

Range Longest Increasing Subsequence and Its Relatives

Karthik C. S.  

Rutgers University, New Brunswick, NJ, USA

Saladi Rahul  

Indian Institute of Science, Bangalore, India

Abstract

Longest increasing subsequence (LIS) is a classical textbook problem which is still actively studied in various computational models. In this work, we present a few results for the range longest increasing subsequence problem (*Range-LIS*) and its variants. The input to *Range-LIS* is a sequence \mathcal{S} of n real numbers and a collection \mathcal{Q} of m query ranges and for each query in \mathcal{Q} , the goal is to *report* the LIS of the sequence \mathcal{S} restricted to that query. Our two main results are for the following generalizations of the *Range-LIS* problem:

2D Range Queries: In this variant of the *Range-LIS* problem, each query is a pair of ranges, one of indices and the other of values, and we provide a randomized algorithm with running time¹ $\tilde{O}(mn^{1/2} + n^{3/2}) + O(k)$, where k is the cumulative length of the m output subsequences. This improves on the elementary $\tilde{O}(mn)$ runtime algorithm when $m = \Omega(\sqrt{n})$. Previously, the only known result breaking the quadratic barrier was of Tiskin [SODA'10] which could only handle 1D range queries (i.e., each query was a range of indices) and also just outputted the *length* of the LIS (instead of reporting the subsequence achieving that length). Subsequent to our paper, Gawrychowski, Gorbachev, and Kociumaka in a preprint have extended Tiskin's approach to handle reporting 1D range queries in $O(n(\log n)^3 + m + k)$ time.

Colored Sequences: In this variant of the *Range-LIS* problem, each element in \mathcal{S} is colored and for each query in \mathcal{Q} , the goal is to report a monochromatic LIS contained in the sequence \mathcal{S} restricted to that query. For 2D queries, we provide a randomized algorithm for this colored version with running time $\tilde{O}(mn^{2/3} + n^{5/3}) + O(k)$. Moreover, for 1D queries, we provide an improved algorithm with running time $\tilde{O}(mn^{1/2} + n^{3/2}) + O(k)$. Thus, we again improve on the elementary $\tilde{O}(mn)$ runtime algorithm. Additionally, we prove that assuming the well-known Combinatorial Boolean Matrix Multiplication Hypothesis, that the runtime for 1D queries is essentially tight for combinatorial algorithms.

Our algorithms combine several tools such as dynamic programming (to precompute increasing subsequences with some desirable properties), geometric data structures (to efficiently compute the dynamic programming entries), random sampling (to capture elements which are part of the LIS), classification of query ranges into large LIS and small LIS, and classification of colors into light and heavy. We believe that our techniques will be of interest to tackle other variants of LIS problem and other range-searching problems.

2012 ACM Subject Classification Theory of computation \rightarrow Data structures design and analysis; Theory of computation \rightarrow Computational geometry

Keywords and phrases Longest Increasing Subsequence, Range Query, Fine-Grained Complexity

Digital Object Identifier 10.4230/LIPIcs.ITCS.2026.87

Related Version *Full Version:* <https://arxiv.org/abs/2404.04795>

Funding *Karthik C. S.:* This work was supported by the National Science Foundation under Grant CCF-2313372 and by the Simons Foundation, Grant Number 825876, Awardee Thu D. Nguyen.

¹ The notation \tilde{O} hides polylogarithmic factors in n and m .



Saladi Rahul: This work supported in part by the Walmart Center for Tech Excellence at IISc (CSR Grant WMGT-23-0001).

Acknowledgements We thank the anonymous reviewers for several helpful comments that improved the presentation of our results.

1 Introduction

In the *longest increasing subsequence* (LIS) problem, the input is a sequence of n real numbers $\mathcal{S} = (a_1, a_2, \dots, a_n)$ and the goal is to *report* indices $1 \leq i_1 < i_2 < \dots < i_t \leq n$, for the largest possible value of t , such that $a_{i_1} < a_{i_2} < \dots < a_{i_t}$. The *length* of the LIS is then defined to be t . For simplicity of discussion, we will assume that all the real numbers in the sequence are distinct. A standard dynamic programming algorithm reports the LIS in $O(n^2)$ time, which can be improved to $O(n \log n)$ by performing patience sort [51, 52, 11, 27] or by suitably augmenting a binary search tree which computes each entry in the dynamic programming table in $O(\log n)$ time.

In many applications, such as time-series data analysis, the user might not be interested in the LIS of the entire sequence. Instead they would like to focus their attention only on a “range” of indices in the sequence. Here are a few examples to illustrate this point.

- Consider a stock X in the trading market. Let the sequence represent the daily price of stock X from 1950 till 2022. Instead of querying for the LIS of the entire sequence, the user might gain more insights [38, 49] about the trends of the stock by querying for the LIS in a range of dates (or time-windows) such as Jan 2020 till March 2020 or Feb 2018 till Dec 2018.
- In modern times, social media platforms are the major source for generating enormous content on the Internet on a minute-to-minute basis. Consider a time-series data with statistics at a fine-grained level (of each minute of the day) about the number of Google searches, number of Tweets posted, number of Amazon purchases, or the number of Instagram posts. Monitoring the LIS under time-window constraints on such datasets can lead to more insights in understanding the social media usage trends and can also help in detecting abnormalities.
- Popular genome sequencing algorithms identify high scoring segment pairs (HSPs) between a query transcript sequence and a long reference genomic sequence in order to obtain a global alignment, and this amounts to computing the LIS in the reference genome [9, 80, 8]. Typically many queries are made (corresponding to various transcripts/proteins) w.r.t. the same reference genomic sequence, and this can be modeled as range queries to compute the LIS.

Recently, in the theoretical computer science community, there has been a lot of interest in the dynamic LIS problem [53, 43, 31], where the goal is to maintain exact or approximate LIS under insertion and deletion of elements. Interestingly, an important subroutine which shows up is the computation of LIS of a given range of elements in the sequence. For example, Gawrychowski and Janczewski [31] design a dynamic algorithm to quickly compute the approximate LIS for any range of indices in \mathcal{S} . In fact, Gawrychowski and Janczewski go further and design a dynamic algorithm which maintains the approximate LIS for a special class of *dyadic rectangles* where the query² is an axis-aligned rectangle in 2D and the goal is to compute the approximate LIS of the 2D points (from the above mapping) which lie inside the rectangle.

² Consider mapping the i^{th} element $a_i \in \mathcal{S}$ to a 2D point (i, a_i) , to contextualize the notion of 2D queries.

Motivated by this, in this paper we study several natural variants of the LIS problem where the 1D queries impose restriction on the range of the indices and the 2D queries impose range restriction on both the indices and the values in the sequence. For each of these variants of the LIS problem there is a data structure counterpart that can be studied as well, and these results are later listed in Table 2.

1.1 Vanilla Problem: One dimensional range LIS problem

The first problem is the *1D range longest increasing subsequence* problem (*1D-Range-LIS*). In the *1D-Range-LIS* problem, we are given as input a sequence $\mathcal{S} = (a_1, a_2, \dots, a_n)$ of n real numbers and a collection \mathcal{Q} of m query ranges, where each $q \in \mathcal{Q}$ is a range (or an interval) $[x_1, x_2] \subseteq [1, n]$. For each $q = [x_1, x_2] \in \mathcal{Q}$, the goal is to *report* the LIS of the sequence $\mathcal{S} \cap [x_1, x_2] = (a_{x_1}, a_{x_1+1}, \dots, a_{x_2-1}, a_{x_2})$ (i.e., the output must be the longest increasing subsequence in $\mathcal{S} \cap [x_1, x_2]$). The length of the LIS of the sequence $\mathcal{S} \cap q$ is denoted by $\text{LIS}(\mathcal{S} \cap q)$. Throughout the paper, we abuse notation for simplicity by using LIS to refer to both the subsequence and its length. We clarify the intended meaning wherever it may be unclear.

