} - \frac{1}{n^3} \leq \frac{\texttt{Xloss}(\mathcal{T})}{(1+2\varepsilon_3)w(\mathcal{T})}$. For the upper bound, since $\frac{1}{n^3} \leq \frac{\varepsilon_3}{nw(\mathcal{T})}$. $(1+\varepsilon_3)\frac{\texttt{Xloss}(\mathcal{T}) + \texttt{terr}_{\varepsilon_3}(\mathcal{T})}{w(\mathcal{T})} + \frac{1}{n^3}$ is at most $(1+\varepsilon_3)\frac{\texttt{Xloss}(\mathcal{T}) + \texttt{terr}_{\varepsilon_3}(\mathcal{T})}{w(\mathcal{T})} + \frac{\varepsilon_3}{nw(\mathcal{T})}$. Therefore, $\textbf{XLI}_1(\mathcal{T})$ is a $(\frac{2\varepsilon_3}{1+2\varepsilon_3}\frac{\texttt{Xloss}(\mathcal{T})}{w(\mathcal{T})}, \varepsilon_3\frac{\texttt{Xloss}(\mathcal{T})+1/n}{w(\mathcal{T})} + (1+\varepsilon_3)\frac{\texttt{terr}_{\varepsilon_3}(\mathcal{T})}{w(\mathcal{T})})$-quality $\texttt{Xloss}$ indicator for $\mathcal{T}$.

To analyze the running time, we see that $\textbf{XLI}_1(\mathcal{T})$ consists of running **BaseGapTest**$(\mathcal{T}, a, b)$ for different values of a, b with $b = (1+\varepsilon_3)a$ until it outputs 0 or

1. Once i exceeds the value of $\log_{1+\varepsilon_3}(\frac{w(\mathcal{T})}{\texttt{Xloss}(\mathcal{T})})$, $\frac{w(\mathcal{T})}{(1+\varepsilon_3)^i}$ will be smaller than $\texttt{Xloss}(\mathcal{T})$, meaning **BaseGapTest**$(\mathcal{T}, \frac{w(\mathcal{T})}{(1+\varepsilon_3)^{i+1}}, \frac{w(\mathcal{T})}{(1+\varepsilon_3)^i})$ will return BIG and the loop will stop. For i between 0 and $\log_{1+\varepsilon_3}(\frac{w(\mathcal{T})}{\texttt{Xloss}(\mathcal{T})})$, the loop will reach the i^{th} step if it has not returned 0 in any previous step. This happens with probability $\frac{1}{(1+\varepsilon_3)^i}$, and if it does happen, the cost of the i^{th} step will be $\tilde{O}((1+\varepsilon_3)^i\sqrt{w(\mathcal{T})})$. Summing over all i, the total expected running time is $\tilde{O}(\sqrt{w(\mathcal{T})})$. $\square$

We now prove Lemma 5.1.4. We will need the following claim, which is a minor modification of Claim 4.1 in [4].

Claim 6.2.4 *Consider a set S of n elements and a subset T of m elements. Pick a random subset X of S by picking each element independently with probability p. Let $q = |X \cap T|$. Picking X can be implemented in expected $O(pn)$ time and* $\Pr[|q/p - m| \geq \varepsilon_1 m + \frac{16(2e-1)\log n}{\varepsilon_1^2 p}] \leq \frac{1}{n^3}$

Proof We pick X as follows. Divide S into blocks of size $\frac{1}{p}$ and use the binomial distribution to compute the number of samples in each block. Finally, pick the samples from each block according to the computed number of samples. The expected running time is $O(pn)$. Now consider two cases.

If $m \leq \frac{16\log n}{\varepsilon_1^2 p}$, then by Theorem 2.5.2,
$\Pr[|q/p - m| \geq m * \frac{16(2e-1)\log n}{\varepsilon_1^2 pm}] \leq 2^{-(1+\frac{16(2e-1)\log n}{\varepsilon_1^2 pm})pm} \leq \frac{1}{n^3}$.

If $m > \frac{16\log n}{\varepsilon_1^2 p}$, then by Theorem 2.5.2,
$\Pr[|q/p - m| \geq \varepsilon_1 m] \leq 2e^{-\varepsilon_1^2 pm/4} \leq 2e^{-4\log n} \leq \frac{1}{n^3}$. $\square$

We use Claim 6.2.4 to prove Lemma 5.1.4.

Proof First, note that since **XLI**$_0$ succeeds with probability 2/3, the majority taken on line 3 succeeds with probability $1 - n^{-\Omega(\log n)}$.

Let $d_{out}(\mathcal{T})$ be the number of indices in $X(\mathcal{T})$ whose match lies outside of $\mathcal{T}$. Let $d_{in}(\mathcal{T})$ be the number of indices in $X(\mathcal{T})$ whose match lies inside $\mathcal{T}$, but not on the LCS of $\mathcal{T}$ (Note that this definition is a rewording of the definition given in section 2.1.5). We have $d_{out}(\mathcal{T}) + d_{in}(\mathcal{T}) = \texttt{Xloss}(\mathcal{T})$. Our goal will be to show that V approximates $d_{out}(\mathcal{T})$ and W approximates $d_{in}(\mathcal{T})$.

We first show V approximates $d_{out}(\mathcal{T})$. Let N be the number of matches in $\mathcal{T}$. When $b \leq \frac{16(2e-1)}{\varepsilon_4^2}\sqrt{w(\mathcal{T})}\log n$, we get the exact value of N (and therefore $d_{out}(\mathcal{T})$) by reading the whole block. Now consider the case $b > \frac{16(2e-1)}{\varepsilon_4^2}\sqrt{w(\mathcal{T})}\log n$. As defined in **BaseGapTest**, let $\mathcal{C}$ be the number of collisions which lie in $\mathcal{T}$. Since each point has its X-index and Y-index each selected independently with probability p, we have $\mathbb{E}[C] = p^2 N$. Consider two cases.

If $N > \frac{2\varepsilon_4 b}{(2e-1)}$, then by Theorem 2.5.2,

$$\begin{aligned}
\Pr[|\frac{C}{p^2} - N| \geq N\frac{2\varepsilon_4 b}{N}] &\leq 2e^{-(\frac{2\varepsilon_4 b}{N})^2 p^2 \frac{N}{4}} \\
&\leq 2e^{-\frac{\varepsilon_4^2 b^2}{N} * \frac{C' w(\mathcal{T}) \log^2 n}{b^2}} \\
&\leq 2e^{-C'\varepsilon_4^2 \log^2 n} \\
&\leq \frac{1}{n^2}.
\end{aligned}$$

If $N \leq \frac{2\varepsilon_4 b}{(2e-1)}$, then by Theorem 2.5.2,

$$\begin{aligned}
\Pr[|\frac{C}{p^2} - N| \geq N\frac{2\varepsilon_4 b}{N}] &\leq 2^{-(1+\frac{2\varepsilon_4 b}{N})p^2 N} \\
&\leq 2^{-(1+\frac{2\varepsilon_4 b}{N}) * \frac{C' N w(\mathcal{T}) \log^2 n}{b^2}} \\
&\leq 2^{-2C'\varepsilon_4 \frac{w(\mathcal{T})}{b} \log^2 n} \\
&\leq \frac{1}{n^2}.
\end{aligned}$$

Thus, with high probability, $|\frac{C}{p^2} - N| < 2\varepsilon_4 b$, so $|V - d_{out}(\mathcal{T})| < 2\varepsilon_4 b$

Next, we show that W approximates $d_{in}(\mathcal{T})$. Let m be the number of matches in $\mathcal{T}$ that would be classified as bad by $\mathbf{XLI}_0$. By assumption,

$$d_{in}(\mathcal{T}) \leq m \leq (1+\varepsilon_4)(\texttt{Xloss}(\mathcal{T}) + \texttt{terr}_{\varepsilon_3}(\mathcal{T})) - d_{out}(\mathcal{T})$$

By Claim 6.2.4, with high probability,

$$(1-\varepsilon_4)m - \frac{16(2e-1)\log n}{\varepsilon_4^2 r^2} \leq \frac{D}{r^2} \leq (1+\varepsilon_4)m + \frac{16(2e-1)\log n}{\varepsilon_4^2 r^2}$$

Thus, with high probability,

$$\begin{aligned}
&W - (\texttt{Xloss}(\mathcal{T}) + \texttt{terr}_{\varepsilon_3}(\mathcal{T}) - d_{out}(\mathcal{T})) \\
&\leq \varepsilon_4((1+\varepsilon_4)(\texttt{Xloss}(\mathcal{T}) + \texttt{terr}_{\varepsilon_3}(\mathcal{T})) - d_{out}(\mathcal{T})) + \varepsilon_4 b
\end{aligned}$$

and

$$d_{in}(\mathcal{T}) - W \leq \varepsilon_4 d_{in}(\mathcal{T}) + \varepsilon_4 b$$

If $\texttt{Xloss}(\mathcal{T}) + \texttt{terr}_{\varepsilon_3}(\mathcal{T}) < a \leq \frac{b}{1+10\varepsilon_4}$, then

$$\begin{aligned}
&|(V+W) - (\texttt{Xloss}(\mathcal{T}) + \texttt{terr}_{\varepsilon_3}(\mathcal{T}))| \\
&\leq |V - d_{out}(\mathcal{T})| + |W - (\texttt{Xloss}(\mathcal{T}) + \texttt{terr}_{\varepsilon_3}(\mathcal{T}) - d_{out}(\mathcal{T}))| \\
&\leq 2\varepsilon_4 b + \varepsilon_4(1+\varepsilon_4)b + \varepsilon_4 b \\
&\leq 5\varepsilon_4 b
\end{aligned}$$

Therefore, $V + W < (1 - 4\varepsilon_4)b$.

On the other hand, if $\texttt{Xloss}(\mathcal{T}) > b$, then

$$\begin{aligned}
V + W &\geq d_{out}(\mathcal{T}) - 2\varepsilon_4 b + (1-\varepsilon_4)d_{in}(\mathcal{T}) - \varepsilon_4 b \\
&> (1-\varepsilon_4)b - 3\varepsilon_4 b \\
&\geq (1-4\varepsilon_4)b
\end{aligned}$$

Thus, the test on line 4 successfully distinguishes between these two cases.

To analyze the running time, first note that if $b = \tilde{O}(\sqrt{w(\mathcal{T})})$, then $\frac{w(\mathcal{T})\sqrt{w(\mathcal{T})}}{b} = \tilde{\Omega}(w(\mathcal{T}))$, meaning that if the condition on line 1a is met, the linear amount of time it takes to read each character in $X(\mathcal{T}), Y(\mathcal{T})$ is within this bound. The number of characters read on lines 1b and 2 is also seen to be within this bound with high probability by a standard Chernoff argument (for line 2 using the fact that $b \leq w(\mathcal{T})$. Furthermore, another Chernoff argument gives us that, with high probability, the number of collisions on line 2 is $\tilde{O}(\frac{w(\mathcal{T})}{b})$. By assumption $\mathbf{XLI}_0$ runs in time $\tilde{O}(\sqrt{w(\mathcal{T})})$, so line 3 runs in time $\tilde{O}(\frac{w(\mathcal{T})\sqrt{w(\mathcal{T})}}{b})$. Therefore, all of these steps can be shown to run in time $\tilde{O}(\frac{w(\mathcal{T})}{b}\sqrt{w(\mathcal{T})})$. $\square$

Chapter 7

Implementing the LIS Algorithm

7.1 Improving the running time of the LIS algorithm

In this section, we present our improvement to the LIS algorithm of [21], as mentioned in section 5.1.2. In the LIS setting, we will refer to X-indices as indices. We also borrow the following notation from [21].

- $\texttt{lis}(\mathcal{B}) = \texttt{lis}_f(\mathcal{B})$ is the size of a longest increasing (point) sequence (LIS) contained in $\mathcal{B}$.
- $\texttt{loss}(\mathcal{B})$ is the smallest number of matches in $\mathcal{B}$ that must be deleted so that the remaining matches in $\mathcal{B}$ form an increasing sequence.
- $\varepsilon_f = \texttt{loss}_f/n$ is the fraction of matches that must be deleted so that the remaining points form an increasing sequence. In a slight abuse of notation, ε_I refers to ε_f restricted to the index interval I, i.e. ε_I is the fraction of matches in the strip $I|\mathcal{B}$ that must be deleted so that the remaining points in the strip form an increasing sequence.

