A Sub-Quadratic Algorithm for the Longest Common Increasing Subsequence Problem

Lech Duraj
Theoretical Computer Science, Faculty of Mathematics and Computer Science,
Jagiellonian University, Kraków, Poland
duraj@tcs.uj.edu.pl

Abstract

The Longest Common Increasing Subsequence problem (LCIS) is a natural variant of the celebrated Longest Common Subsequence (LCS) problem. For LCIS, as well as for LCS, there is an $\mathcal{O}\left(n^2\right)$-time algorithm and a SETH-based conditional lower bound of $\mathcal{O}\left(n^{2-\varepsilon}\right)$. For LCS, there is also the Masek-Paterson $\mathcal{O}\left(n^2/\log n\right)$-time algorithm, which does not seem to adapt to LCIS in any obvious way. Hence, a natural question arises: does any (slightly) sub-quadratic algorithm exist for the Longest Common Increasing Subsequence problem? We answer this question positively, presenting a $\mathcal{O}\left(n^2/\log^a n\right)$-time algorithm for $a = \frac{1}{6} - o(1)$. The algorithm is not based on memorizing small chunks of data (often used for logarithmic speedups, including the "Four Russians Trick" in LCS), but rather utilizes a new technique, bounding the number of significant symbol matches between the two sequences.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases longest common increasing subsequence, log-shaving, matching pairs

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.41

Related Version A full version of the paper is available at `https://arxiv.org/abs/1902.06864`.

Funding *Lech Duraj*: This work was partially supported by Polish National Science Center grant 2016/21/B/ST6/ 02165.

Acknowledgements I would like to thank Grzegorz Guśpiel, Grzegorz Herman and Adam Polak for helpful discussions and proofreading the paper, as well as the reviewers for many insightful comments and suggestions.

1 Introduction

The Longest Common Increasing Subsequence problem (LCIS) is a variant of the well-known and extensively studied Longest Common Sequence (LCS) problem. The LCS is formulated as follows: given two integer sequences $A = (A[1], \ldots, A[n])$ and $B = (B[1], \ldots, B[n])$, determine another sequence C which is a subsequence of both A and B, of maximal possible length. In the LCIS variant, we require C to be a strictly increasing subsequence.

For LCS, a simple algorithm working in $\mathcal{O}\left(n^2\right)$-time was published in 1974 by Wagner and Fischer [27]. The complexity was later brought down to $\mathcal{O}\left(\frac{n^2}{\log n}\right)$ (for constant alphabet size) by Masek and Paterson, using a technique informally called the "Four Russians trick" [21]. Some improvements have been made since then (in particular, [14] shaves another logarithm, down to $\mathcal{O}\left(\frac{n^2 \log\log n}{\log^2 n}\right)$ even with arbitrary alphabet size), but no truly sub-quadratic, $\mathcal{O}\left(n^{2-\epsilon}\right)$-time algorithm has been found. There is even substantial evidence that a better algorithm might in fact not exist: it was shown by Abboud, Backurs and Vassilevska-Williams [1], as well as by Bringmann and Künnemann [8] that a truly sub-quadratic algorithm for LCS would yield a $2^{\delta n}$-time algorithm for SAT, with some $\delta < 1$, thus refuting the Strong Exponential Time Hypothesis (which states, roughly speaking, that such an algorithm is impossible [16, 17]). Hence, if we believe that SETH is true, then we must accept that no fast algorithms for LCS will ever be found. It is worth noting that in recent years several other SETH-based quadratic-time bounds were also shown, e.g., [6] and [24].

As for LCIS, it is arguably one of the most interesting variants of LCS: neither of these problems seems to be reducible to the other (unless we count the reduction of LCIS to 3-sequence LCS [18], which does not seem strong enough to have meaningful consequences). Therefore no algorithm or hardness result for LCS can be easily translated to a corresponding result for LCIS. The "obvious" dynamic programming algorithm for LCIS is $\mathcal{O}\left(n^3\right)$, the first $\mathcal{O}\left(n^2\right)$-time algorithm was given in [31], and possibly the simplest one was explicitly stated in [32]. A conditional lower bound was proven in [13]: it turns out that, as for LCS, any $\mathcal{O}\left(n^{2-\varepsilon}\right)$-time algorithm for LCIS would refute the Strong Exponential Time Hypothesis. The proof is based, like the one in [1], on a reduction from the Orthogonal Vectors problem (introduced in [28]), but the reduction itself needs a quite different gadget construction. It is also worth mentioning that the problem of Longest Common Weakly Increasing Subsequence, similar to LCIS but with only weak monotonicity required, also has a conditional quadratic lower bound [23]. LCWIS, unlike LCIS, is also non-trivial when restricted to constant-size alphabets [19, 12]. It still remains an open question whether LCWIS admits a sub-quadratic algorithm for any alphabet size greater than 3.

The LCIS problem itself has been studied quite extensively, and other algorithms have been proposed: Sakai [25] found an algorithm which can retrieve the LCIS in linear space, Kutz et al. [19] presented an algorithm that works in (roughly speaking) $\mathcal{O}(n \cdot d)$ time, where d is the output size (i.e. the length of LCIS). Chan et al. [11] proved that LCIS can be found in $\mathcal{O}(r \log\log n)$, where r is the number of *matching pairs* of symbols (i.e. the pairs (x, y) with $A[x] = B[y]$). These algorithms work much faster for some specific cases (for example, they are sub-quadratic for "random" inputs with a reasonable notion of "randomness"), but no algorithm that achieves $o\left(n^2\right)$ worst-case complexity has been given so far. Arguably, one of the reasons is that the "Four Russians Trick" does not seem to adapt to current dynamic-programming LCIS algorithms – at least, not in any easy way. In light of known conditional lower bound of this problem, we can only hope for complexity similar to $\mathcal{O}\left(\frac{n^2}{\log^a n}\right)$ for some $a > 0$, but achieving this would seem interesting enough. "The Art of Shaving Logs", as called by Timothy M. Chan [9], has already been practised for a variety of problems [7, 10, 29, 20, 15, 30], sometimes yielding surprising results – for example, some remarkable consequences in circuit complexity [3, 2]. Therefore, it appears natural to ask the question: *Is there any slightly sub-quadratic (i.e. $o\left(n^2\right)$-time) algorithm for LCIS?*

This paper gives a positive answer to this question, by presenting an $\mathcal{O}\left(\frac{n^2(\log\log n)^2}{\log^{1/6} n}\right)$-time algorithm for LCIS. Our algorithm iterates over matching pairs of symbols (as the one in [11]), but to achieve sub-quadratic time, a new „log-shaving" technique is introduced: we do not try to precompute the results for small chunks of data, as in LCS algorithms. Instead, we choose a useful subset of matching pairs – so-called *significant pairs* – prove that there are $o\left(n^2\right)$ such pairs, and adapt the algorithm to exploit this fact.