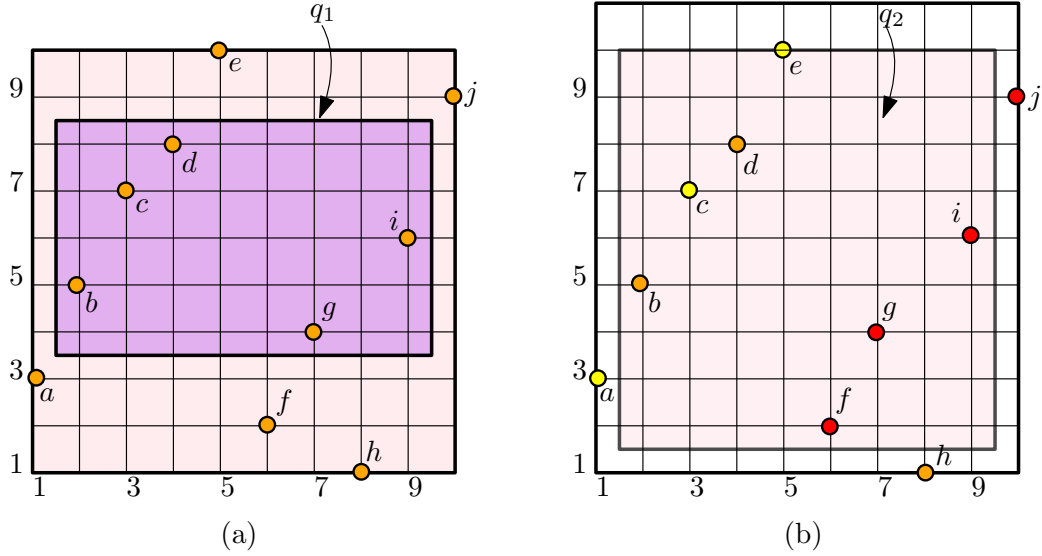
A straightforward approach to solve the *1D-Range-LIS* problem would involve no preprocessing: for each range $[x_1, x_2]$ in \mathcal{Q} , we will retrieve the elements $a_{x_1}, a_{x_1+1}, \dots, a_{x_2-1}, a_{x_2}$ and report their LIS. This would require $\Omega(mn \log n)$ time in the worst-case (for instance when many of the ranges in \mathcal{Q} are of $\Omega(n)$ length). In a remarkable work, Tiskin [74, 76, 75] broke the quadratic barrier, by designing an $O((n+m) \log n)$ time algorithm, albeit for the computationally easier *length* version of the *1D-Range-LIS* problem, in which for all $q \in \mathcal{Q}$, the goal is to only output the length of the LIS in the range q (i.e., $\text{LIS}(\mathcal{S} \cap q)$). While the algorithm designed by Tiskin is tailored to handle the length version of the *1D-Range-LIS* problem, in a subsequent work to the current paper, Gawrychowski, Gorbachev, and Kociumaka [29] extend Tiskin’s algorithm to handle the reporting version as well by providing an algorithm which answers the reporting version of the *1D-Range-LIS* problem in $O(n(\log n)^3 + m + k)$ time.

Since the vanilla version of the range LIS problem is completely understood (up to log factors), we focus on the following three variants.

1.2 Problem-I: Two dimensional range LIS problem

The next problem is the *2D range longest increasing subsequence* problem (*2D-Range-LIS*) which is a natural generalization of the *1D-Range-LIS* problem. As before, we are given as input a sequence $\mathcal{S} = (a_1, a_2, \dots, a_n)$. Consider a mapping where the i^{th} element $a_i \in \mathcal{S}$ is mapped to a 2D point $p_i = (i, a_i)$. Let $\mathcal{P} = \{p_1, p_2, \dots, p_n\}$ be a collection of these n points. We are also given as input a collection \mathcal{Q} of m query ranges, where each range in \mathcal{Q} is an axis-aligned rectangle in 2D. For each $q \in \mathcal{Q}$, the goal is to report the LIS of $\mathcal{P} \cap q$, i.e., the points of \mathcal{P} lying inside q . See Figure 1(a) for an example.

Once again the naive approach to solve the problem would take $O(mn \log n)$ time, where for each range $q \in \mathcal{Q}$, we will scan the points in \mathcal{P} to filter out the points lying inside q and then compute their LIS. One might be tempted to use *orthogonal range searching* data structures [3, 5, 24, 56, 13] to efficiently report $\mathcal{P} \cap q$, but in the worst-case we could have many queries with $|\mathcal{P} \cap q| = \Omega(n)$. In this paper, even for the *2D-Range-LIS* problem, we succeed to improve on the basic quadratic runtime algorithm when m is roughly $\Omega(\sqrt{n})$. We obtain the following result where the $\tilde{O}(\cdot)$ notation hides polylog factors.



■ **Figure 1** (a) Each element in the sequence $S = (3, 5, 7, 8, 10, 2, 4, 1, 6, 9)$ is mapped to a 2D point. For example, the element 7 is mapped to $c = (3, 7)$. The $2D$ -Range-LIS of the points lying inside q_1 is (b, c, d) . (b) The same input sequence $S = (3, 5, 7, 8, 10, 2, 4, 1, 6, 9)$ is shown here. The set of colors $\mathcal{C} = \{\text{orange, yellow, red}\}$. Points b, d and h have orange color, points a, c and e have yellow color, and points f, g, i and j have red color. The $Colored$ - $2D$ -Range-LIS of the points lying inside q_2 (the shaded region) is red color realized by the sequence (f, g, i) .

▶ **Theorem 1.** *There is a randomized algorithm to answer the reporting version of $2D$ -Range-LIS problem in $\tilde{O}(\sqrt{n} \cdot (m + n)) + O(k)$ time, where k is the cumulative length of the m output subsequences. The bound on the running time and the correctness of the solution holds with high probability.*

1.3 Problem-II: Colored 1D range LIS problem

In the geometric data structures literature, *colored range searching* [41, 57, 59, 54, 45, 40, 14, 6, 34, 37, 46, 55, 71, 33] is a well-studied generalization of traditional range searching problems. In the colored setting, the geometric objects are partitioned into disjoint groups (or colors), and given a query region q , the typical goal is to efficiently report [22, 25, 23, 47], or count [39], or approximately count [60] the number of colors intersecting q , or find the majority color inside q [21, 44].

In the same spirit, we study the *colored 1D range longest increasing subsequence* problem ($Colored$ - $1D$ -Range-LIS). In addition to the sequence S and 1D range queries \mathcal{Q} given as input to the $1D$ -Range-LIS problem, in the $Colored$ - $1D$ -Range-LIS problem we are additionally given a coloring of S using the color set \mathcal{C} . Each element $a_i \in S$ has a *color* chosen from \mathcal{C} (the corresponding 2D point p_i also has the same color). For all $c \in \mathcal{C}$, let $\mathcal{P}_c \subseteq \mathcal{P}$ denote the set of points with color c and for any $q \in \mathcal{Q}$, with a slight abuse of notation, we will use $\text{LIS}(\mathcal{P}_c \cap q)$ to denote the length of the LIS of $\mathcal{P}_c \cap q$. Moreover, we say a subsequence is *monochromatic*, if all the elements in the subsequence have the same color. For each $q \in \mathcal{Q}$, the goal is then to report the longest monochromatic increasing subsequence whose length is equal to $\max_{c \in \mathcal{C}} \text{LIS}(\mathcal{P}_c \cap q)$. See Figure 1(b) for an example in 2D. We obtain the following result.

► **Theorem 2.** *There is a randomized algorithm to answer the reporting version of the Colored-1D-Range-LIS problem in $\tilde{O}(\sqrt{n} \cdot (m+n)) + O(k)$ time, where k is the cumulative length of the m output subsequences. The bound on the running time and the correctness of the solution holds with high probability.*

We complement the above result with a conditional lower bound that indicates that the above runtime for *Colored-1D-Range-LIS* might be (near) optimal.

► **Theorem 3.** *Assuming the Combinatorial Boolean Matrix Multiplication Hypothesis, for every $\varepsilon > 0$ and $C \in \mathbb{N}$, there is no combinatorial algorithm to answer the reporting version of the Colored-1D-Range-LIS problem in $k/2 + C \cdot n^{0.5-\varepsilon} \cdot (m+n)$ time, where k is the cumulative size of the m outputs. The lower bound continues to hold even when we are only required to report the color of the longest monochromatic increasing subsequence for each range query.*

The Combinatorial Boolean Matrix Multiplication Hypothesis (CBMMH) roughly asserts that Boolean matrix multiplication of two $\sqrt{n} \times \sqrt{n}$ matrices cannot be computed by any combinatorial algorithm running in time $n^{1.5-\varepsilon}$, for any constant $\varepsilon > 0$. Here “combinatorial algorithm” is typically defined as “non-Strassen-like algorithm” (as defined in [12]), and this captures all known fast matrix multiplication algorithms.

While CBMMH is widely explored since the 1990s [70, 48], there is no strong consensus in the computer science community about its plausibility. In a recent breakthrough, Abboud et al. [1], have come tantalizingly close to even refuting it. All that said, CBMMH remains a widely used hypothesis for proving conditional lower bounds [66, 2, 35]. At the very least, Theorem 3 provides evidence that any algorithm for *Colored-1D-Range-LIS* that is significantly faster than the runtime given in Theorem 2, must be highly non-trivial.

The proof of Theorem 3 follows as a simple consequence of a CBMMH based conditional lower bound in [21] for computing the mode of a sequence subject to range queries.