We begin with the following definition:

Definition Let $0 < \mu < 1/2$. We say that i is μ-safe if for all index intervals $[i, j]$, the number of violations with i is at most $\mu|[i, j]| = \mu(j - i + 1)$. (A similar condition holds for all $[j, i]$.)

In [21], a similar definition ((μ, L)-safe) is defined. It is worth noting that this characterization of being μ-safe is equivalent to the characterization of being $(\mu, 0)$-safe. Additionally, an index i is μ-unsafe if it is not μ-safe. In that case, there is a *witnessing interval* where the safeness is violated.

Claim 7.1.1 *Every μ-unsafe index has a witnessing interval that contains no μ-safe index.*

Proof Let i be μ-unsafe and $[i, j]$ be the smallest witnessing interval. (The proof is analogous if some $[j, i]$ is the witnessing interval.) Suppose $k \in [i, j]$ is μ-safe. We split into two cases.

- i, k are violation: Then at least half the indices in $[i, k]$ are violations with either i or k. Since k is μ-safe, at least half of this interval is in violation with i. But then $[i, k]$ is a witnessing interval for i, which is smaller than $[i, j]$. Contradiction.
- i, k are comparable: The number of violations with k in $[k, j]$ is at most $\mu(j-k+1)$. This also upper bounds the number of violations with i in $[k, j]$. Hence, the number of violations with i in $[i, k)$ is at least $\mu(j - i + 1) - \mu(j - k + 1) = \mu(k - i) = \mu|[i, k)|$. Thus, $[i, k)$ is a witnessing interval. Contradiction.

□

We state a version of the key dichotomy lemma proved in [21], which was a generalization of a lemma from [1].

Lemma 7.1.2 *Let $\mu < 1/2$. All μ-safe points form an increasing sequence. The number of μ-unsafe points is at most $(1 + 1/\mu)\varepsilon_f n$.*

The following is a corollary of the previous results.

Lemma 7.1.3 *Consider two μ-safe indices $i < j$ and let $I = (i, j)$. Suppose the number of μ-unsafe indices in I is at least $|I|/6$. Then $\varepsilon_I \geq \mu/12$.*

Proof Consider any μ-unsafe point in I. By Claim 7.1.1, there exists a witnessing interval lying completely in I (as i, j are μ-safe). Let us focus our attention completely on I, and work with the function $f|_I$. The number of μ-unsafe indices is still at least $5|I|/6$. Combining with bound from Lemma 7.1.2, $(1 + 1/\mu)\varepsilon_I|I| \geq |I|/6$. Hence, $\varepsilon_I \geq \mu/12$. □

In [21], there is a procedure **FindSplitter** which takes as input a box and essentially attempts to find a μ-safe index in that box. Using this building block, [21] defines a further procedure **TerminalBox**, which takes as input a box and an index i within the horizontal range of the box, and attempts to find a subbox containing i with few μ-safe indices. It does this by calling **FindSplitter** to find a μ-safe index j, and repeating **FindSplitter** on a subbox containing i determined by the point at j. If **FindSplitter** finds i itself as a μ-safe index, the singleton interval containing i is output.

Our improvement will use this procedure **TerminalBox**. To do so, we establish some properties of **TerminalBox**. We first focus our attention on **FindSplitter** calls made by **TerminalBox**.

Claim 7.1.4 *Let $\mathcal{B}$ be a box given to a call of* **TerminalBox** *as input. Any* **FindSplitter** *call made by* **TerminalBox** *has the following properties (whp).*

- *If an index k is returned, then $k \in X(\mathcal{B})$, k is 2μ-safe, and k is ρ-balanced in $\mathcal{B}$.*
- *If no index is returned, then there are at most $5w(\mathcal{B})/6$ μ-safe indices in $X(\mathcal{B})$.*
- *The running time of this call to* **FindSplitter** *is at most $(\mu^{-1} \log n)^c$.*

Proof First, Proposition 5.8 of [21] establishes the validity of the running time claim. Addressing the other two properties, this proposition also guarantees that the output is reliable, as defined earlier in [21]. Unpacking this definition gives us the following. First, let $B_{k,\mathcal{B}}$ be the number of violations with k in $\mathcal{B}$, and let $G_{k,\mathcal{B}}$ be the number of comparable points with k in $\mathcal{B}$. Reliability says that if **FindSplitter** returns k, then $(1-\mu)B_{k,\mathcal{B}} - \mu G_{k,\mathcal{B}} \leq C\gamma w(\mathcal{B})$. Moving things around, we get $B_{k,\mathcal{B}} \leq \mu(B_{k,\mathcal{B}} + G_{k,\mathcal{B}} + C\gamma w(\mathcal{B}) \leq (\mu + C\gamma)w(\mathcal{B})$. Upon inspection, it is apparent that the setting of the parameters in [21] ensures $C\gamma \leq \mu$, meaning k is 2μ-safe. Reliability also ensures that $k \in X(\mathcal{B})$ and k is ρ-balanced in $\mathcal{B}$, and that if no index is returned, then at most $\rho w(\mathcal{B})$ indices are (μ, L)-safe. Since $\rho \leq 5/6$ and any μ-safe index is (μ, L)-safe, the second item in the claim is satisfied as well. □

We now have the following claim.

Claim 7.1.5 *Consider the sequence $0 = s_1 < s_2 < \ldots < s_k = n+1$ of all indices where* **TerminalBox**$(s_\ell, \mathcal{B}) = s_\ell$.

- *For all ℓ, s_ℓ is μ-safe.*

- *Let* $I_\ell = (s_\ell, s_{\ell+1})$ *be non-empty. Then* $\varepsilon_{I_\ell} \geq \mu/24$ *and* $\forall i \in I_\ell$,

TerminalBox$(i, \mathcal{B}) = I_\ell$.

- *All calls to* **TerminalBox**$(i, \mathcal{B})$ *take* $(\mu^{-1} \log n)^{c+1}$ *time.*

Proof By the properties given by Claim 7.1.4, s_ℓ is 2μ-safe. It is also apparent that $\forall i \in I_\ell$, **TerminalBox**$(i) = I_\ell$. For any non-empty I_ℓ, **FindSplitter** failed to return a point. By Claim 7.1.4, there are most $5|I_\ell|/6$ μ-safe indices in I_ℓ. By Lemma 7.1.2, $\varepsilon_{I_\ell} \geq \mu/24$. The runtime bound holds because in any recursive call, the size of the argument I decreases by a constant fraction. Hence, $O(\log n)$ calls to **FindSplitter** are made, where each call runs in $(\mu^{-1} \log n)^c$. $\square$

7.1.1 Reducing additive error to multiplicative error

In [21] a procedure **Classify** is used to label points as "good" or "bad". Several properties of **Classify** are proved in [21], we list 3 of these in the following theorem.

Theorem 7.1.6 *Fix any* $\delta < 1$, *and an input array* $g : [m] \mapsto \mathbb{R}$. *Let* $\mathcal{B}$ *be the box given by* g. *There exists a procedure* **Classify**$_{t_{max}}(i, \mathcal{B}, \delta)$ *that outputs either "good" or "bad" and has the following properties (whp over all possible calls).*

- *Each call takes* $(1/\delta)^{1/\delta} \log^c n$ *time.*
- *The good points form an increasing sequence.*
- *The number of bad points is in the range* $[\varepsilon_g m, (\varepsilon_g + \delta)m]$.

Proof The first property is established by Theorem 1.1 in [21], which is proved by setting $\overline{\delta} = \overline{\tau} = \delta/2$ in Theorem 4.2 in [21]. The second property is established by Proposition 8.3 in [21]. For the third property, we look at section 8.3 in [21]. Letting $Good_g$ be the set of good points, and Bad_g be the set of bad points, [21] defines $\Delta = \texttt{lis}_g - |Good_g| - \overline{\tau}\texttt{loss}_g$, and proceeds to show $\Delta \leq \overline{\delta} m - |\zeta_1|$, where ζ_1 is an error term. Using these facts and moving things around, we get

$$\begin{aligned}|Good_g| &\geq \texttt{lis}_g - \overline{\tau}\texttt{loss}_g - \overline{\delta}m + |\zeta_1| \\ &\geq m - \varepsilon_g m - \delta/2m - \delta/2\texttt{loss}_g + |\zeta_1| \\ &\geq m - \varepsilon_g m - \delta m \\ &= m - (\varepsilon_g + \delta)m\end{aligned}$$

so $|Bad_g| \leq (\varepsilon_g + \delta)m$. This completes the proof of the third property, and the lower bound on $|Bad_g|$ follows from the second property. □

We will apply the procedure **Classify** on subintervals I of $[n]$. We will refer to this invocations as **Classify**$_I$. It basically means that the function g in Theorem 7.1.6 is set to $f|_I$. Later in the paper, μ is set to be a global parameter, but since this parameter is being used here, we include it as an input. Additionally in **Classify**, we set the multiplicative error parameter $\delta = \mu^2/24$.

```
DMI_1($i, \mathcal{B}, \mu$)
Output: Label ACCEPT or REJECT.
1. Call TerminalBox_{t_max}($i, \mathcal{B}, \mu$) and let $I$ be the index interval of the output.
2. If $I$ is the singleton interval $i$, ACCEPT.
3. Denote $I = (s, s')$. If $i$ is in violation with either $s$ or $s'$, REJECT.
4. Otherwise, call Classify_{t_max}($i, I \times Y(\mathcal{B}), \mu^2/24$). If this outputs "good", ACCEPT, else REJECT.
```

Lemma 7.1.7 *Suppose* $I_\ell = (s_\ell, s_{\ell+1})$ *is non-empty. The number of indices in* I_ℓ *rejected by* **DMI**$_1$ *is at most* $(1 + 3\mu)\varepsilon_I|I| + 4\mu$.

Proof Henceforth, we say that i is good/bad depending on the output of **Classify**$_{t_{max}}(i, I_\ell \times Y(\mathcal{B}), \mu^2/24)$. Index $i \in I_\ell$ is rejected for two reasons: either i is in violation with the end indices s_ℓ or $s_{\ell+1}$, or i is bad. Let x be the rightmost good index in violation with s_ℓ, and y be the leftmost good index in violation with $s_{\ell+1}$. Any good index in (x, y) is consistent with s_ℓ and $s_{\ell+1}$. Since the good indices form an increasing sequence, the number of good indices in $(s_\ell, x]$ is at most $(x - s_\ell + 1)\mu$, by the

μ-safeness of s_ℓ. Hence, the number of bad indices is at least $(x - s_\ell) - \mu(x - s_\ell + 1) = (1-\mu)(x - s_\ell) - \mu$. Let $G_{\leq x}$ and $B_{\leq x}$ denote the number of good/bad indices in $(s_\ell, x]$. By simple algebra,

$$G_{\leq x} \leq \frac{\mu}{1-\mu}(B_{\leq x} + \mu) + \mu \leq 2\mu B_{\leq x} + 2\mu$$

Applying an analogous argument to the interval $[y, s_{\ell+1})$, we deduce that $G_{\leq x} + G_{\geq y} \leq 2\mu(B_{\leq x} + B_{\geq y}) + 4\mu$. By Theorem 7.1.6, the total number of bad indices is at most $\varepsilon_I|I| + \mu^2|I|/24$. The total number of rejected indices is at most the number of bad indices plus $G_{\leq x} + G_{\geq y}$. Thus, this is at most $(1 + 2\mu)\varepsilon_I|I| + \mu^2|I|/24 + 4\mu$. By Claim 7.1.5, $\varepsilon_I \geq \mu/24$, so the number of rejected indices is at most $(1 + 3\mu)\varepsilon_I|I| + 4\mu$. $\square$

Theorem 7.1.8 *The total number of rejected indices is in the range* $[\varepsilon_f n, (1+5\mu)\varepsilon_f n]$.