2 Basic notions and paper outline

Let A and B be the input sequences – for most of the paper, it is convenient to allow A and B to have different lengths. Later, for the final complexity results, we will assume $|A| = |B|$. We use array-like notation for elements of A and B, i.e. $A = (A[1], A[2], \ldots)$, $B = (B[1], B[2], \ldots)$. We will refer to the elements of A and B as *symbols*, remembering that the symbols are in fact integers, and thus can be compared with each other. Also, we may assume that all those integers are positive and not exceeding $\mathcal{O}(|A| + |B|)$ – if not, we can rename all the elements to be in range $\{1, 2, \dots, |A| + |B|\}$, while preserving their relative order.

▶ **Definition 1** (Matching pair). *A pair of indices (x, y) for some $1 \leq x \leq |A|$, $1 \leq y \leq |B|$ is a* matching pair *if $A[x] = B[y]$. For $\sigma = A[x] = B[y]$, we can say that (x, y) is a σ-matching pair, or simply a σ-pair. We also say that σ is the symbol of (x, y) and sometimes write $\sigma = symbol(x, y)$.*

▶ **Definition 2** (Orders on pairs). *Let (x, y) and (x', y') be matching pairs.*

(1) *We say that $(x, y) \leq (x', y')$ if $x \leq x'$ and $y \leq y'$.*

(2) *We say that $(x, y) \prec (x', y')$ if $x < x'$, $y < y'$ and $symbol(x, y) < symbol(x', y')$.*

▶ **Definition 3** (Common increasing subsequence). *A common increasing sequence of A and B is a sequence of matching pairs $(x_1, y_1), \dots, (x_s, y_s)$ such that $(x_1, y_1) \prec \dots \prec (x_s, y_s)$.*

Our main problem is to find the longest possible common increasing subsequence. Sometimes, we wish to consider only some prefixes of A and B, for which we will need the following two definitions:

▶ **Definition 4.** *For any $x \leq |A|$ and $y \leq |B|$, we define $lcis(x, y)$ as the maximal possible length of a common increasing subsequence that ends with some (x', y') with $x' \leq x$ and $y' \leq y$. In other words $lcis(x, y)$ is the length of the longest common increasing subsequence of $A[1..x]$ and $B[1..y]$.*

▶ **Definition 5.** *For any matching pair (x, y), we define $lcis^{\rightarrow}(x, y)$ as the maximal possible length of a common increasing subsequence that ends with (x, y).*

▶ Remark 6. The value of $lcis(x, y)$ is equal to $\max_{(x',y') \leq (x,y)} lcis^{\rightarrow}(x', y')$. In particular, for any $(x', y') \leq (x, y)$ we have $lcis^{\rightarrow}(x', y') \leq lcis(x, y)$.

A sequence realizing $lcis^{\rightarrow}(x, y)$ must have some pair (x', y') as the next-to-last element (providing that $lcis^{\rightarrow}(x, y) \geq 2$). Clearly, $lcis^{\rightarrow}(x', y') = lcis^{\rightarrow}(x, y) - 1$. We call such a pair the *predecessor* of (x, y). There may be multiple candidates for the predecessor, so we break the ties first by y, then by x. Formally:

▶ **Definition 7** (Predecessor). *For a matching pair (x, y) the predecessor $\pi(x, y)$ is a matching pair $(x', y') \prec (x, y)$ such that:*

(1) *$lcis^{\rightarrow}(x', y') = lcis^{\rightarrow}(x, y) - 1$,*

(2) *(x', y') has the minimal possible y' of all pairs satisfying (1),*

(3) *(x', y') has the minimal possible x' of all pairs satisfying (1) and (2).*

An example is shown in Figure 1 below:

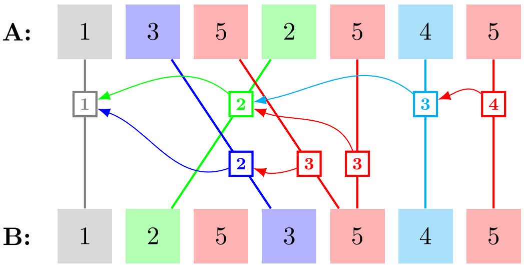

Figure 1 An example: for two sequences $A = (1, 3, 5, 2, 5, 4, 5)$ and $B = (1, 2, 5, 3, 5, 4, 5)$ some matching pairs are shown. A pair (x, y) is labeled with $lcis^{\rightarrow}(x, y)$ and an arrow leads from (x, y) to $\pi(x, y)$. Some pairs were omitted for clarity.

▶ **Definition 8.** *For a matching pair (x, y) with $lcis^{\rightarrow}(x, y) > k$, the k-th predecessor $\pi^k(x, y)$ is defined inductively as (x, y) for $k = 0$ and $\pi(\pi^{k-1}(x, y))$ for $k \geq 1$. In particular, $\pi^1(x, y) = \pi(x, y)$.*

The algorithm for LCIS in [11] iterates over all matching pairs in A and B. There may be, however, as many as $\Theta(n^2)$ of them – it is easy to construct an example of such sequences by including a lot of equal elements. Observe, though, that some of the matching pairs may not really matter in the solution – for example, if $A[x + 1] = A[x]$, then a matching pair $(x + 1, y)$ for any y is as good as (x, y), and we could drop $A[x + 1]$ from A altogether. We generalize this observation to form the notion of a *significant pair*, which is the central concept of this paper, allowing us to construct the desired faster algorithm for LCIS.

▶ **Definition 9** (Significant pair). *Let (x, y) be a σ-pair, i.e. $\sigma = A[x] = B[y]$. We say that (x, y) is a significant pair if for every σ-pair $(x', y') \leq (x, y)$, if $(x', y') \neq (x, y)$ then $lcis^{\rightarrow}(x', y') < lcis^{\rightarrow}(x, y)$.*

Again, we include an example to make this important definition more clear:

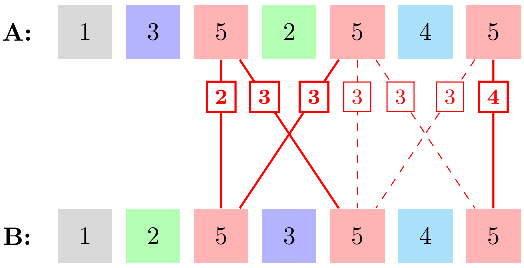

Figure 2 An example of two sequences A and B with some matching pairs. A pair (x, y) is labeled with $lcis^{\rightarrow}(x, y)$; the significant pairs are drawn with solid lines, while the insignificant ones – with dashed lines. Some pairs were ommitted for clarity.