1.4 Problem-III: Colored 2D range LIS problem

The most general problem that we study in this paper is the *colored 2D range longest increasing subsequence* problem (*Colored-2D-Range-LIS*). The queries in \mathcal{Q} will be axis-aligned rectangles. For each $q \in \mathcal{Q}$, the goal is then to report the longest monochromatic increasing subsequence which attains $\max_{c \in C} \text{LIS}(\mathcal{P}_c \cap q)$. See Figure 1(b) for an example. We obtain the following result.

► **Theorem 4.** *There is a randomized algorithm to answer the reporting version of the Colored-2D-Range-LIS problem in $\tilde{O}(n^{2/3} \cdot (m+n)) + O(k)$ time, where k is the cumulative length of the m output subsequences. The bound on the running time and the correctness of the solution holds with high probability.*

We remark that the conditional lower bound of Theorem 3 continues to hold here too.

In Table 1, we have summarized the state-of-the-art upper and (conditional) lower bounds for the various variants of *Range-LIS* discussed in this paper.

1.5 Data Structure Counterpart of the Three Variants of LIS Problem

A natural data structure counterpart question to the aforementioned four problems is to design a data structure for these problems and measure the tradeoff in the time needed for preprocessing the input sequence versus the time needed to answer a single range query. In Table 2, we have summarized the preprocessing and query time bounds that can be derived from the proofs of Theorems 1, 2, and 4.

■ **Table 1** State-of-the-art upper and lower bounds are listed for the variants of *Range-LIS* problem considered in this paper. In the table, all upper bounds from this paper are randomized, and all lower bounds from this paper are conditional and only against combinatorial algorithms. Also, recall that n is the length of input sequence, m is the number of range queries, and k is the cumulative size of the m outputs.

Problem	Upper Bound	Lower Bound
1D-Range LIS	$O(n(\log n)^3 + m + k)$ (Theorem 5.13 in [29])	$\Omega(m + n + k)$ (Trivial)
2D-Range LIS	$\tilde{O}(\sqrt{n} \cdot (m + n)) + O(k)$ (Theorem 1)	$\Omega(m + n + k)$ (Trivial)
Colored 1D-Range LIS	$\tilde{O}(\sqrt{n} \cdot (m + n)) + O(k)$ (Theorem 2)	$\Omega(n^{1/2-o(1)}(m + n) + k)$ (Theorem 3)
Colored 2D-Range LIS	$\tilde{O}(n^{2/3} \cdot (m + n)) + O(k)$ (Theorem 4)	$\Omega(n^{1/2-o(1)}(m + n) + k)$ (Theorem 3)

■ **Table 2** State-of-the-art upper bounds are listed for the data structure variants of *Range-LIS* problems considered in this paper. In the table, all the bounds are randomized. Also, k_q is the length of the output of a single query.

Problem	Preprocessing Time	Query time	Reference
1D-Range LIS	$O(n(\log n)^3)$	$O(k_q)$	[29]
2D-Range LIS	$\tilde{O}(n^{3/2})$	$\tilde{O}(n^{1/2}) + O(k_q)$	This Paper
Colored 1D-Range LIS	$\tilde{O}(n^{3/2})$	$\tilde{O}(n^{1/2}) + O(k_q)$	This Paper
Colored 2D-Range LIS	$\tilde{O}(n^{5/3})$	$\tilde{O}(n^{2/3}) + O(k_q)$	This Paper

Finally, we note that the proof of Theorem 3 also implies that assuming CBMMH, for every $\varepsilon > 0$, there is no data structure for the reporting version of the *Colored-1D-Range-LIS* problem (or the *Colored-2D-Range-LIS* problem) which can be preprocessed using a combinatorial algorithm in $O(n^{1.5-\varepsilon})$ time such that each query can be answered using a combinatorial algorithm in $\frac{k_q}{2} + O(n^{0.5-\varepsilon})$ time.

1.6 Related Works

LIS in popular settings and models

To the best of our knowledge, there is not much prior work on the reporting version of *Range-LIS*. The results of Tiskin [74, 76, 75] on the length version of *1D-Range-LIS* were later adapted to the reporting version in [29], subsequent to our work. Therefore, for the rest of this subsection, we list works on the LIS problem in some popular settings/models.

In the Dynamic LIS problem we need to maintain the length of LIS of an array under insertion and deletion. The LIS problem in the dynamic setting was initiated by Mitzenmacher and Sedgighin [53]. In [31], an algorithm running in $O(\varepsilon^{-5}(\log n)^{11})$ time and providing

$(1 + \varepsilon)$ -approximation was presented (this approximation algorithm can be adapted to approximate the *Range-LIS* problem in the dynamic setting), and in [43] a randomized exact algorithm with the update and query time $\tilde{O}(n^{2/3})$ was provided. Finally, in [30], the authors provide conditional polynomial lower bounds for exactly solving *Range-LIS* in the dynamic setting.

In the streaming model, computing the LIS requires $\Omega(n)$ bits of space [32], so it is natural to resort to approximation algorithms. In [32] is a deterministic $(1 + \varepsilon)$ -approximation in $O(\sqrt{n/\varepsilon})$ space for LIS problem, and this was shown to be optimal [26, 28]. We remark here that the LIS problem has also been implicitly studied in the streaming algorithms literature as estimating the sortedness of an array [7, 50, 32, 72].

In the setting of sublinear time algorithms, the authors of [68] showed how to approximate the length of the LIS to within an additive error of $\varepsilon \cdot n$, for an arbitrary $\varepsilon \in (0, 1)$, with $(1/\varepsilon)^{1/\varepsilon} \cdot (\log n)^{O(1)}$ queries. Denoting the length of LIS by ℓ , in [67] the authors designed a non-adaptive $O(\lambda^{-3})$ -multiplicative factor approximation algorithm, where $\lambda = \ell/n$, with $O(\sqrt{n}/\lambda^7)$ queries (and also obtained different tradeoff between the dependency on λ and n). In [58], the authors proved that adaptivity is essential in obtaining polylogarithmic query complexity for the LIS problem. Recently, for any $\lambda = o(1)$, in [10] a (randomized) non-adaptive $1/\lambda^{o(1)}$ -multiplicative factor approximation algorithm was provided with $n^{o(1)}/\lambda$ running time.

In the read-only random access model, the authors of [42] showed how to find the length of LIS in $O((n^2/s) \log n)$ time and only $O(s)$ space, for any parameter $\sqrt{n} \leq s \leq n$.

Range-aggregate queries

Range-aggregate queries have traditionally been well-studied in the database community and the computational geometry community. See the survey papers [5, 65, 33]. In a typical range-aggregate query, the input is a collection of points in a d -dimensional space, and the user specifies a range query (such as an axis-aligned rectangle, or a disk, or a halfspace), and the goal is to compute an informative summary of the subset of points which lie inside the range query, such as the top- k points [15, 63, 64, 4, 17], a random subset of k points [73, 36], reporting or counting *colors* or groups [25, 22, 61], statistics such as mode, min [16], median [18], or sum, the closest pair of points [77, 78, 79], and the skyline points [19, 20, 62].

In an *orthogonal range-max* problem, the input is a set \mathcal{P} of n points in d -dimensional space. Each point in \mathcal{P} has a weight associated with it. Preprocess \mathcal{P} into a data structure, so that given an axis-parallel box q , the goal is to report the point in $\mathcal{P} \cap q$ with the largest weight. In this work, we will use *vanilla range trees* from the textbook [13] to answer range-max queries (since the goal of this work is not to shave polylogarithmic factors). The preprocessing time to build a vanilla range tree is $O(n \log^{d-1} n)$ and the query time is $O(\log^d n)$.

2 Our first technique: Handling small LIS

We design a general technique to obtain all our upper bounds. We note that throughout the paper, there is ample scope to shave logarithmic factors in the runtime of our various algorithms and subroutines. However, that is not the focus of this paper.