Proof The lower bound follows because the accepted indices form an increasing sequence. For the upper bound, we sum the bound of Lemma 7.1.7 over all non-empty intervals $I_i = (s_i, s_{i+1})$. Let $\mathcal{S}$ be the set of these intervals. The total number of rejected indices is at most $(1+3\mu)\sum_{I\in\mathcal{S}} \varepsilon_I|I| + 2\mu|\mathcal{S}|$. Since all intervals in $\mathcal{S}$ are disjoint, $\sum_{I\in\mathcal{S}} \varepsilon_I|I| \leq \varepsilon_f n$. Any increasing sequence in $\mathcal{F}$ loses at least one index in each $I \in \mathcal{S}$. This is because (by Claim 7.1.5), $\varepsilon_I = \Omega(\mu) > 0$. Thus, $|\mathcal{S}| \leq \varepsilon_f n$. The proof is completed by plugging in these bounds. $\square$

Theorem 7.1.9 *There is an algorithm with running time* $(1/\mu)^{O(1/\mu^2)}\varepsilon_f^{-1}\log^{c'} n$ *that given any* μ *outputs a value* $\varepsilon \in [(1-\mu)\varepsilon_f, (1+\mu)\varepsilon_f]$

Proof By Theorem 7.1.8, it suffices to estimate the number of rejected indices. For any index, the time required to run $\mathbf{DMI}_1$ is $(1/\mu)^{O(1/\mu^2)}\log^c n$. By a standard multiplicative Chernoff bound, it suffices to determine the labels on $\Theta(\mu^{-2}\varepsilon^{-1}\log n)$ uniform random indices to estimate the number of rejected indices with multiplicative $(1 \pm \mu)$ error. We rescale μ to get the final bound. $\square$

7.2 Modifications

In this section, we describe the modifications that we make to the LIS algorithm given in section 7.1 (as mentioned in section 5.4), and show that these modifications do not increase the error of the algorithm.

We begin with the following proposition.

Proposition 7.2.1 *Suppose f_X and f_Y are permutations and $f = f_Y^{-1} \circ f_X$. Then the common subsequences of f_X and f_Y correspond to increasing sequences in f and vice versa. In particular, $\texttt{lcs}_{f_X,f_Y} = \texttt{lis}_f$.*

Proof If we look at a common subsequence of f_X and f_Y, we can view it as a map from a subset of X-indices to a subset of Y-indices. In particular, an X-index x in a common subsequence is mapped to the Y-index $f(x)$. Furthermore, if we define $x_1 < x_2 < ... < x_k$ to be the X-indices of such a common subsequence, then $f(x_1) < f(x_2) < \ldots < f(x_k)$, i.e. these X-indices form an increasing subsequence in f. On the other hand, if we let $x_1 < x_2 < \ldots < x_k$ be the indices of an increasing subsequence of f, then $f(x_1) < f(x_2) < \ldots < f(x_k)$, so if we let y_i be the Y-index given by $f(x_i)$, $y_1 < y_2 < \ldots < y_k$, i.e. the pairs (x_i, y_i) for $i \in [k]$ form a common subsequence of f_X and f_Y. From this, the statement $\texttt{lcs}_{f_X,f_Y} = \texttt{lis}_f$ follows trivially. $\square$

As this proposition shows, for a box $\mathcal{B}$, any common subsequence of f_X and f_Y in $\mathcal{B}$ corresponds to an increasing sequence of f lying in the corresponding box (which we will also refer to as $\mathcal{B}$ as the intervals $X(\mathcal{B})$ and $Y(\mathcal{B})$ are identical in the two cases). Therefore, in order to find the number of points lying on the longest common subsequence of $\mathcal{B}$, we can instead find the length of the longest increasing sequence of f lying in $\mathcal{B}$. As shown in section 7.1, we have an algorithm ($\mathbf{DMI}_1$) which, given an index x, an input function f and a box $\mathcal{B}$, attempts to classify x depending on whether or not it lies on the LIS of $\mathcal{B}$. We can simulate this algorithm in the LCS setting as follows: Given input sequences f_X and f_Y with input box $\mathcal{B}$, run $\mathbf{DMI}_1$ on index x and box $\mathcal{B}$ with the function $f = f_Y^{-1} \circ f_X$. Whenever $\mathbf{DMI}_1$ attempts to query $f(x')$ for any index x', simulate this query by querying $f_X(x')$ as well as $f_Y(y)$ for each $y \in Y(\mathcal{B})$, and returning the value y such that $f_Y(y) = f_X(x')$. This simulation of $\mathbf{DMI}_1$ will approximate `lis`$(\mathcal{B})$ and therefore `lcs`$(\mathcal{B})$ to within the same level of accuracy as in the LIS setting, and its running time is $h(\mathcal{B})$ times the running time of $\mathbf{DMI}_1$ in the LIS setting, since each query which could be done in unit time in the LIS setting now takes $O(h(\mathcal{B}))$ time in the LCS setting.

Our goal is to reduce the $h(\mathcal{B})$ blowup in cost incurred by simulating queries in the LCS setting. Looking more closely at the queries made by $\mathbf{DMI}_1$, we observe that whenever such queries are made, the indices x' for which $f(x')$ is queried are obtained randomly from the X-interval of a specified box (call this box $\mathcal{T}$). As a result, in order to simulate this query, it suffices to generate a match uniformly at random from the matches of the elements of $X(\mathcal{T})$. Another way of saying this is that we would like to find a random point in the box $X(\mathcal{T}) \times Y(\mathcal{B})$ ($\mathcal{B}$ being the input box). To do so, we can sample $O(\sqrt{h(\mathcal{B})})$ Y-indices from $Y(\mathcal{B})$ and $O(\sqrt{h(\mathcal{B})})$ X-indices from $X(\mathcal{T})$, querying the values of f_Y and f_X for the indices in these two sets respectively. By a birthday paradox argument, with high probability, $\exists y \in Y(\mathcal{B}), x \in X(\mathcal{T})$ s.t. $f_Y(y) = f_X(x)$, provided that $w(\mathcal{T}) = \Omega(\sqrt{h(\mathcal{B})})$.

Our goal now is to ensure that any box from which we would like to query random points has width at least $\sqrt{h(\mathcal{B})}$. To do this, we look more closely at the $\mathbf{DMI}_1$ algorithm. The $\mathbf{DMI}_1$ algorithm is a modification of the [21] algorithm, which is fairly complicated, so we will introduce only the aspects of the algorithm that will be necessary for us to analyze. The approach of $\mathbf{DMI}_1$ is to construct a rooted tree of boxes. In actuality, a call to $\mathbf{DMI}_1$ for an index x and a box $\mathcal{B}$ only traverses one root to leaf path in such a tree, however from an analytical standpoint, for a fixed setting of the random bits, if we consider calls $\mathbf{DMI}_1(x, \mathcal{B})$ for each $x \in X(\mathcal{B})$, the boxes encountered by these calls form a rooted tree of boxes. This tree has the full box $\mathcal{B}$ at the root, and for any box $\mathcal{T}$ in the tree, its children form a box chain spanning $\mathcal{T}$. The leaves of this tree consist of two types of boxes:

(1) Boxes $\mathcal{T}$ on which the algorithm attempts to find an LIS of $\mathcal{T}$.

(2) Boxes $\mathcal{T}$ on which the algorithm "gives up", i.e. classifies all indices in $X(\mathcal{T})$ as BAD.

We will refer to boxes of type (1) as *green boxes* and boxes of type (2) as *red boxes.* In the case of the algorithm $\mathbf{DMI}_1$, all green boxes have width 1. As a result, for $\mathbf{DMI}_1$, the task of finding an LIS of a green box is trivial, as it consists solely of determining whether or not the match of the X-index lies in the Y-interval of the box.

Since the leaves of any such tree form a box chain, the concatenation of the LIS's of the green leaves is itself an increasing sequence in $\mathcal{B}$, so its length is at most $\texttt{lis}(\mathcal{B})$ (this holds even when the green leaves do not necessarily have width 1). As a result, the accuracy of such an estimation procedure is determined by the guarantee we can prove for the length of this concatenation (the longer the better).

For such a tree G, define the *ω-truncation* of G to be the tree obtained by removing the children of every node $\mathcal{T}$ with $w(\mathcal{T}) \leq \omega$ and coloring $\mathcal{T}$ green. If G is the tree given by $\mathbf{DMI}_1$, let $\mathbf{DMI}_2$ be the algorithm that simulates $\mathbf{DMI}_1$ with the modification that, whenever it reaches a leaf $\mathcal{T}$ in the ω-truncation of G (for a fixed value of ω which we will choose later), it computes $\texttt{lis}(\mathcal{T})$ exactly, labeling an index of $X(\mathcal{T})$ GOOD iff it lies on this LIS. Since the union of the LIS's of all green leaves of G which are subboxes of $\mathcal{T}$ is an increasing sequence of $\mathcal{T}$, replacing this union with an LIS of $\mathcal{T}$ can only increase the number of indices labeled GOOD. Therefore, regardless of the choice of ω, $\mathbf{DMI}_2$ will classify at least as many points GOOD as $\mathbf{DMI}_1$, so the estimate given by $\mathbf{DMI}_2$ is at least as accurate as the estimate given by $\mathbf{DMI}_1$. We state this in the following Lemma:

Lemma 7.2.2 *Suppose that, for a box $\mathcal{B}$ and $\delta > 0$, the number of indices $x \in X(\mathcal{B})$ classified as BAD by $\mathbf{DMI}_1$ is at most $(1+\delta)\texttt{Xloss}(\mathcal{B})$ with high probability. Then the number of indices $x \in X(\mathcal{B})$ classified as BAD by $\mathbf{DMI}_2$ is at most $(1+\delta)\texttt{Xloss}(\mathcal{B})$ with high probability.*

Next, we attempt to simulate the task of querying values of $f(x)$ for randomly chosen X-indices $x \in X(\mathcal{B})$. Now it may be the case that $f(x) \notin Y(\mathcal{B})$, in which case $\mathbf{DMI}_2$ ignores the query. As a result, we can simulate such queries by finding random matches in $\mathcal{B}$. The full primitive of $\mathbf{DMI}_2$ that we simulate consists of obtaining a list of such queries. This primitive in $\mathbf{DMI}_2$ is carried out by repeating the task of querying a random X-index in $X(\mathcal{B})$ m times, with repetitions. We do the same, finding a random match m times with repetitions. The only issue we run into with this is that, since we do not know the number of points in $\mathcal{B}$ a priori, we cannot be certain how many of these m queries will successfully find a match in $\mathbf{DMI}_2$. As a result, we construct a simulation which (with high probability) is guaranteed to find at least as many matches in $\mathcal{B}$.

In section 7.3, we build a procedure **BoxSample**, which achieves these properties. We will prove the following lemma (For simplicity, we let $h = h(\mathcal{B}), w = w(\mathcal{B}), d_m = d_{min}(\mathcal{B}), d_M = d_{max}(\mathcal{B})$):

Lemma 7.2.3 *Let $\mathcal{B}$ be a box such that $h \geq 4$, $w \geq 2\sqrt{h}$. Suppose $\mathcal{B}$ contains δw matches. The following statements hold:*

1. *If $\delta \geq 1/4$, then with probability at least $1 - e^{-\Omega(m)}$,* **BoxSample**$(m, \mathcal{B})$ *returns a sample of size at least m.*
2. *If $\delta < 1/4$, then the number of points returned by* **BoxSample**$(m, \mathcal{B})$ *is bounded below by a random variable $Z \sim Bin(m, \delta)$.*
3. *With probability at least $1 - e^{O(\log m) - \Omega(\sqrt{h})}$,* **BoxSample**$(m, \mathcal{B})$ *makes $O(m\sqrt{h})$ queries.*

We say that, for random variables Z_1 and Z_2, Z_2 *dominates* Z_1 with failure κ if $\forall k \geq 0$, $Pr[Z_2 \geq k] \geq Pr[Z_1 \geq k] - \kappa$. As this lemma shows, the random variable representing the number of points returned by **BoxSample**$(m, \mathcal{B})$ dominates the random variable representing the number of points returned by the sampling procedure of $\mathbf{DMI}_2$ (with parameters $m, \mathcal{B}$) with failure $e^{-\Omega(m)}$. Since every call to this sampling procedure has $m = \Omega(\log^2 n)$, this failure is at most $n^{-\Omega(\log n)}$. Therefore, since the entire algorithm makes at most polylogarithmic calls to this sampling procedure, by a union bound the probability of failure of any of these sampling calls is at most $n^{-\Omega(\log n)}$. As a result, in order to show that we can replace the sampling procedure of $\mathbf{DMI}_2$ with **BoxSample** without incurring additional error (with high probability), it remains to show that increasing the number of points returned by the sampling procedure does not increase the error of the algorithm. To show this, we will unfortunately have to look more closely at some of the details of the algorithm **DMI**$_2$ (which are identical to the corresponding details of the [21] algorithm).

First, we look at the parts of the [21] algorithm which use random sampling of indices. We find that random sampling is used in the following subprocedures:

1. **ApproxLIS**, which takes a random sample of points and uses the fraction of bad points in the sample as an approximation of the total number of bad points.
2. **FindSplitter**, which attempts to find a good splitter in a random sample of points, or declares that few good splitters exist if it fails to find one.
3. **BuildGrid**, which uses the random sample to be able to build an α-fine $\mathcal{B}$-grid.
4. **approxZ**, which given a candidate splitter P and a box $\mathcal{B}$ uses a random sample of points in $\mathcal{B}$ to estimate the fraction of points in $\mathcal{B}$ which are in violation with P.