▷ Claim 10. If (x, y) is a matching pair, then $\pi(x, y)$, if exists, is a significant pair.

Proof. Easy from the tie-breaking rule in the predecessor definition: let $(x', y') = \pi(x, y)$. If (x', y') is not significant, then there is a better candidate for the predecessor. ◁

Having defined the significant pairs, we propose the following two theorems, which together form our main result. The first bounds the number of such pairs, the second proposes an algorithm that exploits this bound:

▶ **Theorem 11.** *For any A, B with $|A|, |B| \leq n$, the number of significant pairs is at most $\mathcal{O}\left(\frac{n^2}{\log^{1/3} n}\right)$.*

▶ **Theorem 12.** *Suppose that $|A|, |B| \leq n$ and that there are at most $\mathcal{O}\left(\frac{n^2}{t}\right)$ significant pairs, with $t = t(n)$ satisfying $\log t = \Theta(\log\log n)$. There is an algorithm which finds LCIS in $\mathcal{O}\left(\frac{n^2(\log\log n)^2}{\sqrt{t}}\right)$ time complexity.*

The obvious consequence of Theorems 11 and 12 is the following:

▶ **Corollary 13.** *There is an algorithm which finds LCIS of two sequences A, B with $|A|, |B| \leq n$ in $\mathcal{O}\left(\frac{n^2(\log\log n)^2}{\log^{1/6} n}\right)$ time complexity.*

The rest of the paper is devoted to proving the two main theorems: in Section 3 we prove Theorem 11, whereas Section 4 describes the algorithm of Theorem 12.

3 Counting significant pairs

3.1 The idea

In this section we present the high-level idea behind the bound for the number of significant pairs. Please note that while the proof originally stems from this concept, its final version needs also some careful counting and balancing arguments, as well as a few non-intuitive tricks. Therefore, we start with an informal sketch to give some intuitions, while the full, formal proof will be presented in Sections 3.2-3.4.

Imagine two sequences A and B with $|A| = |B| = n$ and with the number of significant pairs „very close" to $\Theta\left(n^2\right)$. This requires at least one symbol σ generating a lot of significant σ-pairs itself (as opposed to, for example, $\sqrt{n}$ different symbols generating $\Theta\left(n^{3/2}\right)$ pairs each – this is impossible, as it would imply $|A| > n$ or $|B| > n$). We then focus on one particular such symbol σ and imagine a graph G_σ with all occurrences of σ in A and B as vertices of G_σ, and all significant σ-pairs as its edges. (An example is already provided with Figure 2 – for $\sigma = 5$, the red elements of A and B are the vertices of G_σ, and red solid lines are the edges).

Denote by $s = s(n)$ the largest integer such that G_σ has at least s vertices of degree at least s – the total number of edges cannot exceed $\mathcal{O}(n \cdot s)$, so simplifying a little bit, our goal is to show that $s = o(n)$. But every σ-pair must have its predecessors, which are τ-pairs for some other symbols $\tau < \sigma$. The proof is based on the observation that these predecessors need quite a lot of different symbols – we argue that the total number of required elements of A and B is asymptotically greater than s. This forces $s = o(n)$, and after careful calculations we obtain more specific bounds.

To take a closer (but still preliminary) look at our main tools, consider a vertex $A[x]$ in G_σ with edges $(x, y_1), \ldots, (x, y_s)$ with $y_1 > y_2 > \ldots > y_s$. Consider, for a fixed $k > 0$, all predecessors $\pi^k(x, y_i)$ for $1 \leq i \leq s$ (let us not worry, for the moment, whether the predecessors exist), denoting $\pi^k(x, y_i) = (u_i, v_i)$. We claim that for every i, $v_{i+k} < v_i$. This is because $v_{i+k} < y_{i+k}$, while $v_i < y_{i+k}$ would lead to $(u_i, v_i) \prec (x, y_{i+k})$, which would in turn yield $lcis^{\rightarrow}(x, y_{i+k}) \geq lcis^{\rightarrow}(u_i, v_i) + 1 = lcis^{\rightarrow}(x, y_i) - k + 1$. But the values $lcis^{\rightarrow}(x, y_i), \ldots, lcis^{\rightarrow}(x, y_{i+k})$ must all be different (as the pairs are significant), which implies $lcis^{\rightarrow}(x, y_{i+k}) \leq lcis^{\rightarrow}(x, y_i) - k$ – a contradiction. Therefore $v_i > y_{i+k} > v_{i+k}$. This shows that there must be at least $\frac{s}{k}$ different elements in B to accommodate the k-th predecessors of the pairs (x, y_i) for $i = 1, 2, \ldots, s$. A notable edge case is that for every k, all those predecessors happen to use the same symbol τ_k, implying at least $\frac{s}{k}$ occurrences of τ_k in B and thus at least $s + s/2 + s/3 + \ldots = \Omega\left(s \log s\right)$ symbols in B, which would immediately imply $s = \mathcal{O}\left(n/\log n\right)$. Of course, we cannot hope to be that fortunate, but we can salvage some bounds from this argument: if, for some $\delta = \delta(n)$, which is $\omega(1)$ and $o\left(\log n\right)$, and for any particular $x \in A$, the k-th predecessors use at most δ different symbols for every k, we can prove that this yields $\Omega\left(\frac{s \log s}{\delta}\right)$ symbols in B, so $s = \mathcal{O}\left(\frac{n\delta}{\log n}\right) = o\left(n\right)$ and we are done – this is formally proven in more general form in Section 3.4. On the other hand, if there are more than δ different symbols among predecessors for every possible x, we expect δ different elements before x in A. Using similar arguments as before, we argue that those sets of elements must be (at least partly) disjoint for different picks of x. But there are $\Omega\left(s\right)$ possible choices of a high-degree vertex x, which also implies $s = o\left(n\right)$ – this sketch roughly corresponds to Section 3.3.

We now move on to the full proof. Before presenting the core observations, we start with padding the sequences – adding dummy elements to make computing predecessors easier.