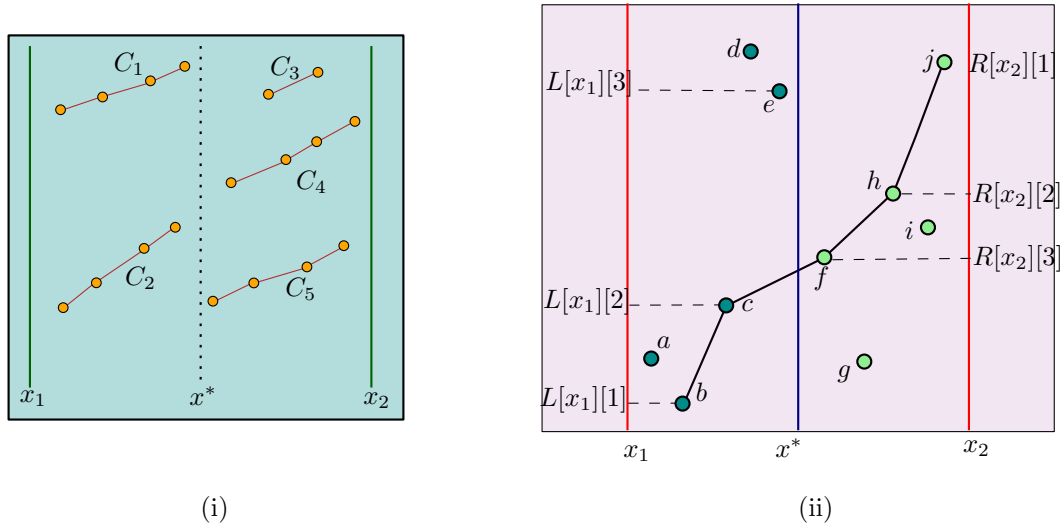
For the sake of simplicity, we provide an overview of the general technique for the simplest case of *1D-Range-LIS* which is in 1D and does not involve colors. Our strategy to solve the *1D-Range-LIS* problem is the following. Consider an integer parameter $\tau \in [1, n]$. For the

1D-Range-LIS problem, τ is set to \sqrt{n} . We will design two different techniques. The first technique is deterministic and reports the correct answer if $\text{LIS}(\mathcal{S} \cap q) \leq \tau$. If $\text{LIS}(\mathcal{S} \cap q) > \tau$, then correctness is not guaranteed. The second technique is randomized and reports the correct answer with high probability if $\text{LIS}(\mathcal{S} \cap q) \geq \tau$. As such, for each $q \in \mathcal{Q}$, w.h.p., the correct result is obtained by one of the two algorithms.

In this subsection, we will give an overview of the first technique for 1D-Range-LIS. In our discussion we will use the terms element and point interchangeably. As discussed before, the i^{th} element in sequence \mathcal{S} is equivalent to the 2D point (i, a_i) in \mathcal{P} . Consider the special case where there is a vertical line with x -coordinate x^* such that each query range in \mathcal{Q} contains x^* (in the full version of the paper we show how to remove the assumption).

2.1 Idea-1: Lowest peaks and highest bases

Please refer to Figure 2(i) where eighteen points are shown. Chain C_1 and chain C_2 are increasing sequences to the left of x^* and both have length four. Our first observation is that if we are allowed to store either C_1 or C_2 , then it would be wiser to store C_2 . The reason is that C_2 can “stitch” itself with chain C_3 and C_4 , to form an increasing subsequence of length six and eight, respectively. On the other hand, C_1 cannot stitch itself with C_3 or C_4 to form a larger increasing subsequence. Analogously, between chains C_4 and C_5 , we will prefer C_4 since it can stitch with C_2 , whereas C_5 cannot stitch with any chain to the left of x^* . In fact, the LIS of the eighteen points is realised by the chain $C_2 \cup C_4$ with length eight. This motivates the definition of *lowest peak* and *highest base*.



■ **Figure 2** (i) An example with eighteen points and five chains is shown. (ii) An example with ten points. The query is the range $[x_1, x_2]$. In the range $[x_1, x^*]$, the sequence with the lowest peak of length one is (b) , of length two is (b, c) or (a, c) , of length three is (b, c, e) or (a, c, e) . In the range $[x^*, x_2]$, the sequence with the highest base of length one is (j) , of length two is (h, j) , of length three (f, h, j) or (f, i, j) . The LIS of points in the range $[x_1, x_2]$ is five and the corresponding compatible pair is $(L[x_1][2], R[x_2][3])$.

For an increasing subsequence $S \in \mathcal{S}$, we define *peak* (resp., *base*) to be the last (resp., first) element in S . The algorithm will store two 2-dimensional arrays:

Array L : For all integers $1 \leq x_1 \leq x^*$ and for all $1 \leq \alpha \leq \tau$, among all increasing subsequences of length α in the range $[x_1, x^*]$, let S be the subsequence with the *lowest* peak. Then $L[x_1][\alpha]$ stores the value of the last element in S .

Array R : For all integers $x^* < x_2 \leq n$ and for all $1 \leq \beta \leq \tau$, among all increasing sequences of length β in the range $(x^*, x_2]$, let S be the subsequence with the *highest* base. Then $R[x_2][\beta]$ stores the value of the first element in S .

For example, in Figure 2(i), for the range $[x_1, x^*]$ and $\alpha = 4$, the chain C_2 is the subsequence with the lowest peak; and for $(x^*, x_2]$ and $\alpha = 4$, chain C_4 is the subsequence with the highest base. Efficient computation of the $O(n\tau)$ entries in arrays L and R will be discussed later.

A pair $(L[x_1][\alpha], R[x_2][\beta])$ is defined to be *compatible* if $L[x_1][\alpha] < R[x_2][\beta]$, i.e., it is possible to “stitch” the sequences corresponding to $L[x_1][\alpha]$ and $R[x_2][\beta]$ to obtain an increasing sequence of length $\alpha + \beta$. For a query range $q = [x_1, x_2]$, our goal is to find the compatible pair $(L[x_1][\alpha], R[x_2][\beta])$ which maximizes the value of $\alpha + \beta$, i.e.,

$$\text{LIS}(\mathcal{S} \cap [x_1, x_2]) = \max_{\alpha, \beta} \{\alpha + \beta \mid L[x_1][\alpha] < R[x_2][\beta]\}.$$

2.2 Idea-2: Monotone property of peaks and bases

For a given query $q = [x_1, x_2]$, a naive approach is to inspect all pairs of the form $(L[x_1][\alpha], R[x_2][\beta])$, for all $1 \leq \alpha, \beta \leq \tau$, and then pick a compatible pair which maximizes the value of $\alpha + \beta$. The number of pairs inspected is $\Theta(\tau^2)$ which we cannot afford. However, the following *monotone property* will help in reducing the number of pairs inspected to $O(\tau \log \tau)$:

- For a fixed value of x_1 , the value of $L[x_1][\alpha]$ increases as the value of α increases, i.e., for any $1 \leq \alpha \leq \alpha' \leq \tau$, we have $L[x_1][\alpha] \leq L[x_1][\alpha']$.
- Analogously, for a fixed value of x_2 , the value of $R[x_2][\beta]$ decreases as the value of β increases, i.e., for any $1 \leq \beta \leq \beta' \leq \tau$, we have $R[x_2][\beta] \geq R[x_2][\beta']$.

See Figure 2(ii) for an example where $L[x_1][\alpha]$ values are increasing with increase in α and $R[x_2][\beta]$ values are decreasing with increase in β . For each $1 \leq i \leq \tau$, the largest β_i for which $(L[x_1][i], R[x_2][\beta_i])$ is compatible can be found in $O(\log \tau)$ time by doing a binary search on the monotone sequence $(R[x_2][1], R[x_2][2], \dots, R[x_2][\tau])$. Finally, report $\max_i (i + \beta_i)$ as the length of the LIS for $\mathcal{S} \cap q$. Overall, the time taken to answer the m query ranges will be $O(m\tau \log \tau)$.

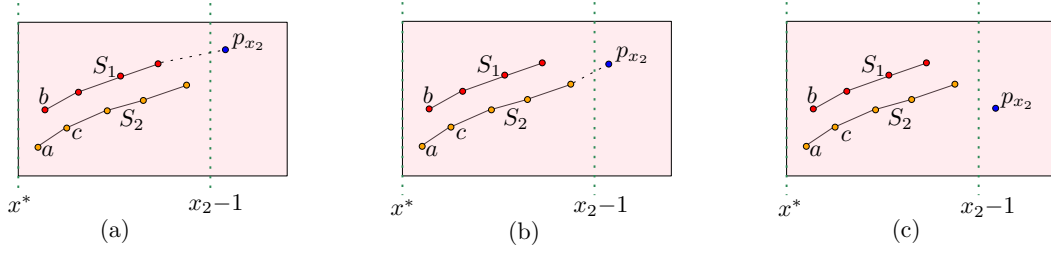
2.3 Idea-3: Pivot points and dynamic programming

Now the key task is to design a preprocessing algorithm which can quickly compute each entry in the two-dimensional arrays L and R . We will look at R and an analogous discussion will hold for L . Assume that we have computed the value of $R[x_2 - 1][\beta]$ and would like to next compute $R[x_2][\beta]$. Let $p_{x_2} = (x_2, a_{x_2})$ be the point with x -coordinate x_2 which is encountered when we move the vertical line from $x_2 - 1$ to x_2 .

Formally, we claim that the value of $R[x_2][\beta]$ is higher than $R[x_2 - 1][\beta]$ if and only if there is an increasing subsequence with the following three *properties*: (i) the subsequence length is β , (ii) the base of the subsequence is higher than $R[x_2 - 1][\beta]$, and (iii) the last point in the subsequence is p_{x_2} . Please refer to Figure 3 for an example.