When analyzing **ApproxLIS**, [21] shows that the random sample obtained gives a good approximation to the fraction of bad points using the following Chernoff Bound:

Proposition 7.2.4 *Let I be an X-interval and $\gamma \in [0,1]$ and $s \in \mathbb{N}^+$. Let $A \subseteq I$ be fixed. For a random sample $x_1, \ldots, x_s$ from I, let r denote the fraction of points that belong to A. Then $Pr[|r - \frac{|A|}{|I|}| \geq \gamma] \leq 2e^{-2\gamma^2 s}$.*

The number of sample points is given by s in this lemma. If we increase s, $2e^{-2\gamma^2 s}$ decreases, meaning the probability that our estimate is not a good approximation decreases as we increase the size of the sample. Therefore, such an increase in the size of the sample does not increase the error incurred from **ApproxLIS**.

When analyzing **FindSplitter**, [21] starts by supposing that, for the given box $\mathcal{B}$ and a parameter $0 \leq \rho \leq 1$, $\rho w(\mathcal{B})$ of the X-indices in $\mathcal{B}$ are the X-indices of good splitters in $\mathcal{B}$. From this, [21] argues that the probability that a random sample of size m fails to find any such good splitter is $(1-\rho)^m$. m is chosen so that this quantity is sufficiently small. As in the **ApproxLIS** case, increasing m will not increase the value of this quantity (since $0 \le (1-\rho) \le 1$), so increasing the size of the sample does not increase the error incurred from **FindSplitter**.

For the case of **BuildGrid**, a similar argument could be used to show that our sampling procedure will suffice, however as we will mention later, it is simpler to replace **BuildGrid** with a procedure that does not require random sampling in our setting. As a result, we do not have to worry about the random sampling in the procedure **BuildGrid**.

Lastly, we look at the procedure **approxZ**. In [21], the probability of error of **approxZ** is bounded using a Hoeffding bound. When the Hoeffding bound is applied, the expression for the probability of error is $\exp(-\Omega(L^2 m/w(\mathcal{B})^2))$, where L is an error threshold parameter and m represents the size of the sample. Again, we see that increasing m (while holding L constant) will decrease the probability of error.

As the above discussion shows, in every instance in $\mathbf{DMI}_2$ where the sampling primitive is used, increasing the size of the sample does not increase the probability of error. Therefore, if we let $\mathbf{DMI}_3$ be the modification of $\mathbf{DMI}_2$ obtained by replacing this sampling primitive with **BoxSample**, $\mathbf{DMI}_3$ will achieve at least the same level of accuracy as $\mathbf{DMI}_2$. We state this in the following lemma, adopting the convention that any X-index x with $f(x) \notin Y(\mathcal{B})$ is automatically classified as BAD by $\mathbf{DMI}_3$ (since $\mathbf{DMI}_3$ only takes as input points in $\mathcal{B}$).

Lemma 7.2.5 *Suppose that, for a box $\mathcal{B}$ and $\delta > 0$, the number of indices $x \in X(\mathcal{B})$ classified as BAD by $\mathbf{DMI}_2$ is at most $(1+\delta)\texttt{Xloss}(\mathcal{B})$ with high probability. Then the number of indices $x \in X(\mathcal{B})$ classified as BAD by $\mathbf{DMI}_3$ is at most $(1+\delta)\texttt{Xloss}(\mathcal{B})$ with high probability.*

As mentioned above, the procedure **BuildGrid** in $\mathbf{DMI}_1$ also uses queries $f(x)$ for random indices $x \in X(\mathcal{B})$, but it is easier to simulate without random sampling in the LCS setting. It is not necessary for us to implement this procedure without random sampling, but given how simple the modified procedure is, we choose to implement this modification. Since this simplification is not strictly necessary, we will defer the details to section 7.7. The procedure **MakeGrid** which we define achieves the same properties as **BuildGrid**, so if $\mathbf{DMI}_4$ is the procedure obtained by replacing **BuildGrid** with **MakeGrid** in $\mathbf{DMI}_3$, $\mathbf{DMI}_4$ achieves the same accuracy guarantees as $\mathbf{DMI}_3$.

Lemma 7.2.6 *Suppose that, for a box $\mathcal{B}$ and $\delta > 0$, the number of indices $x \in X(\mathcal{B})$ classified as BAD by $\mathbf{DMI}_3$ is at most $(1+\delta)\mathtt{Xloss}(\mathcal{B})$ with high probability. Then the number of indices $x \in X(\mathcal{B})$ classified as BAD by $\mathbf{DMI}_4$ is at most $(1+\delta)\mathtt{Xloss}(\mathcal{B})$ with high probability.*

7.3 A Sampling Procedure

In this section, we state the procedure **BoxSample** described in section 7.2, and prove the associated lemma given in that same section.

Given a box $\mathcal{B}$ and a parameter m, we will want to simulate the process of choosing m random indices from $\mathcal{B}$ and finding all matches of chosen indices which lie in $\mathcal{B}$. As mentioned earlier, we do this using a birthday paradox technique. (For simplicity, we let $h = h(\mathcal{B}), w = w(\mathcal{B}), d_m = d_{min}(\mathcal{B}), d_M = d_{max}(\mathcal{B})$).

We define the procedure **BoxSample**$(m, \mathcal{B})$, which takes as input a box $\mathcal{B}$ and a parameter m, and makes m calls to **PointSample**$(\mathcal{B})$. **PointSample**$(\mathcal{B})$ attempts to generate a match in $\mathcal{B}$ uniformly at random among all matches in $\mathcal{B}$, and has the property that, if $\mathcal{B}$ contains δw matches, then **PointSample**$(\mathcal{B})$ successfully finds a match with sufficiently high probability as a function of δ (assuming $\mathcal{B}$ is not too unbalanced).

```
BoxSample($m, \mathcal{B}$)
Output: A list of matches in $\mathcal{B}$.
1. Run PointSample($\mathcal{B}$) $3m$ times.
2. Output the list of matches returned by the calls to PointSample($\mathcal{B}$).
```

```
PointSample(B)
Output: A match in $\mathcal{B}$ or FAIL.
1. Set $p = \sqrt{\frac{4h}{w^2}}, q = \sqrt{\frac{4}{h}}$. If $p > 1$ or $q > 1$, output FAIL.
2. Otherwise, let $S_x = \textbf{Sample}(X(\mathcal{B}), p)$, $S_y = \textbf{Sample}(Y(\mathcal{B}), q)$.
3. Set $S = \{(x, y) : x \in S_x, y \in S_y, y = f(x)\}$.
4. If $S$ is nonempty, return a uniformly random element of $S$, else return FAIL.
```

The following lemma states the guarantees that we'll need for **PointSample**.

Lemma 7.3.1 *Let $\mathcal{B}$ be a box such that $h \geq 4$, $w \geq 2\sqrt{h}$. Suppose $\mathcal{B}$ contains δw matches. For a call* **PointSample**$(\mathcal{B})$*, the following statements hold:*

1. *Each match in $\mathcal{B}$ is equally likely to be the output.*
2. **PointSample**$(\mathcal{B})$ *returns a match with probability at least* $\min(2\delta, \frac{1}{2})$.
3. *With probability at least $1 - e^{-\Omega(\sqrt{h})}$,* **PointSample**$(m, \mathcal{B})$ *makes $O(\sqrt{h})$ queries.*

To prove Lemma 7.3.1, we will use the following claim.

Claim 7.3.2 *For nonnegative real numbers k, n with $n \geq 2$, $nk \leq 1$, $(1 - k)^n \leq (1 - nk + \frac{n(n-1)}{2}k^2)$*

Proof By the generalized binomial theorem, $(1 - k)^n = (1 - nk + \binom{n}{2}k^2 - \binom{n}{3}k^3 + \cdots)$. Each successive term is at most nk times the previous term. Since $nk \leq 1$, this alternating series telescopes, meaning it can be bounded by its partial sums (bounded above when the final term is positive, or below when the final term is negative). Using the definition of the generalized binomial coefficient $\binom{n}{2}$, the result follows. □

We now prove Lemma 7.3.1.

Proof For the first item, a given match is in $S_x \times S_y$ iff its X-index is in S_x (which happens with probability p) and its Y-index is in S_y (which happens with probability q), independently of the other matches. Therefore, each match of $\mathcal{B}$ is in $S_x \times S_y$ independently with probability pq. Since the output of **PointSample**$(\mathcal{B})$ is a uniformly chosen match in $S_x \times S_y$, the statement follows.

For the second item, suppose first that $\delta \leq 1/4$. As stated above, each match is in $S_x \times S_y$ independently with probability $pq = \frac{4}{w}$. Therefore, the probability that none of the δw matches in $\mathcal{B}$ lie in $S_x \times S_y$ is $(1 - \frac{4}{w})^{\delta w}$. Since $\delta \leq 1/4$, $\delta w \frac{4}{w} \leq 1$. Therefore, by Claim 7.3.2,

$$\begin{aligned}(1 - \frac{4}{w})^{\delta w} &\leq 1 - 4\delta + \frac{\delta w(\delta w - 1)}{2}(\frac{4}{w})^2 \\ &\leq 1 - 4\delta + \frac{1}{2}(\delta w)^2(\frac{4}{w})^2 \\ &\leq 1 - 4\delta + \frac{1}{2}(4\delta)^2 \\ &\leq 1 - 4\delta + \frac{1}{2}(4\delta) \\ &\leq 1 - 2\delta\end{aligned}$$

Since no matches are found with probability at most $1 - 2\delta$, the statement follows.

Suppose instead that $\delta \geq 1/4$. In this case, note that the above bound holds when using $\delta = \frac{1}{4}$. Since $(1 - \frac{4}{w})^{\delta w}$ only gets smaller as δw increases, the bound of $1 - 2\frac{1}{4} = 1/2$ holds for $\delta \geq 1/4$, so again the statement follows.

For the third item, let Z be a random variable representing the size of S_x. We have $Z \sim Bin(w, \sqrt{\frac{4h}{w^2}})$. Note that $\mu = \sqrt{4h}$. We have

$$\begin{aligned}Pr[Z > 3/2\sqrt{4h}] &= Pr[Z > (1 + 1/2)\sqrt{4h}] \\ &\leq \exp(-\frac{(1/2)^2\sqrt{4h}}{3}) \\ &\leq \exp(-\frac{\sqrt{4h}}{12}).\end{aligned}$$

An analogous argument achieves the same bound on the size of S_y, establishing the claim. □

The reader should note that in every call to **PointSample** in our procedure, $h \geq \Omega(\log^2(n))$, so the probability that the third item fails is at most $n^{-\Omega(\log(n))}$.

We now prove Lemma 7.2.3, which states guarantees for the output and running time of **BoxSample**$(m, \mathcal{B})$, provided $h \geq 4$, $w \geq 2\sqrt{h}$.

Proof For the first item, assume $\delta \geq 1/4$. By Lemma 7.3.1, **PointSample**$(\mathcal{B})$ returns a match with probability at least $1/2$. Therefore, the number of matches returned by

BoxSample$(m, \mathcal{B})$ is bounded below by a random variable $W \sim Bin(3m, 1/2)$. By the Chernoff Bound,

$$\begin{aligned} Pr[W < m] &\le Pr[W < (1 - 1/3)\frac{3m}{2}] \\ &\le \exp(-\frac{(1/3)^2 \frac{3m}{2}}{2}) \\ &\le \exp(-\frac{m}{12}) \end{aligned}$$

For the second item, suppose $\delta < 1/4$. By Lemma 7.3.1, **PointSample**$(\mathcal{B})$ returns a match with probability at least 2δ. Therefore, the number of matches returned by **BoxSample**$(m, \mathcal{B})$ is bounded below by a random variable $W \sim Bin(3m, 2\delta)$, which in turn is bounded below by the random variable $Bin(m, \delta)$.