3.2 Preliminaries – padding the sequences

Consider two sequences A, B with $|A|, |B| \leq n$. Suppose that $lcis(|A|, |B|) \geq 1$ – if there are no common elements in A and B, there is nothing to be proven. We can also assume $n \geq 2$. For the sake of analyzing significant pairs between A and B, we shall modify the sequences a little bit. First, we can assume, without loss of generality, that A and B contain only positive integers (adding any constant to all the elements does not change anything). Then we pad both the sequences, inserting a prefix of dummy elements $P_n = (-2n, -2n+1, -2n+2, \ldots, -1)$ in the front, obtaining new sequences $\hat{A}$ and $\hat{B}$. More precisely, we put $\hat{A} = P_n \circ A$ and $\hat{B} = P_n \circ B$, where $\circ$ is the operator of sequence concatenation. These new elements now contribute to all previous common increasing subsequences, increasing their lengths by exactly $2n$. This does not change the significance of any "old" matching pairs, i.e. any significant pair (x, y) present in A and B remains a significant pair $(2n + x, 2n + y)$ in $\hat{A}$ and $\hat{B}$. Therefore the number of significant pairs can only increase in this operation, so it is enough to prove the inequality of Theorem 11 for $\hat{A}$ and $\hat{B}$. The padding operation also ensures that for every matching pair (x, y) with $x, y > 2n$ we have $lcis^{\rightarrow}(x, y) > 2n$, allowing us to compute up to $2n$ predecessors of (x, y).

3.3 The σ-pair graph

First, let $\delta = \delta(n) = \frac{\log^{1/3} n}{4}$ – our goal is to bound the number of significant pairs by $\mathcal{O}\left(\frac{n^2}{\delta(n)}\right)$, and the order of magnitude of δ is chosen as „the highest one for which the proof still holds".

The crucial step of the proof starts with fixing a symbol σ and bounding only the number of significant σ-pairs, which we will then sum up over all possible σ. Without loss of generality we remove – for a while – all elements greater than σ from $\hat{A}$ and $\hat{B}$, as they do not affect the significance of any σ-pair.

Let A_σ (resp. B_σ) be the set of all positions x with $\hat{A}[x] = \sigma$ (resp. $\hat{B}[x] = \sigma$). Consider a bipartite graph G_σ with the set of vertices $V(G_\sigma) = A_\sigma \cup B_\sigma$, and the edge set $E(G_\sigma) = \{(x, y) \in A_\sigma \times B_\sigma : (x, y) \text{ is a significant pair}\}$. Our main goal is to bound the number of edges in G_σ. To do that, we first define some family of „bad" configurations of edges and prove that if any of them is forbidden, the number of edges can be bounded as desired. Finally – which is the most technical part – we show that at least one of those configurations does not, indeed, appear in the graph.

For any $1 \le x \le |\hat{A}|$, let us denote by $A_x^{\rightarrow}$ the suffix of $\hat{A}$ starting at x (including x). For an integer k we say that $A_x^{\rightarrow}$ is a *k-dense suffix*, if:

- $|A_x^{\rightarrow}| \le \lceil k \cdot \delta \rceil$,
- There are $\lfloor n/\delta \rfloor$ distinct edges $(x, c_1), \ldots, (x, c_{\lfloor n/\delta \rfloor}) \in G_\sigma$,
- Every c_i has, in turn, k distinct edges (y_{ij}, c_i) for $j = 1, 2, \ldots, k$ and some $y_{ij} \in A_x^{\rightarrow}$.

The following lemma is the core idea of the proof, as it forbids at least one dense suffix to appear in G_σ. As its proof needs careful analysis (boiling down to counting predecessors of σ-pairs), and is somewhat technical, we defer the proof until Section 3.4. Before that, we will use Lemma 14 to show our ultimate goal, Theorem 11.

▶ **Lemma 14.** *For every A and B with $|A|, |B| \le n$, and for the corresponding graph G_σ, there exists some positive integer $k \le n/\delta$ such that there is no k-dense suffix in $\hat{A}$.*

To prove that this lemma bounds the number of edges in G_σ, we first split vertices of $\hat{A}$ according to their degree. The *small* vertices are these with degree at most $\frac{2n}{\delta}$, the rest being *large* vertices. The following observation is straightforward:

▶ Remark 15. The total number of edges incident to small vertices of $\hat{A}$ is at most $|A_\sigma| \cdot \frac{2n}{\delta}$.

It remains to bound the number of edges incident to large vertices in $\hat{A}$. For every connected substring $S \subseteq \hat{A}$, let us denote by $L(S)$ the number of such edges between S and $\hat{B}$.

▶ **Lemma 16.** *For every A and B with $|A|, |B| \le n$, $L(\hat{A}) \le |B_\sigma| \cdot \frac{2|\hat{A}|}{\delta}$.*

Proof. We use induction on $|\hat{A}|$ (please note that n remains fixed throughout the proof, so $|\hat{A}|$ is always equal to $|A| + 2n$). The minimal length of a padded sequence is $|\hat{A}| = 2n + 1$ – i.e. with only one non-dummy element – and in this case we have $|E| \le |B_\sigma|$. Suppose now that we have some $\hat{A}$ with $|\hat{A}| = a$, and have already proven the statement for all A' with $|A'| < a$.

Let k be the integer obtained from Lemma 14 and let Y be the suffix of of $\hat{A}$ of length $\lceil k\delta \rceil$ (as $\lceil k\delta \rceil \le n$, Y is a proper suffix). We can now use Lemma 14 to bound the number of edges between Y and $\hat{B}$. Let us initially place k tokens on every σ-element of $\hat{B}$ and consider all elements of Y, starting from the last one. For every large vertex $x \in Y$ we look at all its neighbors and remove one token from each of them, whenever they still have one. We claim that every time, at least half of these neighbours (which is at least n/δ, as we are dealing with large vertices) must still have a token to spare. This is because if at least n/δ neighbors of x were already tokenless, than $A_x^{\rightarrow}$ would be a k-dense suffix: clearly $|A_x^{\rightarrow}| \le \lceil k\delta \rceil$, and we have just found that x has $\lfloor n/\delta \rfloor$ neighbors which have already lost their k tokens, so each of them has k neighbours in $A_x^{\rightarrow}$.

Therefore, every x must be able to take a token from at least half of its neighbors. As there are only $k \cdot |B_\sigma|$ tokens to be removed, the total number of edges $L(Y)$ cannot exceed $2k|B_\sigma| \le 2 \cdot \frac{\lceil k\delta \rceil}{\delta} \cdot |B_\sigma| = |Y| \cdot \frac{2|B_\sigma|}{\delta}$.