We define p_{x_2} to be a *pivot point*. Now to efficiently compute $R[x_2][\beta]$, we will need the support of another two-dimensional array B defined as follows: among all increasing subsequences of length β in the range $(x^*, x_2]$ containing pivot point p_{x_2} as the last element, let S be the subsequence with the highest base. Then $B[x_2][\beta]$ is defined to be the value of

87:10 Range Longest Increasing Subsequence and Its Relatives



■ **Figure 3** Two sequences S_1 and S_2 lie to the left of the vertical line $x_2 - 1$. The value of $R[x_2 - 1][5]$ is realized by the y -coordinate of point a . In (a) and (b), we have $R[x_2][5] > R[x_2 - 1][5]$, and in (c), we have $R[x_2][5] = R[x_2 - 1][5]$. In case (a), point p_{x_2} stitches with S_1 to obtain a subsequence of length five with higher base than a . In case (b), by stitching point p_{x_2} with S_2 and removing point a , we obtain a subsequence of length five and it has a higher base than a .

the first element in S . In Figure 3(a) and Figure 3(b), $B[x_2][5]$ is realized by the y -coordinate of b and c , respectively. As a result, we can succinctly encode the three properties stated above as follows:

$$R[x_2][\beta] \leftarrow \max\{R[x_2 - 1][\beta], B[x_2][\beta]\}.$$

In other words, an increasing sequence of length β in the interval $(x^*, x_2]$ with the highest base will either contain the pivot element p_{x_2} or not. The first term (i.e., $R[x_2 - 1][\beta]$) captures the case where p_{x_2} is not included, whereas the second term (i.e., $B[x_2][\beta]$) captures the case where p_{x_2} is included. As such, the task of efficiently computing R has been *reduced* to the task of efficiently computing B .

2.4 Idea-4: 2D range-max data structure

The final step is to compute each entry in B in polylogarithmic time. The entries will be computed in increasing values of β . Assume that the entries corresponding to sequences of length at most $\beta - 1$ have been computed. Then the entries $B[x_2][\beta]$'s can be reduced to entries of $B[\cdot][\beta - 1]$'s as follows:

$$B[x_2][\beta] = \begin{cases} a_{x_2}, & \text{if } \beta = 1; \\ -\infty, & \text{if } a_i > a_{x_2} \text{ for all } x^* < i < x_2; \\ \max\{B[i][\beta - 1] \mid x^* < i < x_2 \text{ and } a_i < a_{x_2}\}, & \text{otherwise.} \end{cases}$$

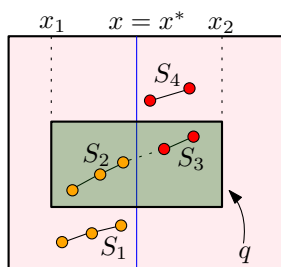
We will efficiently compute $B[x_2][\beta]$'s by constructing a data structure which can answer *2D range-max queries*. In a 2D range-max query, we are given n weighted points \mathcal{P} in 2D. Each point is associated with a weight. For each point $p_i = (i, a_i) \in \mathcal{P}$, given a query region of the form $q' = (-\infty, i) \times (-\infty, a_i)$, the goal is to report the point in $\mathcal{P} \cap q'$ with the maximum weight. The 2D range-max problem can be solved in $O(n \log n)$ time (via reduction to the so-called *2D orthogonal point location* problem [69]).

Now with each point $p_i \in \mathcal{P}$, we associate the weight $B[i][\beta - 1]$ and build a 2D range-max data structure. For each point $p_{x_2} = (x_2, a_{x_2}) \in \mathcal{P}$, the point in $\mathcal{P} \cap (-\infty, x_2) \times (-\infty, a_{x_2})$ with the maximum weight is reported. If point p_i is reported, then set $B[x_2][\beta] = B[i][\beta - 1]$. The correctness follows from the third case in the above dynamic programming. Since $\beta \leq \tau$, the time taken to construct B is $O(\tau) \times O(n \log n) = O(n\tau \log n)$.

Therefore, the overall time taken by this technique to answer *1D-Range-LIS* will be $\tilde{O}(m\tau + n\tau) + O(k) = \tilde{O}(m\sqrt{n} + n\sqrt{n}) + O(k)$ by setting $\tau \leftarrow \sqrt{n}$.

2.5 Idea-5: Generalizing L and R to handle $2D$ -Range-LIS

We briefly describe the challenge arising while handling $2D$ -Range-LIS where the queries are axis-parallel rectangles. The definition of highest bases and lowest peak fail to be directly useful. For example, see Figure 4, where $\text{LIS}(\mathcal{P} \cap q)$ is equal to five, and is realized by stitching S_2 and S_3 . However, $L[x_1][3]$ will correspond to the sequence S_1 and $R[x_2][2]$ will correspond to the sequence S_4 , both of which lie completely outside q .



■ **Figure 4** Four increasing subsequences lying in the range $[x_1, x_2] \times (-\infty, +\infty)$.

The key observation in the 2D setting is that for an increasing subsequence S and a query rectangle $q = [x_1, x_2] \times [y_1, y_2]$, if the first and the last point of S lie inside q , then all the points of S lie inside q . Accordingly, we generalize the definitions of L and R . Specifically, in the 2D setting, R takes two arguments, the query q and an integer β , and is defined as follows:

$$R(q, \beta) \leftarrow \max_{p_i=(i, a_i)} \{B[i][\beta] \mid p_i \in q, i > x^* \text{ and } B[i][\beta] > y_1\}.$$

For example, in Figure 4, the new definition of $R(q, 2)$ ensures that the sequence S_3 gets captured instead of S_4 . We construct L analogously. Note that in the 1D setting, L and R were *explicitly* computed. However, in the 2D setting, the number of combinatorially different axis-parallel rectangles is significantly more than our time budget. Therefore, L and R *cannot* be precomputed and stored. Instead, for all $1 \leq \alpha \leq \tau$ (resp., $1 \leq \beta \leq \tau$), we will compute $L(q, \alpha)$ (resp., $R(q, \beta)$) during the query algorithm in $O(\tau \cdot \text{polylog } n)$ time.

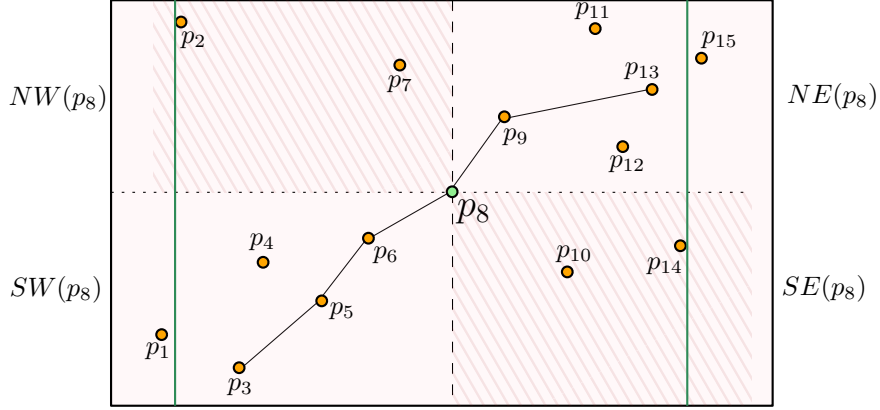
3 Our second technique: Handling large LIS

Now we give an overview of our second technique for the $1D$ -Range-LIS problem (which we show in the full version of the paper that it also extends to the $2D$ -Range-LIS problem). If $\text{LIS}(\mathcal{S} \cap q) \geq \tau$, then this technique will report the answer correctly with high probability.

3.1 Idea-1: Stitching elements

One challenge with the LIS problem is that it is *not decomposable*. Consider a query range $q = [x_1, x_2] \times (-\infty, +\infty)$ and an x^* such that $x_1 < x^* < x_2$. Let $q_\ell = [x_1, x^*] \times (-\infty, +\infty)$ and $q_r = (x^*, x_2] \times (-\infty, +\infty)$ such that $q = q_\ell \cup q_r$. Then the LIS of $\mathcal{P} \cap q$ need not be the concatenation of the LIS of $\mathcal{P} \cap q_\ell$ and the LIS of $\mathcal{P} \cap q_r$, since the rightmost point in the LIS of $\mathcal{P} \cap q_\ell$ might have a larger value than the leftmost point in the LIS of $\mathcal{S} \cap q_r$.