For the third item, by Lemma 7.3.1, each call to **PointSample**$(\mathcal{B})$ makes $O(\sqrt{h})$ queries independently with probability $1 - e^{-\Omega(\sqrt{h})}$. By a union bound, the probability that all of these calls make $O(\sqrt{h})$ queries is at least $1 - 3me^{-\Omega(\sqrt{h})}$. Since the total number of queries made by **BoxSample**$(m, \mathcal{B})$ is just the sum of the number of queries made by each call to **PointSample**$(\mathcal{B})$, the result follows. □

7.4 FindLCS

In section 7.2, several modifications to **DMI**$_1$ were introduced, yielding the procedure **DMI**$_4$. The first modification involved stopping the procedure once it reached a box of width at most ω and computing the LIS of the box exactly. As shown earlier, there is a correspondence between increasing sequences and common subsequences. (Here we can revert back to the LCS setting, using an algorithm which computes the LCS of a box so that we can use it as a black box without having to worry about simulating LIS queries in the LCS setting.) In most cases, the algorithm of [14] can do this in time $O(\omega \log \omega)$, which is acceptable for us. However, if the input box $\mathcal{T}$ has height significantly larger than ω, the algorithm of [14] will not have the same guarantee. As a result, we use the algorithm of [14] to construct an algorithm whose running time can be bounded by $O(\omega \log \omega)$, which gives us an acceptable approximation to `lis`$(\mathcal{T})$. We call our procedure **FindLCS**, which is described in the following paragraph.

For a box $\mathcal{T}$ and a parameter λ, let $\texttt{BOT}_\lambda(\mathcal{T}) = X(\mathcal{T}) \times [y_B(\mathcal{T}), \min(y_T(\mathcal{T}), y_B(\mathcal{T}) + w(\mathcal{T})/\lambda)])$ and $\texttt{TOP}_\lambda(\mathcal{T}) = X(\mathcal{T}) \times [\max(y_B(\mathcal{T}), y_T(\mathcal{T}) - w(\mathcal{T})/\lambda), y_T(\mathcal{T})])$. **FindLCS**$(\lambda, \mathcal{T})$ runs the algorithm of [14] on $\texttt{BOT}_\lambda(\mathcal{T})$ and $\texttt{TOP}_\lambda(\mathcal{T})$, and returns the longer of the two output sequences (if they are the same length, it chooses the first one). Note that, if $\mathcal{T}$ is λ-proportional, $\mathcal{T} = \texttt{BOT}_\lambda(\mathcal{T}) = \texttt{TOP}_\lambda(\mathcal{T})$, so **FindLCS**$(\mathcal{T}, \lambda)$ just returns the output of the algorithm of [14] run on $\mathcal{T}$.

It is routine to show that this procedure runs in the desired time bound; we state and prove this here.

Proposition 7.4.1 *Given an input of two nonrepeating sequences and a subbox $\mathcal{T}$ of the input box with $w(\mathcal{T}) \leq \omega$,* **FindLCS**$(\lambda, \mathcal{T})$ *runs in time $O(k \log k)$, where $k = \omega/\lambda$.*

Proof Note that **FindLCS** consists of two calls to the LCS algorithm of [14], which are run on ω-small, λ-proportional boxes. These two properties imply $d_{max}(\mathcal{T}) \leq k$. As a result, by Theorem 1 of [14], the given LCS algorithm runs in time $O(k \log k)$. (Note that since the input sequences were nonrepeating, $\mathcal{T}$ contains at most k character matches.) □

It will also be necessary to prove an accuracy guarantee for **FindLCS**. We defer this to a later section, as the argument and guarantee are both fairly technical. First, we will discuss two example input instances which we hope will provide some intuition as to why **FindLCS**, as well as the algorithm as a whole, works.

7.5 Examples

In this section, we give two examples which aim to provide some intuition as to how our algorithm handles certain potential difficulties.

For the first example, suppose the input consists of two identical permutations, so the point-box representation of the input is an increasing sequence of points along the main diagonal of the input box $\mathcal{B}$ (see Fig. 7.1). When our algorithm reaches the procedure $\mathbf{XLI}_2$, it chooses a random box $\mathcal{T}$, on which the procedure $\mathbf{XLI}_0$ will ultimately be run. As described in section 6.2, we can think of this random choice of $\mathcal{T}$

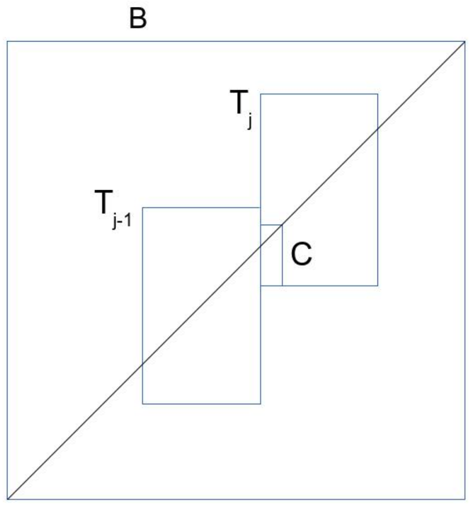

Figure 7.1: Input 1

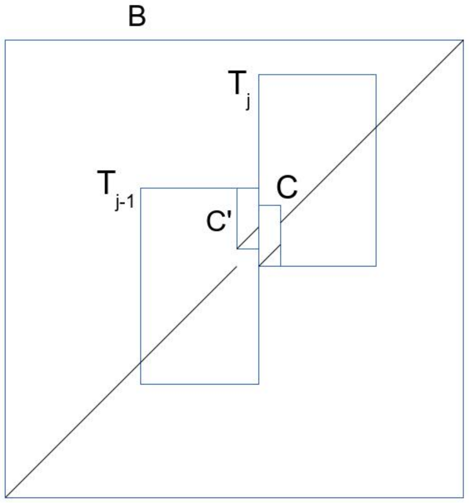

Figure 7.2: Input 2

as chosen by choosing a random partition of X-intervals of size r, and then randomly choosing one of these X-intervals (along with the appropriate Y-interval). As viewed this way, we get a sequence of boxes $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_k$. Let $\mathcal{T}_{j-1}, \mathcal{T}_j$ be two consecutive boxes in this sequence (as shown in Fig. 7.1). Suppose box $\mathcal{T}_j$ is chosen by $\mathbf{XLI}_2$. When $\mathbf{XLI}_0$ is run on this box, any such call will eventually reach a box of width $O(\sqrt{r})$, at which point **FindLCS** will be called. In most cases, **FindLCS** will easily find the LCS of the box. However, the leftmost and rightmost of these boxes will have height $O(\sqrt{r} + a') = O(r)$ (box $\mathcal{C}$ in Fig. 7.1). This box is too unbalanced for the algorithm of [14] to be efficiently run on $\mathcal{C}$. In particular, such a call would not necessarily run in $o(r)$ time. The algorithm **FindLCS** instead calls this algorithm on the (balanced) top and bottom ends of $\mathcal{C}$. In this case, **FindLCS** will find the entire LCS of $\mathcal{C}$ in the top end, successfully classifying all indices as GOOD. Note that for this input, since the ulam distance is 0, our algorithm cannot afford to make any errors, so it would not have been acceptable to just throw out this box.

The reader might now ask what would happen if the LCS did not lie entirely in the top of $\mathcal{C}$. In this case, **FindLCS** may not necessarily find an LCS of $\mathcal{C}$. However, the fact that the LCS contains points outside the top of $\mathcal{C}$ means that the Y range of the LCS is much larger than the X range of the LCS, meaning that there must be many

Y-indices between the bottom point of this LCS and the top point of this LCS that are unmatched in $\mathcal{C}$. If the LCS of the input box does in fact contain this sequence, then it must miss all of these Y-indices, whose number is large enough that the entire width of $\mathcal{C}$ is small compared to it, meaning **FindLCS** could output a value as small as 0 for $\mathcal{C}$. On the other hand, if the LCS of the input box does not contain the LCS of $\mathcal{C}$, then we merely have to look at the intersection of the LCS of the input box with $\mathcal{C}$. If this intersection contains points outside the top of $\mathcal{C}$, then this same argument applies. If not, then **FindLCS** will find all the points in this intersection.

This first example illustrates how our algorithm might be able to find most of the points on the LCS of the input box, but it should not be clear that our algorithm does not classify too many points as GOOD. In other words, the points classified as GOOD by our algorithm may not form a common subsequence (and in fact in some cases they are not). To illustrate, consider the input shown by Fig. 7.2. Here, the ulam distance is $2 \cdot \min(w(\mathcal{C}), w(\mathcal{C}'))$. However, since our algorithm looks at the LCS of each subbox individually, in this case it appears to successfully find a perfect LCS in every box, thus returning 0 as in the first example. Obviously this is unacceptable, so the reader may be puzzled why this does not invalidate the correctness of our algorithm. The key here is that, while we do return 0 when the picture looks like Fig. 7.2, for the given input it is unlikely that this will be the case. Recall that the procedure $\mathbf{XLI}_2$ chose the left endpoint of the box $\mathcal{T}_j$ uniformly at random. It turns out that the width of this random box was chosen to be big enough so that, most of the time, if the input is as in Fig. 7.2, the points in $\mathcal{C}$ and $\mathcal{C}'$ will all lie in the same $\mathcal{T}_j$. If this does happen, the algorithm will correctly classify one of these two sets of points as BAD and correctly output $2 \cdot \min(w(\mathcal{C}), w(\mathcal{C}'))$. Averaging over this random choice, we get that the expected output of the algorithm is close to $2 \cdot \min(w(\mathcal{C}), w(\mathcal{C}'))$, and more importantly, we show that with sufficiently high probability, our algorithm does return a value which is within a $(1 + \varepsilon)$ factor of this value.

Obviously one could construct a more complicated input with these potential issues occuring in several different places for several possible choices of randomness, but the same arguments can still be applied. We now conclude this discussion in the hope that these two examples provided enough insight for the reader to be more comfortable with the correctness of our algorithm.

7.6 Analysis

In this section, we analyze the LIS algorithm outlined in section 5.4, whose code appears in section 7.8. Note that we have already provided the code for some subprocedures of this algorithm, namely **BoxSample** (section 7.3) and **FindLCS** (section 7.4).

First, we analyze the accuracy of the procedure **FindLCS**. It turns out that, if a final box $\mathcal{T}$ is sufficiently unbalanced, then the procedure of [14] is not guaranteed to be able to efficiently find an LCS of $\mathcal{T}$. As a result, we formulate **FindLCS** so that this algorithm is run on subboxes of $\mathcal{T}$ which are adequately balanced. This formulation potentially introduces some error, but we show that this error is manageable, proving a bound on the amount of error introduced by **FindLCS** in Lemma 7.6.1. We remind the reader of the following parameter settings: $\lambda = \frac{\varepsilon_4}{1+\varepsilon_4}$, $\varepsilon_4 = \frac{\varepsilon_3}{10}$. We also recall the following definitions:

- $\texttt{herr}(S, \mathcal{B}) = \texttt{Xloss}(S, \mathcal{B}) - \texttt{Xloss}(\mathcal{B}) = \texttt{lcs}(\mathcal{B}) - |S|$.
- $\texttt{serr}_{\varepsilon}(S, \mathcal{B}) = \min\limits_{S' \subseteq S}(\texttt{herr}(S', \mathcal{B}) + \varepsilon \cdot \texttt{Yloss}_{\texttt{trim}}(S', \mathcal{B}))$.
- $\texttt{terr}_{\varepsilon}(\mathcal{B}) = \max\limits_{S}(\texttt{serr}_{\varepsilon}(S, \mathcal{B}))$, where the max is taken over common subsequences S of $\mathcal{B}$ with length at least $\texttt{lcs}(\mathcal{B}) - \varepsilon \cdot \texttt{Xloss}(\mathcal{B})$.
- $\widetilde{S}_{\varepsilon}(\mathcal{B})$ denotes the sequence S' achieving the minimum in $\texttt{serr}_{\varepsilon}(S, \mathcal{B})$.