Let us denote by $\hat{A} - Y$ the prefix of $\hat{A}$ obtained by deleting Y from the end of $\hat{A}$ (i.e. the prefix of length $|\hat{A}| - |Y|$). Now if $|\hat{A} - Y| \le 2n$ (i.e. Y uses up all non-padding symbols), then $L(\hat{A} - Y) = 0$. Otherwise, we can apply the induction hypothesis to $\hat{A} - Y$, obtaining $L(\hat{A} - Y) \le |\hat{A} - Y| \cdot \frac{2|B_\sigma|}{\delta}$. In both cases we have $L(\hat{A}) = L(\hat{A} - Y) + L(Y) \le |\hat{A}| \cdot \frac{2|B_\sigma|}{\delta}$, as desired. ◀

We can now prove Theorem 11 and bound the total number of significant pairs:

Proof of Theorem 11. We want to show that for any A, B with $|A|, |B| \leq n$, the number of significant pairs is at most $\mathcal{O}\left(\frac{n^2}{\log^{1/3} n}\right)$. We already know that is enough to prove it for padded sequences $\hat{A}$ and $\hat{B}$. For a fixed σ, the number of significant σ-pairs is, from Remark 15 and Lemma 16, at most $|A_\sigma| \cdot \frac{2n}{\delta} + |B_\sigma| \cdot \frac{2|\hat{A}|}{\delta} \leq (|A_\sigma| + |B_\sigma|) \cdot \frac{6n}{\delta}$. Summing this over all possible symbols σ (and using the fact that $\sum_\sigma (|A_\sigma| + |B_\sigma|) = |\hat{A}| + |\hat{B}| \leq 6n$), we get that the total number of significant pairs does not exceed $\frac{36n^2}{\delta} \leq \frac{144n^2}{\log^{1/3} n}$. ◀

To close this section, it may be worth asking if our bound of $\mathcal{O}\left(\frac{n^2}{\log^{1/3} n}\right)$ significant pairs is tight, or at least close to the optimal one. We partially answer that in the full version of the paper, providing an example of two sequences with $\Omega\left(\frac{n^2}{\log n}\right)$ significant pairs. Hence, we cannot go lower than this bound, but there is still a gap for possible future work.

3.4 Dense suffixes and the predecessor matrix

In this section we complete the missing part by proving Lemma 14. To do that, we need to introduce a new concept – the *predecessor matrix* M of significant pairs. To give some intuition what this matrix is, imagine that we first find the longest common increasing subsequence of $\hat{A}$ and $\hat{B}$ and put this sequence of significant pairs into the first column of M, one pair in every cell (starting from last element of LCIS, downwards). Then we delete some final elements of $\hat{B}$ such that the length of LCIS decreases by exactly 1, find the new (possibly very different) LCIS, and form M's second column the same way. We can repeat this process n times, and the padding of the sequences always allows us to compute n predecessors.

Each entry of M is a significant pair (x, y). We refer to $\mathit{symbol}(x, y)$ as *color* of this entry of M, as we feel this gives a better intuition. To analyze M, we look at the number of different colors in each row. We show that too few colors in every row would cause $\hat{B}$ to accumulate more than $3n$ elements – which is impossible – so there is a row (say, k-th) with somewhat more colors – we then prove that this row corresponds to the desired k fulfilling the statement of Lemma 14.

To formally define M, recall the previous assumptions: we have sequences $\hat{A}$ and $\hat{B}$ which are both padded with P_n and do not contain symbols greater than σ. Let $a = |\hat{A}|$, $b = |\hat{B}|$ and $\ell = \mathit{lcis}(a, b)$. Because of padding we know that $2n < a, b \leq 3n$ and that $\ell > 2n$. It is easy to see that for every $y > 1$ we have $\mathit{lcis}(a, y-1) \geq \mathit{lcis}(a, y) - 1$. Therefore, if we iterate y downwards from b to 1, $\mathit{lcis}(a, y)$ takes all values between ℓ and 1. In particular, there must exist elements $b = b_1 > b_2 > \ldots > b_n$ such that:

$$\mathit{lcis}(a, b_j) = \ell - j + 1,$$

for every $j = 1, 2, \ldots, n$.

We define the *predecessor matrix* M as an $n \times n$ matrix of matching pairs. For $1 \leq j \leq n$ we consider the longest common increasing sequence realizing $\mathit{lcis}(a, b_j)$ and define $M[1, j]$ as its last element (x^*, y^*). If there are multiple possibilities, we pick the one with minimal y^* and then with minimal x^*. We then define $M[i, j] = \pi^{i-1}(M[1, j])$. In other words, below every pair in M we put its predecessor. Observe that the properties of predecessors immediately imply $\mathit{lcis}^{\rightarrow}(M[i, j]) = \mathit{lcis}^{\rightarrow}(M[1, j]) - i + 1 = \ell - i - j + 2$.

If we pick, instead of some b_j, another b'_j such that $\mathit{lcis}(a, b'_j) = \mathit{lcis}(a, b_j) = \ell - j + 1$, we will get exactly the same $M[1, j]$, and thus the same $M[i, j]$ for all $i = 1, 2, \ldots, n$ – this is because of the tie-breaker rule for the choice of $M[1, j]$. Thus, the matrix M does not depend on the choice of $b_1, \ldots, b_n$, but only on $\hat{A}$ and $\hat{B}$.

We begin with a technical lemma about M which will be useful later. This observation is a generalization of the idea introduced in Section 3.1 – if s different significant pairs are incident to a single vertex $x \in \hat{A}$, then among their i-th predecessors we expect at least about s/i distinct values.