However, suppose an oracle reveals that point $p_i \in \mathcal{P}$ lies in the LIS of range $q = [x_1, x_2] \times (-\infty, +\infty)$. Then we claim that the problem becomes decomposable. We will define some notation before proceeding. For a point $p_i = (i, a_i) \in \mathcal{P}$ corresponding to an



■ **Figure 5** A collection of fifteen elements. Given a query range $q = [2, 14] \times (-\infty, +\infty)$, suppose an oracle reveals that p_8 lies in the LIS of $\mathcal{P} \cap q$. Then observe that we can discard the points in $SE(p_8)$ and $NW(p_8)$ which are shown as shaded regions. The LIS of $\mathcal{P} \cap q \cap NE(p_8)$ will be (p_8, p_9, p_{13}) and the LIS of $\mathcal{P} \cap q \cap SW(p_8)$ will be (p_3, p_5, p_6, p_8) . Therefore, LIS of $\mathcal{P} \cap q$ will be $(p_3, p_5, p_6, p_8, p_9, p_{13})$.

element $a_i \in \mathcal{S}$, define the *north-east* region of p_i as $NE(p_i) = \{(x, y) \mid x \geq i \text{ and } y \geq a_i\}$. Analogously, define the *north-west*, the *south-west* and the *south-east* region of p_i which are denoted by $NW(p_i)$, $SW(p_i)$ and $SE(p_i)$, respectively. Then it is easy to observe that,

$$\text{LIS}(\mathcal{P} \cap q) = \text{LIS}(\mathcal{P} \cap q \cap NE(p_i)) + \text{LIS}(\mathcal{P} \cap q \cap SW(p_i)) - 1. \quad (1)$$

See Figure 5 for an example. In other words, knowing that p_i belongs to the LIS decomposes the original LIS problem into two LIS sub-problems which can be computed independently. In such a case, we refer to point p_i as a *stitching element*, which is formally defined below.

► **Definition 5** (Stitching element). *For a range q , fix any LIS of $\mathcal{P} \cap q$ and call it S . Then each element in S is defined to be a stitching element w.r.t. q .*

The goal of our algorithm is to:

Construct a small-sized set $\mathcal{R} \subseteq \mathcal{P}$ such that, for any $q \in \mathcal{Q}$, at least one stitching element w.r.t. q is contained in $\mathcal{R} \cap q$.

For each $p_i \in \mathcal{R}$, in the preprocessing phase, the two terms on the right hand side of equation 1 can be computed in $O(n \log n)$ time for all possible queries. Therefore, the preprocessing time will be $O(|\mathcal{R}|n \log n)$.

3.2 Idea-2: Random sampling

By using the fact that $\text{LIS}(\mathcal{P} \cap q) \geq \tau$, it is possible to construct a set \mathcal{R} of size roughly $\frac{n \log n}{\tau} \approx \sqrt{n} \log n$ via random sampling. For each $1 \leq i \leq n$, sample $p_i \in \mathcal{P}$ independently with probability $\frac{c \log n}{\tau}$, where c is a sufficiently large constant. Let $\mathcal{R} \subseteq \mathcal{P}$ be the set of sampled points. Then we can establish the following connection between \mathcal{R} and the stitching elements.

► **Lemma 6.** *For all ranges q such that $\text{LIS}(\mathcal{P} \cap q) \geq \tau$, if S_q is one of the LIS of $\mathcal{P} \cap q$, then with high probability $|\mathcal{R} \cap S_q| = \Omega(\log n)$.*

This ensures that the preprocessing time will be $O(|\mathcal{R}|n \log n) = O(n^{3/2} \log^2 n)$. To answer any range q , first scan \mathcal{R} to identify the points which lie inside q . For each element $p_i \in \mathcal{R} \cap q$, compute the right hand side of equation 1 in $O(\log n)$ time. Finally, report the largest value computed. Therefore, the query time is bounded by $O(|\mathcal{Q}| \cdot |\mathcal{R}| \cdot \log n) = O(m\sqrt{n} \log^2 n)$.

4 Our third technique: Handling small cardinality colors

4.1 A naive approach

The *Colored-2D-Range-LIS* problem is more challenging than the *2D-Range-LIS* problem. Firstly, the result of Theorem 1 does not really help in answering *Colored-2D-Range-LIS* since it completely ignores the information about the colors. Secondly, there might be a temptation to solve the problem “independently” for each color class. For example, consider a setting where each color class has roughly equal number of points, i.e., $n/|\mathcal{C}|$ points. Consider a color c and build an instance of Theorem 1 based on the points of color c . Then, for each query $q \in \mathcal{Q}$, we have the value of $\text{LIS}(\mathcal{P}_c \cap q)$. Repeat this for each color in \mathcal{C} . Finally, for each query $q \in \mathcal{Q}$, report the color c for which $\text{LIS}(\mathcal{P}_c \cap q)$ is maximized.

Now lets (informally) analyze this algorithm. The running time bound of Theorem 1 has three terms. For now, lets ignore the second term and the third term, which represent the time taken to build the data structure and the time taken to report the output of the m queries. The first term is the time taken to answer m queries (which is roughly \sqrt{n} time per query). Adding the first term for each color class, we get

$$\sum_c \tilde{O}(m|\mathcal{P}_c|^{1/2}) \leq \tilde{O}(|\mathcal{C}|m(n/|\mathcal{C}|)^{1/2}) = \tilde{O}(m\sqrt{n|\mathcal{C}|}),$$

which is $\tilde{O}(mn)$ when $|\mathcal{C}| = \Theta(n)$ and $\tilde{O}(mn^{\frac{1+\varepsilon}{2}}) \gg m\sqrt{n}$ when $|\mathcal{C}| = n^\varepsilon$, for any $0 < \varepsilon \leq 1$. Therefore, in the worst-case this approach does not help achieve our target query time bound.

4.2 A technique to handle small cardinality colors

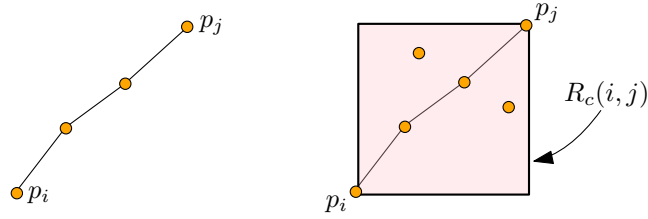
We will design a third technique which will be helpful to answer *Colored-1D-Range-LIS* and *Colored-2D-Range-LIS*. The key idea is to handle all the colors in a “combined” manner. The technique will work well when all the colors have small cardinality. Given a parameter Δ , assume that for each color c , the value of $|\mathcal{P}_c| \leq \Delta$. The key observations made by the algorithm are the following:

- *Bounding the number of output subsequences.* For a query q , let the output LIS be S of color c . Let $p_i \in \mathcal{P}_c$ (resp., $p_j \in \mathcal{P}_c$) be the first (resp., last) point on S . Call this a pair (i, j) . As such, whenever color c is the output, the number of distinct pairs will be $O(|\mathcal{P}_c|^2)$. Adding up over all the colors, the total number of distinct pairs will be:

$$\sum_c O(|\mathcal{P}_c|^2) \leq O\left(\Delta \sum_c |\mathcal{P}_c|\right) = O(n\Delta).$$

If $\Delta \ll n$, then this is a significant improvement over the naive bound of $O(n^2)$ distinct pairs (or distinct output subsequences).

- *Reduction to an uncolored problem.* For each possible output subsequence S of color c with $p_i \in \mathcal{P}_c$ (resp., $p_j \in \mathcal{P}_c$) as the first (resp., last) point point on S , we construct an axis-aligned rectangle $R_c(i, j)$ with p_i (resp., p_j) as the bottom-left (resp., top-right) corner. A weight $w(R_c(i, j))$ is associated with rectangle $R_c(i, j)$ which is equal to $\text{LIS}(\mathcal{P}_c \cap R_c(i, j))$.



■ **Figure 6** On the left is an increasing subsequence of length four with all four points being of color c . On the right is the corresponding rectangle $R_c(i, j)$. The LIS of color c inside $R_c(i, j)$ is four (the other two points of color c do not participate in the LIS). Therefore, the weight associated with $R_c(i, j)$ is four.

See Figure 6. Let \mathcal{R} be the collection of $O(n\Delta)$ such weighted rectangles. As a result, the problem of *Colored-2D-Range-LIS* is now reduced to the *rectangle range-max problem*, where given an axis-parallel rectangle q , among all the rectangles in \mathcal{R} which lie completely inside q , the goal is to report the rectangle with the largest weight.

It is crucial to note that the colored problem has been reduced to a problem on *uncolored* rectangles for which an efficient data structure exists: the rectangle range-max data structure can be constructed in $\tilde{O}(|\mathcal{R}|) = \tilde{O}(n\Delta)$ time and the m range queries can be answered in $\tilde{O}(m) + O(k)$ time.

5 Putting all the pieces together

As an illustration we will consider the *Colored-2D-Range-LIS* problem and put together all the three techniques discussed in the previous subsections in a specific manner.

5.1 Light and heavy colors

Define a parameter Δ which will be set later. A color c is classified as *light* if $|\mathcal{P}_c| \leq \Delta$, otherwise, it is classified as *heavy*. We will design different algorithms to handle light colors and heavy colors. The advantage with a light color, say c , is that we can precompute the LIS for $O(|\mathcal{P}_c|^2)$ 2D axis-parallel ranges and still be within the time budget. On the other hand, the advantage with heavy colors is that there can be only $O(n/\Delta)$ heavy colors.