Lemma 7.6.1 *Let $\mathcal{T}$ be a box, and let S be a common subsequence of matches in $\mathcal{T}$. Let $\vec{\mathcal{C}}$ be a box chain spanning $\mathcal{T}$ with $S \subseteq \vec{\mathcal{C}}$. Suppose that $|\{\mathcal{C} \in \vec{\mathcal{C}} : \widetilde{S}_{\varepsilon_3}(\mathcal{T}) \cap C \neq \varnothing\}| > 1$. Then $\sum\limits_{\mathcal{C} \in \vec{\mathcal{C}}} \texttt{Xloss}(\textbf{FindLCS}(\lambda, \mathcal{C}), \mathcal{C}) \leq (1 + \varepsilon_4)(\texttt{Xloss}(\mathcal{T}) + \texttt{serr}_{\varepsilon_3}(S, \mathcal{T}))$.*

Proof First, consider $\texttt{Yloss}_{\texttt{trim}}(\widetilde{S}_{\varepsilon_3}(\mathcal{T}), \mathcal{T})$. $\texttt{Yloss}_{\texttt{trim}}(\widetilde{S}_{\varepsilon_3}(\mathcal{T}), \mathcal{T})$ counts the number of Y-indices between the ends of $\widetilde{S}_{\varepsilon_3}(\mathcal{T})$ that do not have a match on $\widetilde{S}_{\varepsilon_3}(\mathcal{T})$. For a subbox $\mathcal{C}$, we can count the number of such indices which lie in $Y(\mathcal{C})$. Let $V_{\mathcal{T}}(\widetilde{S}_{\varepsilon_3}(\mathcal{T}), \mathcal{C})$

denote this value. Note that for $\vec{\mathcal{C}}$ a box chain spanning $\mathcal{T}$, $\sum_{\mathcal{C}\in\vec{\mathcal{C}}} V_{\mathcal{T}}(\widetilde{S}_{\varepsilon_3}(\mathcal{T}),\mathcal{C}) = \texttt{Yloss}_{\texttt{trim}}(\widetilde{S}_{\varepsilon_3}(\mathcal{T}),\mathcal{T})$.

Let $\mathcal{C}\in\vec{\mathcal{C}}$. Suppose that $\mathcal{C}$ is λ-proportional. In this case, **FindLCS**$(\lambda,\mathcal{C})$ outputs an LCS of $\mathcal{C}$, so $\texttt{Xloss}(\textbf{FindLCS}(\lambda,\mathcal{C}),\mathcal{C}) = \texttt{Xloss}(\mathcal{C})$.

Now suppose that $\mathcal{C}$ is λ-fat. In this case,
$\texttt{lcs}(\mathcal{C}) \leq h(\mathcal{C}) \leq \lambda w(\mathcal{C}) \leq \frac{\varepsilon_4}{1+\varepsilon_4}(\texttt{Xloss}(\mathcal{C}) + \texttt{lcs}(\mathcal{C}))$, so $\texttt{lcs}(\mathcal{C}) \leq \varepsilon_4\texttt{Xloss}(\mathcal{C})$ and $\texttt{Xloss}(\textbf{FindLCS}(\lambda,\mathcal{C}),\mathcal{C}) \leq w(\mathcal{C}) \leq (1+\varepsilon_4)\texttt{Xloss}(\mathcal{C})$.

Lastly, suppose that $\mathcal{C}$ is λ-skinny. If $|\widetilde{S}_{\varepsilon_3}(\mathcal{T})\cap\mathcal{C}| \leq |\textbf{FindLCS}(\lambda,\mathcal{C})|$, then $\texttt{Xloss}(\textbf{FindLCS}(\lambda,\mathcal{C}),\mathcal{C}) \leq \texttt{Xloss}(\widetilde{S}_{\varepsilon_3}(\mathcal{T})\cap\mathcal{C},\mathcal{C})$. Now suppose $|\widetilde{S}_{\varepsilon_3}(\mathcal{T})\cap\mathcal{C}| > |\textbf{FindLCS}(\lambda,\mathcal{C})|$. Recall $\texttt{BOT}_\lambda(\mathcal{C}) = X(\mathcal{C})\times[y_B(\mathcal{C}), y_B(\mathcal{C}) + w(\mathcal{C})/\lambda])$ and $\texttt{TOP}_\lambda(\mathcal{C}) = X(\mathcal{C})\times[y_T(\mathcal{C}) - w(\mathcal{C})/\lambda, y_T(\mathcal{C})])$. Since **FindLCS**$(\lambda,\mathcal{C})$ outputs the longest common subsequence found in either $\texttt{BOT}(\mathcal{C})$ or $\texttt{TOP}(\mathcal{C})$, $\exists s_1, s_2 \in \widetilde{S}_{\varepsilon_3}(\mathcal{T})\cap\mathcal{C}$ such that $s_1 \notin \texttt{BOT}(\mathcal{C}), s_2 \notin \texttt{TOP}(\mathcal{C})$. By assumption, $\exists\mathcal{C}' \neq \mathcal{C}$ such that $\widetilde{S}_{\varepsilon_3}(\mathcal{T})\cap\mathcal{C}' \neq \varnothing$. If $\mathcal{C}'$ lies below $\mathcal{C}$, then all unmatched Y-indices between $y_B(\mathcal{C})$ and s_1 contribute to $\texttt{Yloss}_{\texttt{trim}}(\widetilde{S}_{\varepsilon_3}(\mathcal{T}),\mathcal{T})$. If $\mathcal{C}'$ lies above $\mathcal{C}$, then all unmatched Y-indices between $y_T(\mathcal{C})$ and s_2 contribute to $\texttt{Yloss}_{\texttt{trim}}(\widetilde{S}_{\varepsilon_3}(\mathcal{T}),\mathcal{T})$. In either case, there are at least $\frac{w(\mathcal{C})}{\lambda}$ Y-indices in $Y(\mathcal{C})$ between two points on $\widetilde{S}_{\varepsilon_3}(\mathcal{T})$, of which at most $w(\mathcal{C})$ of them can have matches on $\widetilde{S}_{\varepsilon_3}(\mathcal{T})$. Therefore, there are at least $\frac{w(\mathcal{C})}{\lambda} - w(\mathcal{C}) = \frac{w(\mathcal{C})}{\varepsilon_4}$ Y-indices in $Y(\mathcal{C})$ which contribute to $\texttt{Yloss}_{\texttt{trim}}(\widetilde{S}_{\varepsilon_3}(\mathcal{T}),\mathcal{T})$. Since $|\widetilde{S}_{\varepsilon_3}(\mathcal{T})\cap\mathcal{C}| - |\textbf{FindLCS}(\lambda,\mathcal{C})| \leq w(\mathcal{C})$, this is at most $\varepsilon_4 V_{\mathcal{T}}(\widetilde{S}_{\varepsilon_3}(\mathcal{T}),\mathcal{C}) \leq \varepsilon_3 V_{\mathcal{T}}(\widetilde{S}_{\varepsilon_3}(\mathcal{T}),\mathcal{C})$.

Putting all of these cases together,

$$\begin{aligned}\sum_{\mathcal{C}\in\vec{\mathcal{C}}} \texttt{Xloss}(\textbf{FindLCS}(\lambda,\mathcal{C}),\mathcal{C}) &\leq \sum_{\mathcal{C}\in\vec{\mathcal{C}}} (1+\varepsilon_4)\texttt{Xloss}(\widetilde{S}_{\varepsilon_3}(\mathcal{T})\cap\mathcal{C},\mathcal{C}) + \varepsilon_3 V_{\mathcal{T}}(\widetilde{S}_{\varepsilon_3}(\mathcal{T}),\mathcal{C}) \\ &\leq (1+\varepsilon_4)(\texttt{Xloss}(\widetilde{S}_{\varepsilon_3}(\mathcal{T}),\mathcal{T}) + \varepsilon_3\texttt{Yloss}_{\texttt{trim}}(\widetilde{S}_{\varepsilon_3}(\mathcal{T}),\mathcal{T})) \\ &\leq (1+\varepsilon_4)(\texttt{Xloss}(\mathcal{T}) + \texttt{serr}_{\varepsilon_3}(S,\mathcal{T}))\end{aligned}$$

□

In section 7.8, we will give the code for the subprocedures of $\textbf{XLI}_0$, which is the procedure obtained by replacing the exact computation of LIS with **FindLCS** in $\textbf{DMI}_4$.

Since $\mathbf{XLI}_0$ is a modified version of $\mathbf{DMI}_1$, many of these subprocedures are similar or identical to those found in [21]. We will refer to these procedures ($\mathbf{Classify}_t$, $\mathbf{CriticalBox}_t$, $\mathbf{TerminalBox}_t$, $\mathbf{GridChain}_t$, $\mathbf{ApproxLIS}_t$) by name in the remainder of this section.

In section 7.9, we will prove the following theorem, which gives running time bounds for $\mathbf{ApproxLIS}_t$ and $\mathbf{Classify}_t$. Here, $A_{t_{max}}$ and $C_{t_{max}}$ represent the running of $\mathbf{ApproxLIS}_{t_{max}}$ and $\mathbf{Classify}_{t_{max}}$ on boxes of size n, respectively.

Theorem 7.6.2 *For the input parameter ε given in $\boldsymbol{XLI}_0$, both $A_{t_{max}}$ and $C_{t_{max}}$ are in time $(1/\varepsilon)^{O(1/\varepsilon^2)}(\log n)^{O(1)}\sqrt{n}$.*

Using this theorem, we now state and prove the following theorem. From this theorem, Lemma 5.1.3 follows trivially, completing the proof of Theorem 1.2.2. We first introduce the following terminology.

Consider a call to $\mathbf{XLI}_0(x, \mathcal{B})$. For a fixed choice of the random bits, this call to $\mathbf{XLI}_0(x, \mathcal{B})$ may eventually result in a call to $\mathbf{FindLCS}(x, \mathcal{T})$ for $\mathcal{T} \subset \mathcal{B}$. Call this box $\mathcal{T}$ the *final box* for x. Note that (as described above) if x' is the X-index of another match in $\mathcal{T}$, then the final box of x' is also $\mathcal{T}$. If we look at the set of all final boxes of X-indices in $X(\mathcal{B})$, these form a box chain in $\mathcal{B}$, called the *final chain* of $\mathcal{B}$. Note that the final boxes correspond to the green leaves of the tree described in 7.2.

Theorem 7.6.3 *Let $\mathcal{B}$ be a box with $h(\mathcal{B}) \leq 2w(\mathcal{B})$, define $n = d_{max}(\mathcal{B})$, and let S be the set of matches $(x, y) \in \mathcal{B}$ such that $\mathbf{XLI}_0(x, \mathcal{B})$ returns GOOD (for a fixed setting of the random seed). With probability at least $2/3$ over the choices of the random seed, $\mathbf{XLI}_0(x, \mathcal{B})$ has the following properties.*

- *Each call takes $(1/\varepsilon)^{O(1/\varepsilon^2)}(\log n)^{O(1)}\sqrt{n}$ time.*
- *The good points form an increasing sequence.*
- *The number of bad points is at most $(1+\varepsilon_4)(\mathtt{Xloss}(\mathcal{B}) + \mathtt{terr}_{\varepsilon_3}(\mathcal{B})) - d_{out}(\mathcal{B})$.*

Proof For the first item, the running time of $\mathbf{XLI}_0$ comes from the calls to the procedures $\mathbf{TerminalBox}_{-1}$ and $\mathbf{Classify}_{t_{max}}$. As noted in the proof of Theorem 7.6.2, the running time of $\mathbf{TerminalBox}_{-1}$ is dominated by its calls to $\mathbf{FindSplitter}$, of which there are at most $\log n/\rho_0$. By Claim 7.1.4, the cost of **FindSplitter** in the LIS setting is $(\psi \log n)^{O(1)}$. Any call to **FindSplitter** is made on a box of width $\sqrt{n}(\log n)^{O(1)}$. Therefore, by Lemma 7.2.3, any call to **BoxSample**$(m, \mathcal{T})$ made by **FindSplitter** runs in time $\sqrt{n}(\log n)^{O(1)}$ (In any such call, $\log m = o(\sqrt{h(\mathcal{T})})$). As a result, replacing the primitive of obtaining a random sample of points with the procedure **BoxSample** multiplies the running time of **FindSplitter** by $\sqrt{n}(\log n)^{O(1)}$, so the cost of **FindSplitter**, and subsequently **TerminalBox**$_{-1}$ is $\sqrt{n}(\psi \log n)^{O(1)}$. By Theorem 7.6.2, the cost of **Classify**$_{t_{max}}$ is $(1/\varepsilon)^{O(1/\varepsilon^2)}(\log n)^{O(1)}\sqrt{n}$, which gives us the desired running time.

For the second item, we first note that points are only classified as good as a result of a call to **FindLCS**. Thus, the good points are just the union over all final boxes $\mathcal{S}$ of the output of **FindLCS**$(\lambda, \mathcal{S})$. Since the final boxes form a box chain and the output of **FindLCS**$(\lambda, \mathcal{S})$ is a common subsequence, the good points form a common subsequence.