▶ **Lemma 17.** *For some i, i', j, j' with $1 \leq j < j' \leq n$ and $1 \leq i, i' \leq n$, let $(x, y) = M[i, j]$ and $(x', y') = M[i', j']$. If $j' \geq j + i$, then $y' < y$.*

Proof. Suppose to the contrary that $y \leq y'$. As $y' \leq b_{j'}$ and a is the last element of $\hat{A}$, it would imply $(x, y) \leq (a, b_{j'})$, which would in turn yield $\mathit{lcis}(a, b_{j'}) \geq \mathit{lcis}^{\rightarrow}(x, y) = \mathit{lcis}^{\rightarrow}(M[i, j]) = \mathit{lcis}^{\rightarrow}(M[1, j]) - i + 1 = \mathit{lcis}(a, b_j) - i + 1$. But as $j' \geq j + i$, there must be $\mathit{lcis}(a, b_{j'}) \leq \mathit{lcis}(a, b_j) - i$. This contradiction shows $y' < y$. ◀

As stated before, we will refer to the symbols of $\hat{A}$ and $\hat{B}$ as *colors*, imagining that every entry (x, y) in M is painted with a color corresponding to $\mathit{symbol}(x, y)$, the (common) symbol of $\hat{A}[x]$ and $\hat{B}[y]$. In every column of M the colors are strictly decreasing, and thus different. Hence, no color can have more than n entries in M. It is also evident that two entries in M must correspond to different elements in $\hat{A}$ and $\hat{B}$ if they have different colors. The main lemma of this section states, roughly, that the rows of M do not contain too few colors:

▶ **Lemma 18.** *There is some $2 \leq k \leq n/\delta$ such that every submatrix of M consisting of some $\lceil \frac{n}{\delta} \rceil$ columns (not necessarily consecutive) and rows $1, \ldots, k-1$ uses at least $\lceil k\delta \rceil$ colors.*

Proof. Consider all k that are powers of 2: $k = 2^q$ for $1 \leq q \leq \lfloor \log n - \log \delta \rfloor$. Suppose, to the contrary, that for every such $k = 2^q$ we can find some $\lceil \frac{n}{\delta} \rceil$ columns $c_1, \ldots, c_{\lceil \frac{n}{\delta} \rceil}$ of M which have at most $\delta \cdot 2^q$ colors in total in rows $1, 2, \ldots, 2^q - 1$. From these columns c_i and the lower half of these rows $(2^{q-1}, \ldots, 2^q - 1)$ we form a submatrix M_q of M. These matrices are defined for $1 \leq q \leq \lfloor \log n - \log \delta \rfloor$ and have the following properties:

- they are disjoint submatrices of M (as every one takes different rows),
- for any q, the matrix M_q contains $2^{q-1} \cdot \lceil \frac{n}{\delta} \rceil$ pairs,
- for any q, the entries of M_q use at most $\delta \cdot 2^q$ colors between them.

Let a color be *q-strong*, if at least $\frac{n}{4\delta^2}$ entries in M_q are of that color. Observe that a particular color can be q-strong for at most $4\delta^2$ distinct values of q, otherwise – as all M_q's are disjoint – there would be more than n entries of that color in M, which is impossible.

For any q, the colors which are not q-strong can make up for at most half of entries in M_q (as there are $\delta \cdot 2^q$ colors in M_q, none of which can have more than $\frac{n}{4\delta^2}$ entries – a total of $\frac{n}{2\delta} \cdot 2^{q-1}$). Hence, there are at least $\frac{n2^{q-1}}{2\delta}$ pairs in M_q which have q-strong colors. Let us *mark* all these entries of M_q.

For any $t = 0, 1, \ldots, 2^q - 1$ let $\mathcal{M}_q(t)$ be the set of columns $M_q[\cdot, j]$ with $j \equiv t \mod 2^q$. For a fixed q, at least one of the sets $\mathcal{M}_q(t)$ must contain at least $\frac{n}{4\delta}$ marked entries, as there are $\frac{n2^{q-1}}{2\delta}$ marked entries in M_q split between 2^q sets. But we can show that if (x, y) and (x', y') are two different pairs in some $\mathcal{M}_q(t)$, then $y \neq y'$. Indeed, both pairs are either in the same column – which makes them have different colors and thus no common elements – or at least 2^q columns apart, in which case $y \neq y'$ because of Lemma 17. This in turn means that for every q there is a set B_q of at least $\frac{n}{4\delta}$ distinct elements of $\hat{B}$, each of a q-strong color. A color can be q-strong for at most $4\delta^2$ values of q, so in the sum $B_1 \cup \ldots \cup B_{\lfloor \log n - \log \delta \rfloor}$ every element can be repeated at most $4\delta^2$ times. This accounts for at least $\lfloor \log n - \log \delta \rfloor \cdot \frac{n}{4\delta} \cdot \frac{1}{4\delta^2} = n \frac{\lfloor \log n - \log \delta \rfloor}{16\delta^3}$ distinct elements of $\hat{B}$. For $\delta = \frac{\log^{1/3} n}{4}$ this is equal to $4n \cdot \frac{\lfloor \log n - 1/3 \cdot \log \log n + 2 \rfloor}{\log n} > 3n \geq |\hat{B}|$. The contradiction proves that at least one $k = 2^q$ for some q must satisfy the statement. ◀

Now let us return to the graph G_σ of significant pairs. Recall that σ is the largest symbol appearing in input sequences – for the rest of the section, we retain this assumption. As Figure 2 shows, any two incident edges must correspond to pairs with different values of LCIS, as otherwise the pairs could not be significant. This is formalized in a following simple observation:

▶ **Lemma 19.**

a) *If $(x, y_1), (x, y_2), \ldots, (x, y_s)$ are significant σ-pairs for $y_1 > y_2 > \ldots > y_s$, then for every $1 \le i \le j \le s$, we have $lcis(x, y_i) \ge lcis(x, y_j) + (j - i)$.*

b) *If $(x_1, y), (x_2, y), \ldots, (x_s, y)$ are significant σ-pairs for $x_1 > x_2 > \ldots > x_s$, then for every $1 \le i \le j \le s$, we have $lcis(x_i, y) \ge lcis(x_j, y) + (j - i)$.*

Proof. The first claim easily follows from the fact that $lcis(x, y_i) \ge lcis(x, y_{i+1}) + 1$, which is part of the definition of the significant pair. The second proof is symmetric. ◀

Finally, we can restate Lemma 14 and prove it using predecessor matrices:

▶ **Lemma 14.** *For every A and B with $|A|, |B| \le n$, and for the corresponding graph G_σ, there exists some positive integer $k \le n/\delta$ such that there is no k-dense suffix in $\hat{A}$.*

Proof. Assume, to the contrary, that for every k there is a k-dense suffix. Pick an arbitrary $k \le n/\delta$ and let $x \in A$ be such that $A_x^{\rightarrow}$ is k-dense. Let $|\hat{A}| = a$, let $r = \lceil n/\delta \rceil$ and let $(x, c_1), \ldots, (x, c_r)$ be the significant pairs from the definition of k-dense suffix (meaning that each c_i has k neighbours in $A_x^{\rightarrow}$). We can assume that $c_1 > c_2 > \ldots > c_r$. We will now show another sequence $c'_1, c'_2, \ldots, c'_r$ such that:

(1) $lcis(a, c'_1) > lcis(a, c'_2) > \ldots > lcis(a, c'_r)$,

(2) $lcis(a, c'_i) \ge lcis(x, c'_i) + k - 1$ for $i = 1, 2, \ldots, r$.