5.2 Handling light colors

Let $\mathcal{P}_\ell \subseteq \mathcal{P}$ be the set of points which belong to a light color. We will use the third technique to answer *Colored-2D-Range-LIS* on \mathcal{P}_ℓ and queries \mathcal{Q} . It follows that the running time is $\tilde{O}(m + n\Delta) + O(k)$ (see the full version of the paper for details).

5.3 Handling heavy colors and small LIS queries

Let $\mathcal{P}_h \subseteq \mathcal{P}$ be the set of points which belong to a heavy color. In the full version of the paper we will prove that for an arbitrary value of τ , the running time of the first technique will be $\tilde{O}(m\tau + n\tau)$. The first technique as described above only handles uncolored points. In the full version of the paper we will “generalize” the first technique to answer *Colored-2D-Range-LIS* problem as well. The running time of this generalized first technique on colored pointset \mathcal{P}_h and queries \mathcal{Q} will be $\tilde{O}\left(\frac{m\tau n}{\Delta} + \frac{n^2\tau}{\Delta}\right) + O(k)$, where the number of heavy colors is $O(n/\Delta)$.

5.4 Handling heavy colors and large LIS queries

In the full version of the paper we will prove that for an arbitrary τ , the running time of the second technique will be $\tilde{O}\left(\frac{mn}{\tau} + \frac{n^2}{\tau}\right) + O(k)$ for *2D-Range-LIS* problem. In fact, we will generalize this algorithm to the large LIS case of *Colored-2D-Range-LIS* with the same running time (ignoring polylogarithmic factors). The generalized algorithm will answer *Colored-2D-Range-LIS* on \mathcal{P}_h and queries \mathcal{Q} .

Combining all the three subroutines, the total running time will be:

$$\underbrace{\tilde{O}\left(m + \frac{mn}{\tau} + \frac{m\tau n}{\Delta}\right)}_{\text{query time}} + \underbrace{\tilde{O}\left(\frac{n^2}{\tau} + n\Delta + \frac{n^2\tau}{\Delta}\right)}_{\text{preprocessing time}} + O(k)$$

We set the parameters $\tau \leftarrow n^{1/3}$ and $\Delta \leftarrow n^{2/3}$ in the above expression to obtain a running time of $\tilde{O}(mn^{2/3} + n^{5/3}) + O(k)$.

6 Open Problems

We conclude the paper with a few open problems.

- Is it possible to extend the algorithm in [29] for the *1D-Range-LIS* problem to solve the *2D-Range-LIS* problem in $O(n \text{ polylog } n + k)$ time? Or, is there a (conditional) hardness result which makes obtaining the upper bound unlikely?
- Another interesting research direction is to design a deterministic algorithm which runs in sub-quadratic time. Specifically, can the construction of stitching set be efficiently derandomized?
- For the *Colored-2D-Range-LIS* problem, can we bridge the gap between the upper bound and the (conditional) lower bound. We conjecture that the upper bound can be further improved.
- Can we extend our algorithmic technique to beat the quadratic barrier for the weighted version of *1D-Range-LIS*?
- Finally, it would be interesting to explore the *Range-LIS* in the dynamic model.

References

- 1 Amir Abboud, Nick Fischer, Zander Kelley, Shachar Lovett, and Raghu Meka. New graph decompositions and combinatorial boolean matrix multiplication algorithms. In *STOC*, 2024. To appear.
- 2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 434–443. IEEE, 2014. doi:10.1109/FOCS.2014.53.
- 3 Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting in three and higher dimensions. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 149–158, 2009. doi:10.1109/FOCS.2009.58.
- 4 Peyman Afshani, Gerth Stolting Brodal, and Norbert Zeh. Ordered and unordered top-k range reporting in large data sets. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 390–400, 2011. doi:10.1137/1.9781611973082.31.
- 5 Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. *Advances in Discrete and Computational Geometry*, pages 1–56, 1998.
- 6 Pankaj K. Agarwal, Sathish Govindarajan, and S. Muthukrishnan. Range searching in categorical data: Colored range searching on grid. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 17–28, 2002. doi:10.1007/3-540-45749-6_6.

- 7 Miklós Ajtai, TS Jayram, Ravi Kumar, and D Sivakumar. Approximate counting of inversions in a data stream. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 370–379, 2002.
- 8 Michael H Albert, Alexander Golynski, Angèle M Hamel, Alejandro López-Ortiz, S Srinivasa Rao, and Mohammad Ali Safari. Longest increasing subsequences in sliding windows. *Theoretical Computer Science*, 321(2-3):405–414, 2004. doi:10.1016/J.TCS.2004.03.057.
- 9 Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.
- 10 Alexandr Andoni, Negev Shekel Nosatzki, Sandip Sinha, and Clifford Stein. Estimating the longest increasing subsequence in nearly optimal time. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 708–719, 2022. doi:10.1109/FOCS54457.2022.00073.
- 11 Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003. doi:10.1007/S00453-003-1021-X.
- 12 Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Graph expansion and communication costs of fast matrix multiplication. *Journal of the ACM (JACM)*, 59(6):1–23, 2013. doi:10.1145/2395116.2395121.
- 13 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.
- 14 Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios K. Tsakalidis. New upper bounds for generalized intersection searching problems. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 464–474, 1995. doi:10.1007/3-540-60084-1_97.
- 15 Gerth Stølting Brodal. External memory three-sided range reporting and top- k queries with sublogarithmic updates. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 47, pages 23:1–23:14. 2016. doi:10.4230/LIPIcs.STACS.2016.23.
- 16 Gerth Stølting Brodal, Pooya Davoodi, and S. Srinivasa Rao. On space efficient two dimensional range minimum data structures. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 171–182, 2010. doi:10.1007/978-3-642-15781-3_15.
- 17 Gerth Stølting Brodal, Rolf Fagerberg, Mark Greve, and Alejandro Lopez-Ortiz. Online sorted range reporting. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 173–182, 2009. doi:10.1007/978-3-642-10631-6_19.
- 18 Gerth Stølting Brodal, Beat Gfeller, Allan Gronlund Jorgensen, and Peter Sanders. Towards optimal range medians. *Theoretical Computer Science*, 412(24):2588–2601, 2011. doi:10.1016/J.TCS.2010.05.003.
- 19 Gerth Stølting Brodal and Kasper Green Larsen. Optimal planar orthogonal skyline counting queries. In *Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 110–121, 2014. doi:10.1007/978-3-319-08404-6_10.
- 20 Gerth Stølting Brodal and Konstantinos Tsakalidis. Dynamic planar range maxima queries. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 256–267, 2011.
- 21 Timothy M Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T Wilkinson. Linear-space data structures for range mode query in arrays. *Theory of Computing Systems*, 55(4):719–741, 2014. doi:10.1007/S00224-013-9455-2.
- 22 Timothy M. Chan, Qizheng He, and Yakov Nekrich. Further results on colored range searching. In *International Symposium on Computational Geometry (SoCG)*, pages 28:1–28:15, 2020. doi:10.4230/LIPIcs.SOCG.2020.28.
- 23 Timothy M. Chan and Zhengcheng Huang. Dynamic colored orthogonal range searching. In *Proceedings of European Symposium on Algorithms (ESA)*, volume 204, pages 28:1–28:13, 2021. doi:10.4230/LIPIcs.ESA.2021.28.

- 24 Timothy M. Chan, Kasper Green Larsen, and Mihai Patrascu. Orthogonal range searching on the ram, revisited. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 1–10, 2011. doi:10.1145/1998196.1998198.
- 25 Timothy M. Chan and Yakov Nekrich. Better data structures for colored orthogonal range reporting. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 627–636, 2020. doi:10.1137/1.9781611975994.38.
- 26 Funda Ergun and Hossein Jowhari. On distance to monotonicity and longest increasing subsequence of a data stream. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 730–736, 2008.
- 27 Michael L Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975. doi:10.1016/0012-365X(75)90103-X.
- 28 Anna Gál and Parikshit Gopalan. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. *SIAM Journal on Computing*, 39(8):3463–3479, 2010. doi:10.1137/090770801.
- 29 Pawel Gawrychowski, Egor Gorbachev, and Tomasz Kociumaka. Core-sparse monge matrix multiplication: Improved algorithm and applications. *CoRR*, abs/2408.04613, 2024. doi:10.48550/arXiv.2408.04613.
- 30 Paweł Gawrychowski and Wojciech Janczewski. Conditional lower bounds for variants of dynamic LIS. *arXiv preprint arXiv:2102.11797*, 2021.
- 31 Pawel Gawrychowski and Wojciech Janczewski. Fully dynamic approximation of LIS in polylogarithmic time. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 654–667, 2021. doi:10.1145/3406325.3451137.
- 32 Parikshit Gopalan, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Estimating the sortedness of a data stream. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 318–327, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283417>.
- 33 Prosenjit Gupta, Ravi Janardan, Saladi Rahul, and Michiel H. M. Smid. Computational geometry: Generalized (or colored) intersection searching. In *Handbook of Data Structures and Applications*, CRC Press, 2nd edition, pages 1042–1057, 2018.
- 34 Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Further results on generalized intersection searching problems: Counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995. doi:10.1006/JAGM.1995.1038.
- 35 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30, 2015. doi:10.1145/2746539.2746609.
- 36 Xiaocheng Hu, Miao Qiao, and Yufei Tao. Independent range sampling. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 246–255, 2014. doi:10.1145/2594538.2594545.
- 37 Ravi Janardan and Mario A. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry and Applications*, 3(1):39–69, 1993. doi:10.1142/S021819599300004X.
- 38 Ruoming Jin, Scott McCallen, and Eivind Almaas. Trend motif: A graph mining approach for analysis of dynamic complex networks. In *Proceedings of International Conference on Management of Data (ICDM)*, pages 541–546, 2007. doi:10.1109/ICDM.2007.92.
- 39 Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Counting colors in boxes. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 785–794, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283467>.
- 40 Haim Kaplan, Micha Sharir, and Elad Verbin. Colored intersection searching via sparse rectangular matrix multiplication. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 52–60, 2006. doi:10.1145/1137856.1137866.