For the third item, suppose that S is the sequence formed by concatenating the LCS's of each box in the final chain of $\mathcal{B}$. By Theorem 7.1.8, the number of X-indices of $\mathcal{B}$ classified as BAD by **DMI**$_1$ is at most $(1+\varepsilon_3)\texttt{Xloss}(\mathcal{B})$ (for the parameters chosen in **XLI**$_0$). By Lemma 7.2.2, Lemma 7.2.5, and Lemma 7.2.6, this same bound holds for the number of X-indices of $\mathcal{B}$ classified as BAD by **DMI**$_4$. Since this quantity is given by $\texttt{Xloss}(S, \mathcal{B})$, $\texttt{Xloss}(S, \mathcal{B}) \leq (1+\varepsilon_3)\texttt{Xloss}(\mathcal{B})$. Moving things around, we get that the length of S is at least $\texttt{lcs}(\mathcal{B}) - \varepsilon_3\texttt{Xloss}(\mathcal{B})$.

Recall that $\widetilde{S}_{\varepsilon_3}(\mathcal{B})$ is the sequence achieving the minimum in $\texttt{serr}_{\varepsilon_3}(S, \mathcal{B})$. Suppose that $\widetilde{S}_{\varepsilon_3}(\mathcal{B})$ is contained in one box in the final chain (call it $\mathcal{C}$). Looking at **XLI**$_0$ and the procedures it calls, **FindLCS** is only called when $w(\mathcal{C}) = \widetilde{O}(\sqrt{d_{max}(\mathcal{B})})$. Therefore, $|\widetilde{S}_{\varepsilon_3}(\mathcal{B})| \leq w(\mathcal{C}) = \widetilde{O}(\sqrt{d_{max}(\mathcal{B})})$. If $d_{max}(\mathcal{B}) = w(\mathcal{B})$, then $w(\mathcal{B}) \leq (1+\varepsilon_4)\texttt{Xloss}(\widetilde{S}_{\varepsilon_3}(\mathcal{B}), \mathcal{B}) \leq (1+\varepsilon_4)(\texttt{Xloss}(\mathcal{B}) + \texttt{serr}_{\varepsilon_3}(S, \mathcal{B})) \leq (1+\varepsilon_4)(\texttt{Xloss}(\mathcal{B}) + \texttt{terr}_{\varepsilon_3}(\mathcal{B}))$, so the number of bad points is at most $(1+\varepsilon_4)(\texttt{Xloss}(\mathcal{B}) + \texttt{terr}_{\varepsilon_3}(\mathcal{B})) - d_{out}(\mathcal{B})$. If $d_{max}(\mathcal{B}) = h(\mathcal{B})$, then since $h(\mathcal{B}) = O(w(\mathcal{B}))$, the same bound follows.

Now suppose that $\widetilde{S}_{\varepsilon_3}(\mathcal{B})$ intersects more than one box in the final chain. The total number of X-indices classified as bad is $\sum_{\mathcal{C} \in \vec{\mathcal{C}}} \texttt{Xloss}(\textbf{FindLCS}(\lambda, \mathcal{C}), \mathcal{C})$. By Lemma 7.6.1,

$\sum_{\mathcal{C}\in\vec{\mathcal{C}}} \mathtt{Xloss}(\mathbf{FindLCS}(\lambda, \mathcal{C}), \mathcal{C}) \leq (1+\varepsilon_4)(\mathtt{Xloss}(\mathcal{B})+\mathtt{serr}_{\varepsilon_3}(S, \mathcal{B})) \leq (1+\varepsilon_4)(\mathtt{Xloss}(\mathcal{B})+$ $\mathtt{terr}_{\varepsilon_3}(\mathcal{B}))$, since the length of S is at least $\mathtt{lcs}(\mathcal{B})-\varepsilon_3\mathtt{Xloss}(\mathcal{B})$. Therefore, the number of bad points in $\mathcal{B}$ is at most $(1+\varepsilon_4)(\mathtt{Xloss}(\mathcal{B})+\mathtt{terr}_{\varepsilon_3}(\mathcal{B})) - d_{out}(\mathcal{B})$. $\square$

7.7 MakeGrid

In this section, we state the procedure **MakeGrid** and prove that it satifies the properties guaranteed by **BuildGrid** in [21]. We borrow the notion of a grid and a grid digraph from [21]:

Grids A *grid* Γ is any Cartesian product $I \times J$ where I is a set of X-indices and J is a set of Y-indices. Thus Γ is a grid if and only if $\Gamma = X(\Gamma) \times Y(\Gamma)$. A box is the special case of a grid in which both $X(\Gamma)$ and $Y(\Gamma)$ are intervals. We refer to the sets of the form $\{x\} \times Y(\Gamma)$ for $x \in X(\Gamma)$ as columns of Γ and sets of the form $X(\Gamma) \times \{y\}$ for $y \in Y(\Gamma)$ as rows of Γ.

Grid digraph $D(\Gamma)$: This is associated with a grid Γ defined on a box $\mathcal{B}$. The vertex set is $\Gamma \cup \{P_{BL}(B), P_{TR}(B)\}$. The arc sets consists of pairs $(P_{BL}(B), Q)$ where Q lies in the leftmost columnn of Γ, $(Q, P_{TR}(B))$ where Q belongs to the rightmost column of Γ, and (P, Q) where $x(P) < x(Q)$, $y(P) \leq y(Q)$, and P and Q are in adjacent columns of Γ. $D(\Gamma)$ is acyclic and has unique source $P_{BL}(B)$ and unique sink $P_{TR}(B)$. A $D(\Gamma)$-path is a source-to-sink path in $D(\Gamma)$. Every arc (P, Q) of $D(\Gamma)$ corresponds to a box with bottom left corner P and top right corner Q, and a $D(\Gamma)$-path corresponds to a box chain spanning $\mathcal{B}$. A box chain arising in this way is a Γ-chain. Each box correponding to an arc is called a grid box.

In [21], the procedure **BuildGrid** was used to construct an α-fine $\mathcal{B}$-grid, given a box $\mathcal{B}$ and $\alpha > 0$. It turns out that in our setting, constructing such a grid is far simpler, as the set of Y-indices is known in advance. As a result, we construct an α-fine $\mathcal{B}$-grid using the procedure **MakeGrid**. We remind the reader of the relevant definitions from [21] before describing this procedure.

$\mathcal{B}$-strips and $\mathcal{B}$-strip decompositions If $\mathcal{B}$ is a box, a $\mathcal{B}$-strip is a subbox $\mathcal{S}$ of $\mathcal{B}$ such that $Y(\mathcal{S}) = Y(\mathcal{B})$. Thus a $\mathcal{B}$-strip has the same vertical extent as $\mathcal{B}$ and is specified relative to $\mathcal{B}$ by its X-index set $X(\mathcal{S})$. If $I \subseteq X(\mathcal{B})$ is an X-interval then $I|\mathcal{B}$ denotes the $\mathcal{B}$-strip with X-index set I. Similarly if $\mathcal{T}$ is a subbox of $\mathcal{B}$ then $\mathcal{T}|\mathcal{B}$ denotes the strip $X(\mathcal{T})|\mathcal{B}$. A $\mathcal{B}$-strip decomposition is a partition of $\mathcal{B}$ into strips. A $\mathcal{B}$-strip decomposition into r strips is specified by a sequence $x_0 = x_L(\mathcal{B}) < x_1 < \cdots < x_r = x_R(\mathcal{B})$, where the j^{th} strip is $(x_{j-1}, x_j]|\mathcal{B}$. We use the sequence notation $\vec{\mathcal{S}}$ to denote a $\mathcal{B}$-strip decomposition in the natural left-to-right order. In particular if $\Gamma \subset \mathcal{B}$ is a grid then Γ naturally defines a strip decomposition of $\mathcal{B}$ obtained by taking the strips that end at successive columns of Γ with the final strip ending at $x_R(\mathcal{B})$.

Given a box, we want to select a suitably representative set of Y-indices from the box. Let us say that a Y-interval J is *α-popular for box $\mathcal{B}$* if there are at least $\alpha w(\mathcal{B})$ X-indices $x \in X(\mathcal{B})$ such that $f(x) \in J$. If $\mathcal{B}$ is a box and $\alpha \in (0, 1)$, a *α-value net* for $\mathcal{B}$ is a subset V of $Y(\mathcal{B})$ such that:

- $y_T(\mathcal{B}) \in V$.
- For all subintervals J of $Y(\mathcal{B})$ that are α-popular for $\mathcal{B}$, $V \cap J = \emptyset$.

A grid Γ is a *$\mathcal{B}$-grid* if $X(\Gamma) \subseteq X(\mathcal{B}) - \{x_L(\mathcal{B}), x_R(\mathcal{B})\}$ and $Y(\mathcal{B})$ is a value net for $\mathcal{B}$. If $x_1 < \cdots < x_k$ are the X-indices of $X(\Gamma)$, then they define a $\mathcal{B}$-strip decomposition of $X(\mathcal{B})$ whose associated X-index partition is $(x_L(\mathcal{B}), x_1], (x_1, x_2], \cdots, (x_k, x_R(\mathcal{B})]$. We call this the strip decomposition of $\mathcal{B}$ induced by Γ.

Definition For a box $\mathcal{B}$ and $\alpha > 0$, a grid Γ is an *α-fine* $\mathcal{B}$-grid if:

- $X(\Gamma)$ contains an X-index from every subinterval I of $X(\mathcal{B})$ having size exceeding $\alpha w(\mathcal{B})$.
- $Y(\Gamma)$ is a $\frac{\alpha}{|X(\Gamma)|}$-value net.

We now describe the procedure **MakeGrid**. The reader should note that this procedure does not require any queries to the input.

```
**MakeGrid**$(\mathcal{B}, \alpha)$
Output: A $\mathcal{B}$-grid $\Gamma$.
1. Let $s_a = \lfloor \frac{1}{\alpha} \rfloor$, $s_b = \lfloor \frac{s_a(h(\mathcal{B})-1)}{\alpha w(\mathcal{B})} \rfloor$
2. Set $X(\Gamma) = \{x_L(\mathcal{B}) + k\alpha w(\mathcal{B}) : 1 \leq k \leq s_a\}$, $Y(\Gamma) = \{y_T(\mathcal{B}) - \frac{k\alpha w(\mathcal{B})}{s_a} : 0 \leq k \leq s_b\}$
3. Output $X(\Gamma) \times Y(\Gamma)$.
```

Proposition 7.7.1 *The grid that* **MakeGrid**$(\mathcal{B}, \alpha)$ *outputs is* α*-fine.*

Proof The first condition of α-fine is easily satisfied, as any interval of size exceeding $\alpha w(\mathcal{B})$ will contain an X-index of the form $x_L(\mathcal{B}) + k\alpha w(\mathcal{B})$ for some integer k. For the second condition, we need to verify that $Y(\Gamma)$ is a $\frac{\alpha}{|X(\Gamma)|}$-value net. It is clear that $y_T(\mathcal{B}) \in Y(\Gamma)$, and if J is a subinterval of $Y(\mathcal{B})$ that is $\frac{\alpha}{|X(\Gamma)|}$-popular for $\mathcal{B}$, then J contains a Y-index of the form $y_T(\mathcal{B}) - \frac{k\alpha w(\mathcal{B})}{s_a}$ for some integer k. $\square$

By lemma 5.15 in [21], it is possible to construct a box chain spanning $\mathcal{B}$ using this grid which misses at most $\alpha w(\mathcal{B})$ points from any common subsequence of $\mathcal{B}$.

7.8 Label

In this section, we state the procedure **XLI**$_0$ and many of the procedures called by it. Again, these procedures are similar to the corresponding procedures in [21].

7.8.1 Procedures

$\mathbf{XLI}_0(i, \mathcal{B})$

Output: ACCEPT or REJECT.

1. Set $n = d_{max}(\mathcal{B})$

2. Fix global parameters (unchanged throughout algorithm, j, r nonnegative integers, C_2 a sufficiently large constant).

Name	**Symbol**	**Value**
Multiplicative parameter	$\overline{\tau}$	$\varepsilon_3^2/1200$
Additive parameter	$\overline{\delta}$	$\varepsilon_3^2/1200$
Maximum level	t_{max}	$\lceil 4/\overline{\tau} \rceil$
Error controller	ψ	$\max(C_2, t_{\max}/\overline{\delta})$
Sample size parameter	σ	$100\psi^3$
Grid precision parameter	α	$\frac{1}{(C_2\psi)^4}$
Width threshold	ω	$4C_2\psi\sqrt{n}\log^2 n$
Tainting parameter	η	$1/10\psi$
Primary splitter parameter	μ_r	$\frac{2}{r+3}$
Secondary splitter parameter	γ_j	$16^j\alpha/(\log n)^4$
Splitter balance parameter	ρ_j	$(\gamma_j)^{1/4} = 2^j\alpha^{1/4}/\log n$
Initial splitter parameter	μ_{-1}	$\varepsilon_3/5$
Proportion threshold	λ	$\frac{\varepsilon_4}{1+\varepsilon_4}$

3. Call $\mathbf{TerminalBox}_{-1}(i, \mathcal{B})$ and let $\mathcal{T}$ be the output box.

4. If $\mathcal{T}$ is $\frac{10\log n}{\rho}\sqrt{n}$-small, run $\mathbf{FindLCS}(\lambda, \mathcal{T})$. If i is on the LCS, ACCEPT, else REJECT.