To do that, we set $c'_1 = c_1$, and for $i > 1$, we pick $c'_{i+1} = c_{i+1}$ if $lcis(a, c_{i+1}) < lcis(a, c'_i)$. If not, we define c'_{i+1} to be the largest element with $lcis(a, c'_{i+1}) = lcis(a, c'_i) - 1$. Observe that in the second case we know that $lcis(a, c_{i+1}) \ge lcis(a, c'_i) > lcis(a, c'_{i+1})$, so always $c'_{i+1} \le c_{i+1}$.

Inequality (1) follows immediately from the definition of c'_i. To see (2), first observe that if $c'_i = c_i$, then there are k significant σ-pairs between (x, c_i) and (a, c_i) – neighbors of c_i – which we will denote by $(y_1, c_i), \ldots, (y_k, c_i)$ and assume that $y_1 > \ldots > y_k$. As we assume σ to be the largest symbol, we can write $lcis(y_j, c_i) = lcis^{\rightarrow}(y_j, c_i)$. From this and Lemma 19 we derive:

$$lcis(a, c_i) \ge lcis(y_1, c_i) \ge lcis(y_k, c_i) + k - 1 \ge lcis(x, c_i) + k - 1.$$

Now consider the case $c'_i < c_i$. Let β be the smallest integer such that $\beta < i$ and $c_{i-\beta} = c'_{i-\beta}$ (it always exists, as we can take $\beta = i - 1$). From the definition of c'_i we have $lcis(a, c'_i) = lcis(a, c'_{i-1}) - 1 = \ldots = lcis(a, c'_{i-\beta}) - \beta = lcis(a, c_{i-\beta}) - \beta \ge lcis(x, c_{i-\beta}) + k - 1 - \beta$. Now, because all (x, c_j) are significant, we have $lcis(x, c_{i-\beta}) \ge lcis(x, c_i) + \beta$ from Lemma 19, so $lcis(a, c'_i) \ge lcis(x, c_i) + k - 1 \ge lcis(x, c'_i) + k - 1$.

The pairs (a, c'_i) are some choice of r different columns of the predecessor matrix M. Consider any c'_i, and its corresponding column j. Pick a positive integer $s \le k - 1$. Let $(a^*_i, c^*_i) = M[s, j]$. Recall that from the definition of M we have $lcis^{\rightarrow}(a^*_i, c^*_i) = lcis(a, c'_i) - s + 1$. We also know that $c^*_i \le c'_i \le c_i$. If $a^*_i < x$, then $(a^*_i, c^*_i) \le (x, c'_i)$, which implies $lcis(a, c'_i) - s + 1 = lcis(a^*_i, c^*_i) \le lcis(x, c'_i) \le lcis(a, c'_i) - k + 1$, which is impossible for $s \le k - 1$. Then $a^*_i \ge x$. So the only colors available for $M[s, j]$ for the chosen r columns and $s \leq k-1$ are those appearing in $A_x^{\rightarrow}$, and there are at most $\lceil k\delta \rceil$ of them. Hence, for any k we can produce, from a k-dense suffix, an $\lceil \frac{n}{\delta} \rceil$-column submatrix of M having at most $\lceil k\delta \rceil$ colors in total in its first $k-1$ rows. This contradicts Lemma 18 and proves that for some k there are no k-dense suffixes. ◀

4 The algorithm

To implement our algorithm, we need a specific data structure – an associative array which can store a number of elements ordered by their *keys*. We assume the keys to be distinct integers between 1 and n. This data structure A must provide the following operations:

- Insert(A, s) – adds the element s to A,
- Delete(A, x) – removes the element having key x from the A (we assume that this is called only for $x \in A$),
- Find(A, x) – returns the element whose key is x if there is one, or NULL otherwise,
- Next(A, x), Prev(A, x) – returns the first element whose key is larger (respectively, smaller) than x.

To achieve the desired running time, we need all these operations to work in $\mathcal{O}(\log\log n)$ complexity (possibly amortized), and *van Emde Boas queue* [26] does exactly that. While the standard implementation requires $\mathcal{O}(n)$ space (and thus $\mathcal{O}(n)$ initialization time, which would be too much for us, as we employ $\mathcal{O}(n)$ queues), there is also a randomized version ([22]) that needs only $\mathcal{O}(m)$ time and space, where m is the maximal number of elements on the queue. In the full version of the paper we show how to construct a deterministic van Emde Boas queue with $\mathcal{O}\left(m + \frac{n}{\log^c n}\right)$ time and space bounds, with c being any desired constant, while retaining the $\mathcal{O}(\log\log n)$ query complexity. These bounds also suit our needs.

The algorithm takes, as the input, two integer sequences A and B with $|A| = |B| = n$. Its main idea is to consider all symbols from A and B in increasing order (there are at most $2n$ of them, so we can sort them in $\mathcal{O}(n \log n)$). For every symbol σ, the algorithm finds and stores all significant σ-pairs. For that, we employ n van Emde Boas queues $Q_1, Q_2, \ldots, Q_n$, with every Q_k storing the significant pairs (x, y) with $lcis^{\rightarrow}(x, y) = k$, sorted by x. For convenience, we define $Q = Q_1 \cup \ldots \cup Q_n$. We also keep Q_0 as one-element queue $(0, 0)$.

Whenever some Q_k contains two pairs $(x, y) \leq (x', y')$, we drop (x', y') from Q_k. Informally, we can do it because (x, y) can replace (x', y') in every situation. We say that (x, y) *dominates* (x', y') and remove any dominated pairs from any Q_k. Observe that a pair is significant if and only if it is not dominated by any pair of the same symbol (a significant pair, however, may still be dominated by other pairs with larger symbols).

This leads to the following invariant:

▷ Claim 20 (Algorithm invariant). For every k, all pairs $(x, y) \in Q_k$ are in strict increasing order with respect to x and in strict decreasing order with respect to y.

To keep the invariant, we modify Insert() into the following Insert-Inv() procedure. It only inserts a pair (x, y) if it is not dominated by another pair, and after inserting it removes all larger pairs.

The amortized complexity of Insert-Inv() is $\mathcal{O}(\log\log n)$: the loop in lines 14-19 deletes an element with every iteration (so it cannot do more iterations than the total number of elements in queue), and outside the loop there is only a constant number of standard queue operations.