- 41 Marek Karpinski and Yakov Nekrich. Top-k color queries for document retrieval. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 401–411, 2011. doi:10.1137/1.9781611973082.32.
- 42 Masashi Kiyomi, Hirotaka Ono, Yota Otachi, Pascal Schweitzer, and Jun Tarui. Space-efficient algorithms for longest increasing subsequence. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, 2018.
- 43 Tomasz Kociumaka and Saeed Seddighin. Improved dynamic algorithms for longest increasing subsequence. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 640–653, 2021. doi:10.1145/3406325.3451026.
- 44 Danny Krizanc, Pat Morin, and Michiel H. M. Smid. Range mode and range median queries on lists and trees. *Nordic Journal of Computing*, 12(1):1–17, 2005.
- 45 Ying Kit Lai, Chung Keung Poon, and Benyun Shi. Approximate colored range and point enclosure queries. *Journal of Discrete Algorithms*, 6(3):420–432, 2008. doi:10.1016/J.JDA.2007.10.001.
- 46 Kasper Green Larsen and Rasmus Pagh. I/O-efficient data structures for colored range and prefix reporting. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 583–592, 2012. doi:10.1137/1.9781611973099.49.
- 47 Kasper Green Larsen and Freek van Walderveen. Near-optimal range reporting structures for categorical data. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 265–276, 2013. doi:10.1137/1.9781611973105.20.
- 48 Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *Journal of the ACM (JACM)*, 49(1):1–15, 2002. doi:10.1145/505241.505242.
- 49 Youhuan Li, Lei Zou, Huaming Zhang, and Dongyan Zhao. Longest increasing subsequence computation over streaming sequences. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 30(6):1036–1049, 2017. doi:10.1109/TKDE.2017.2761345.
- 50 David Liben-Nowell, Erik Vee, and An Zhu. Finding longest increasing and common subsequences in streaming data. *Journal of Combinatorial Optimization*, 11:155–175, 2006. doi:10.1007/S10878-006-7125-X.
- 51 Colin L Mallows. Patience sorting. *SIAM Review*, 4(2):148–149, 1962.
- 52 Colin L Mallows. Patience sorting. *SIAM Review*, 5(4):375, 1963.
- 53 Michael Mitzenmacher and Saeed Seddighin. Dynamic algorithms for LIS and distance to monotonicity. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 671–684, 2020. doi:10.1145/3357713.3384240.
- 54 S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 657–666, 2002. URL: <http://dl.acm.org/citation.cfm?id=545381.545469>.
- 55 Yakov Nekrich. Efficient range searching for categorical and plain data. *ACM Transactions on Database Systems (TODS)*, 39(1):9, 2014.
- 56 Yakov Nekrich and Saladi Rahul. 4d range reporting in the pointer machine model in almost-optimal time. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1862–1876, 2023. doi:10.1137/1.9781611977554.CH71.
- 57 Yakov Nekrich and Jeffrey Scott Vitter. Optimal color range reporting in one dimension. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 743–754, 2013. doi:10.1007/978-3-642-40450-4_63.
- 58 Ilan Newman and Nithin Varma. New sublinear algorithms and lower bounds for LIS estimation. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 100:1–100:20, 2021. doi:10.4230/LIPIcs.ICALP.2021.100.
- 59 Manish Patil, Sharma V. Thankachan, Rahul Shah, Yakov Nekrich, and Jeffrey Scott Vitter. Categorical range maxima queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 266–277, 2014. doi:10.1145/2594538.2594557.

- 60 Saladi Rahul. Approximate range counting revisited. In *33rd International Symposium on Computational Geometry (SoCG)*, volume 77, pages 55:1–55:15, 2017. doi:10.4230/LIPIcs.SOCG.2017.55.
- 61 Saladi Rahul. Approximate range counting revisited. *Journal of Computational Geometry*, 12(1):40–69, 2021. doi:10.20382/JOCG.V12I1A3.
- 62 Saladi Rahul and Ravi Janardan. Algorithms for range-skyline queries. In *Proceedings of ACM Symposium on Advances in Geographic Information Systems (GIS)*, pages 526–529, 2012. doi:10.1145/2424321.2424406.
- 63 Saladi Rahul and Yufei Tao. On top-k range reporting in 2d space. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 265–275, 2015. doi:10.1145/2745754.2745777.
- 64 Saladi Rahul and Yufei Tao. Efficient top-k indexing via general reductions. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 277–288, 2016. doi:10.1145/2902251.2902290.
- 65 Saladi Rahul and Yufei Tao. A guide to designing top-k indexes. *SIGMOD Record*, 48(2):6–17, 2019. doi:10.1145/3377330.3377332.
- 66 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61:389–401, 2011. doi:10.1007/S00453-010-9401-5.
- 67 Aviad Rubinfeld, Saeed Seddighin, Zhao Song, and Xiaorui Sun. Approximation algorithms for lcs and LIS with truly improved running times. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1121–1145, 2019. doi:10.1109/FOCS.2019.00071.
- 68 Michael Saks and C Seshadhri. Estimating the longest increasing sequence in polylogarithmic time. *SIAM Journal of Computing*, 46(2):774–823, 2017. doi:10.1137/130942152.
- 69 Neil Sarnak and Robert Endre Tarjan. Planar point location using persistent search trees. *Communications of the ACM (CACM)*, 29(7):669–679, 1986. doi:10.1145/6138.6151.
- 70 Giorgio Satta. Tree-adjointing grammar parsing and boolean matrix multiplication. *Computational linguistics*, 20(2):173–191, 1994.
- 71 Qingmin Shi and Joseph JáJá. Optimal and near-optimal algorithms for generalized intersection reporting on pointer machines. *Information Processing Letters (IPL)*, 95(3):382–388, 2005. doi:10.1016/J.IPL.2005.04.008.
- 72 Xiaoming Sun and David P Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 336–345, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283419>.
- 73 Yufei Tao. Algorithmic techniques for independent query sampling. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 129–138, 2022. doi:10.1145/3517804.3526068.
- 74 Alexander Tiskin. Semi-local longest common subsequences in subquadratic time. *Journal of Discrete Algorithms*, 6(4):570–581, 2008. doi:10.1016/J.JDA.2008.07.001.
- 75 Alexander Tiskin. Fast distance multiplication of unit-monge matrices. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1287–1296, 2010. doi:10.1137/1.9781611973075.103.
- 76 Alexandre Tiskin. Semi-local string comparison: Algorithmic techniques and applications. *Math. Comput. Sci.*, 1(4):571–603, 2008. doi:10.1007/S11786-007-0033-3.
- 77 Jie Xue. Colored range closest-pair problem under general distance functions. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 373–390, 2019. doi:10.1137/1.9781611975482.24.
- 78 Jie Xue, Yuan Li, Saladi Rahul, and Ravi Janardan. Searching for the closest-pair in a query translate. *Journal of Computational Geometry*, 11(2):26–61, 2020. doi:10.20382/JOCG.V11I2A3.

87:20 Range Longest Increasing Subsequence and Its Relatives

- 79 Jie Xue, Yuan Li, Saladi Rahul, and Ravi Janardan. New bounds for range closest-pair problems. *Discrete & Computational Geometry*, 68(1):1–49, 2022. doi:10.1007/S00454-022-00388-7.
- 80 Hongyu Zhang. Alignment of blast high-scoring segment pairs based on the longest increasing subsequence algorithm. *Bioinformatics*, 19(11):1391–1396, 2003. doi:10.1093/BIOINFORMATICS/BTG168.