5. Otherwise, call $\mathbf{Classify}_{t_{max}}(i, \mathcal{T})$. If this outputs “good”, ACCEPT, else REJECT.

```
Classify_t$(x, \mathcal{B})$
Output: good or bad
1. If $t = 0$, return bad.
2. Otherwise ($t \geq 1$): $\mathcal{C} \longleftarrow$ CriticalBox_t$(x, \mathcal{B})$
3. If $\mathcal{C}$ is $\omega$-small, run FindLCS$(\lambda, \mathcal{T})$. If $x$ is on the LCS, return good, else return bad.
4. Otherwise, run Classify_{t-1}$(x, \mathcal{C})$ and return its output.
```

```
CriticalBox_t$(x, \mathcal{B})$
Output: Subbox $\mathcal{C}$ of $\mathcal{B}$ such that $x \in X(\mathcal{C})$
1. $\mathcal{T} \longleftarrow$ TerminalBox_t$(x, \mathcal{B})$.
2. Call GridChain_t$(\mathcal{T})$ and let $\vec{\mathcal{C}}(\mathcal{T})$ be the chain of boxes returned.
3. Return $\vec{\mathcal{C}}(\mathcal{T})[x]$ (the box $\mathcal{C} \in \vec{\mathcal{C}}(\mathcal{T})$ with $x \in X(\mathcal{C})$).
```

```
TerminalBox_t$(x, \mathcal{B})$
Output: Subbox $\mathcal{T}$ of $\mathcal{B}$ such that $x \in X(\mathcal{T})$
1. Initialize: $\mathcal{T} \longleftarrow \mathcal{B}$; $j \longleftarrow 0$; $\theta \longleftarrow \omega$; $\gamma \longleftarrow \gamma_0$; $\rho \longleftarrow \rho_0$.
2. Repeat until $d_{min}(\mathcal{T}) \leq \theta$:
    (a) Run FindSplitter$(\mathcal{T}, \mathcal{B}, \mu_t, \gamma w(\mathcal{T}), \rho)$: returns boolean splitter_found and index splitter.
    (b) If splitter_found = TRUE then
        (i) If $x \leq$ splitter then replace $\mathcal{T}$ by the box $Box(P_{BL}(\mathcal{T}), F(splitter))$.
        (ii) If $x >$ splitter then replace $\mathcal{T}$ by the box $Box(F(splitter), P_{TR}(\mathcal{T}))$.
    (c) else (so splitter_found = FALSE and new phase starts)
        (i) $\theta \longleftarrow max(\theta, \gamma w(\mathcal{T})/\alpha)$
        (ii) $j \longleftarrow j + 1$; $\gamma \longleftarrow \gamma_j$; $\rho \longleftarrow \rho_j$.
3. Return $\mathcal{T}$.
```

```
GridChain$_t(\mathcal{T},\alpha)$
Output: Box chain $\vec{\mathcal{C}}(\mathcal{T})$ spanning $\mathcal{T}$.
1. Call MakeGrid$(\mathcal{T},\alpha)$ which returns a grid $\Gamma$.
2. Construct the associated digraph $D(\Gamma)$
3. For each grid box $\mathcal{D}$ of $D(\Gamma)$. recursively evaluate ApproxLIS$_{t-1}(\mathcal{D})$.
4. Compute the longest path in $D(\Gamma)$ from $P_{BL}(\mathcal{T})$ to $P_{TR}(\mathcal{T})$ according to the length function ApproxLIS$_{t-1}(\mathcal{D})$.
5. Return the $\Gamma$-chain $\vec{\mathcal{C}}(\mathcal{T})$ associated to the longest path.
```

```
ApproxLIS$_t(\mathcal{B})$
Output: Approximation to lis$(\mathcal{B})$
1. If $\mathcal{B}$ is $\omega$-small, run FindLCS$(\mathcal{B})$
2. Otherwise $(d_{min}(\mathcal{B}) \geq \omega)$: Run BoxSample$(\sigma,\mathcal{B})$.
3. If BoxSample$(\sigma,\mathcal{B})$ returns fewer than $\sigma$ points, return 0.
4. Otherwise, run Classify$_t(x,\mathcal{B})$ on each sample point. Let $g$ be the number of points classified as good and return $gw(\mathcal{B})/\sigma$.
```

7.9 Running Time

In this section, we state and prove a claim which allows us to prove the stated running time of **XLI**$_0$ in Theorem 7.6.3. We first analyze the running time of **Classify**. Our argument closely mirrors the argument in [21]. Let $C_t = C_t(n)$ be the running time of **Classify**$_t$ on boxes of width at most n. Similarly, let $A_t = A_t(n)$ be the running time of **ApproxLCS**$_t$ on boxes of width at most n.

As in [21] we use $P_i = P_i(n)$ to denote functions of the form $a_i(\log n)^{b_i}$, where a_i, b_i are constants that are independent of n and t. We also use $Q_i = Q_i(\psi)$ to denote functions of ψ of the form $c_i(\psi)^{d_i}$ where c_i, d_i are constants.

Claim 7.9.1 *For all $t \geq 1$,*

$$A_t \leq P_1 Q_1 \sqrt{n} + Q_2 C_t.$$

$$C_t \leq C_{t-1} + Q_3 A_{t-1} + P_2 Q_4 \sqrt{n}.$$

Proof For the first recurrence, the $P_1Q_1\sqrt{n}$ term comes from the cost of **FindLCS** as well as the cost of **BoxSample**. **FindLCS** is seen to be in time $P_1Q_1\sqrt{n}$ by Prop. 7.4.1, plugging in the values given in $\mathbf{XLI}_0$ for ω and λ. **BoxSample** is seen to be in time $P_1Q_1\sqrt{n}$ by Lemma 7.2.3, noting the value of σ given by $\mathbf{XLI}_0$, as well as the lower bound of ω for the dimensions of $\mathcal{B}$. The Q_2C_t term comes from running $\mathbf{Classify}_t$ on σ sample points.

For the second recurrence, the final recursive call to $\mathbf{Classify}_{t-1}$ gives the C_{t-1} term. The call to **FindLCS** again can be factored into the $P_2Q_4\sqrt{n}$ term. The rest of the cost comes from $\mathbf{CriticalBox}_t$ which invokes $\mathbf{TerminalBox}_t$, which involves several iterations where the cost of each iterations is dominated by the cost of **FindSplitter**. Each iteration reduces the size of the box $\mathcal{T}$ by at least a $(1 - \rho_0)$ factor so the number of iterations is at most $\log n/\rho_0$. By Claim 7.1.4, the cost of **FindSplitter** in the LIS setting is $(\psi \log n)^{O(1)}$. Any call to **FindSplitter** is made on a box of width $\sqrt{n}(\log n)^{O(1)}$. Therefore, by Lemma 7.2.3, any call to **BoxSample** made by **FindSplitter** runs in time $\sqrt{n}(\log n)^{O(1)}$ (In any such call, $\log m = o(\sqrt{h(\mathcal{T})})$). As a result, replacing the primitive of obtaining a random sample of points with the procedure **BoxSample** multiplies the running time of **FindSplitter** by $\sqrt{n}(\log n)^{O(1)}$, so the cost of **FindSplitter**, and subsequently $\mathbf{TerminalBox}_t$ is $\sqrt{n}(\psi \log n)^{O(1)}$, and thus is included in the term $P_2Q_4\sqrt{n}$.

$\mathbf{CriticalBox}_t$ then calls $\mathbf{GridChain}_t$. This involves taking the grid given by **MakeGrid** and making one call to $\mathbf{ApproxLCS}_t$ for each grid box. Since the number of grid boxes is $O(\alpha^c)$, this can be accounted for by the Q_3A_{t-1} term. $\mathbf{GridChain}_t$ finds a longest path in the grid digraph, which can be absorbed into the $P_2Q_4\sqrt{n}$ term. □

Using this claim, we prove Theorem 7.6.2.

Proof Using the recurrence for A_{t-1} to eliminate A_{t-1} from the recurrence for C_t gives a linear recurrence for C_t in terms of C_{t-1} whose solution has the form $C_t \leq P_5Q_5\sqrt{n}(Q_2Q_3 + 1)^t$. This leads also to $A_t \leq P_6Q_6\sqrt{n}(Q_2Q_3 + 1)^t$, which are both $\sqrt{n}(\log n)^{O(1)}(\psi)^{O(t)}$. Since $t = O(1/\varepsilon^2)$ and $\psi = O(1/\varepsilon^4)$, the bound follows. □

Bibliography

[1] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Estimating the distance to a monotone function. *Random Structures and Algorithms*, 31(3):371–383, 2007. 37, 73

[2] D. Aldous and P. Diaconis. Longest increasing subsequences: from patience sorting to the Baik-Deift-Johannson theorem. *Bulletin of the American Mathematical Society*, 36:413–432, 1999. 3, 4

[3] A. Andoni, P. Indyk, and R. Krauthgamer. Overcoming the ℓ_1 non-embeddability barrier: algorithms for product matrices. In *Proceedings of the 20th Symposium on Discrete Algorithms (SODA)*, pages 865–874, 2009. 5

[4] A. Andoni and H. L. Nguyen. Near-optimal sublinear time algorithms for ulam distance. In *Proceedings of the 21st Symposium on Discrete Algorithms (SODA)*, 2010. iii, 4, 5, 36, 37, 47, 53, 59, 69

[5] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *CoRR*, abs/1412.0348, 2014. 4

[6] Tugkan Batu, Funda Ergun, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of Symposium of Theory of Computing (STOC)*, pages 316–324, 2003. 5

[7] M. Charikar and R. Krauthgamer. Embedding the ulam metric in ℓ_1. *Theory of Computing*, 2:207–224, 2006. 4

[8] D. Critchlow. Ulam's metric. *Encyclopedia of Statistical Sciences*, 9:379–380, 1988. 4

[9] F. Ergun and H. Jowhari. On distance to monotonicity and longest increasing subsequence of a data stream. In *Proceedings of the 19th Symposium on Discrete Algorithms (SODA)*, pages 730–736, 2008. ii, 3

[10] F. Ergun, S. Kannan, R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *Journal of Computer Systems and Sciences (JCSS)*, 60(3):717–751, 2000. 37

[11] M. Fredman. On computing the length of the longest increasing subsequences. *Discrete Mathematics*, 11:29–35, 1975. 3

[12] A. Gal and P. Gopalan. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. In *Proceedings of the 48th Symposium on Foundations of Computer Science (FOCS)*, pages 294–304, 2007. 3

[13] P. Gopalan, T. S. Jayram, R. Krauthgamer, and R. Kumar. Estimating the sortedness of a data stream. In *Proceedings of the 18th Symposium on Discrete Algorithms (SODA)*, pages 318–327, 2007. ii, 3

[14] J. W. Hunt and T. G. Szymanski. A fast algorithm for computing longest common subsequences. *Commun. ACM*, 20(5):350–353, 1977. 87, 88, 89, 91

[15] Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. *Computational Complexity*, 8:596–605, 1995. 34, 35

[16] Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *J. Comput. Syst. Sci.*, 57(1):37–49, August 1998. 34

[17] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. 11

[18] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. 11

[19] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 6(72):1012–1042, 2006. 37

[20] P. Ramanan. Tight $\Omega(n \lg n)$ lower bound for finding a longest increasing subsequence. *International Journal of Computer Mathematics*, 65(3 & 4):161–164, 1997. 3

[21] M. Saks and C. Seshadhri. Estimating the longest increasing sequence in polylogarithmic time. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 458–467, 2010. 5, 7, 36, 37, 38, 41, 52, 72, 73, 74, 75, 79, 82, 83, 92, 95, 97, 100

[22] M. Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *Proceedings of the 24th Symposium on Discrete Algorithms (SODA)*, 2013. ii, 3

[23] C. Schensted. Longest increasing and decreasing subsequences. *Canadian Journal of Mathematics*, 13:179–191, 1961. 3

[24] X. Sun and D. P. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *Proceedings of the 18th Symposium on Discrete Algorithms (SODA)*, pages 336–345, 2007. 31