Algorithm 1 New version of INSERT() keeping the invariant.

```
procedure INSERT-INV($Q_k, (x, y)$)
    $(x', y') \leftarrow$ FIND($Q_k, x$)                ▷ check for other pairs with key $x$
    if $(x', y') \neq$ null and $y' \leq y$ then
        return
    end if
    if $(x', y') \neq$ null and $y' > y$ then
        DELETE($x'$)
    end if
    $(a, b) \leftarrow$ PREV($Q_k, x$)
    if $(a, b) \neq$ null and $b \leq x$ then       ▷ $(a, b) \leq (x, y)$, so we should not insert $(x, y)$
        return
    end if
    INSERT($Q_k, (x, y)$)
    repeat                   ▷ now we remove all $(a, b) \geq (x, y)$, restoring the invariant
        $(a, b) \leftarrow$ NEXT($Q_k, x$)
        if $(a, b) \neq$ null and $\geq (x, y)$ then
            DELETE($Q_k, a$)
        end if
    until $(a, b) =$ null or not $(a, b) \geq (x, y)$
end procedure
```

Apart from queues $Q_1, Q_2, \ldots, Q_n$ we will also need, for every symbol σ, two van Emde Boas queues X_σ and Y_σ which store positions of all σ-symbols in A and B, respectively: $X_\sigma = \{i : A[i] = \sigma\}$, $Y_\sigma = \{j : B[j] = \sigma\}$. These structures do not change during the algorithm, and their sole purpose is finding σ-symbols closest to a given position.

Now we are ready to introduce the main idea of the algorithm. Recall that we assume that the number of significant pairs between A and B is at most $\mathcal{O}\left(\frac{n^2}{t}\right)$ with $t = t(n) = \Theta\left(\log^p n\right)$ for some p. We iterate over all the symbols, dividing them into two categories:

- *frequent* – appearing more than $\frac{n}{\sqrt{t}}$ times in B,
- *infrequent* – with at most $\frac{n}{\sqrt{t}}$ occurrences in B.[‡]

Let us start with an informal sketch of the algorithm behavior for both cases. The frequent symbols are easier: for every such symbol σ we iterate through all previously found pairs, and for every $(x, y) \in Q$ we find the next occurrence of σ after $A[x]$ (say, $A[x^*]$) and the next occurrence y^* of σ in B after y. In other words, we find a σ-pair (x^*, y^*) for which (x, y) is a predecessor. As we will ensure that Q contains only significant pairs (and thus cannot get too big) and there are no more than $\sqrt{t}$ frequent symbols, the total complexity will fit into desired limits.

To handle infrequent symbols, observe that every such symbol in A can form a matching pair with at most $\frac{n}{\sqrt{t}}$ elements of B. Hence, there are at most $\frac{n^2}{\sqrt{t}}$ matching pairs on infrequent symbols, so we can iterate through all of them. The hardest part is to determine, for every infrequent pair (x, y), the value of $lcis^{\rightarrow}(x, y)$. For that, we will need a separate subroutine and a non-trivial analysis.

[‡] The technique of splitting symbols of a string according to their number of occurences is not new – it has been used, e.g. in [4] and [5], though it is more common to have split thresholds closer to $\sqrt{n}$.

The whole algorithm is presented below:

Algorithm 2 LCIS by significant pairs.

```
procedure LCIS($A, B$)
    $Q_0 \leftarrow \{(0,0)\}$
    for all $\sigma$ – symbols in increasing order do
        $T \leftarrow \varnothing$                                          ▷ for storing new pairs
        if $\sigma$ occurs less than $n/\sqrt{t}$ times in $B$ then          ▷ if $\sigma$ is infrequent...
            for all  $x : A[x] = \sigma$, in increasing order  do                   ▷ fix $x$...
                $k \leftarrow 0$
                for all $y : B[y] = \sigma$, in inc. order  do  ▷ ...compute $lcis^{\rightarrow}(x,y)$ for all $y$
                    $k' \leftarrow$ ComputeNextPair($x, y, k$)          ▷ using a special subroutine
                    $T \leftarrow T \cup (x, y, k')$
                    $k \leftarrow k'$                   ▷ $k = lcis^{\rightarrow}(x,y)$, for last considered $y$
                end for
            end for
        else                                                      ▷ If $\sigma$ is frequent...
            for $k = 1, 2, \ldots, n$ do
                for all $(x,y) \in Q_k$ do     ▷ every pair $(x,y) \in Q$ may be a predecessor...
                    $x' \leftarrow$ Next($X_\sigma, x$)
                    $y' \leftarrow$ Next($Y_\sigma, y$)                      ▷ ...of some $\sigma$-pair $(x',y')$
                    $T \leftarrow T \cup (x', y', k+1)$
                end for
            end for
        end if
        for all $(x,y,k) \in T$ do                       ▷ all new pairs are now added to $Q$
            Insert-Inv($Q_k, (x,y)$)
        end for
    end for
    return largest $k$ with $Q_k \neq \varnothing$
end procedure
```

Before analyzing the algorithm, we must explain the ComputeNextPair() subroutine. It takes three arguments: positions $x \in A$, $y \in B$, such that $A[x] = B[y] = \sigma$, and an integer k. It assumes that $lcis^{\rightarrow}(x,y) \geq k$ and its goal is to find the exact value of $lcis^{\rightarrow}(x,y)$. It also assumes that for every $j < lcis^{\rightarrow}(x,y)$, there is a pair $(x_j, y_j) \in Q_j$ with $(x_j, y_j) \prec (x,y)$ – informally, this means that all the predecessors of (x,y) have already been considered, and Lemma 21 will prove that this condition is indeed satisfied whenever ComputeNextPair() is invoked.

Therefore the subroutine must determine the largest $\ell \geq k-1$ for which there is a pair $(x',y') \prec (x,y)$ with $lcis^{\rightarrow}(x',y') = \ell$. We can guess ℓ, and verify whether there exists a right pair $(x',y') \prec (x,y)$ in Q_ℓ: if there is one, then Prev(Q_ℓ, x) $\leq (x,y)$. This allows us to do a binary search for ℓ:

Algorithm 3 Finding $lcis^{\rightarrow}(x, y)$, with assumption that it is at least k.

```
procedure ComputeNextPair($x, y, k$)
    $d \leftarrow 1$                                   ▷ first, find $d$ – a rough approximation for $\ell - k$
    while Prev($Q_{k+d}, x$) $\prec (x, y)$ do
        $d \leftarrow 2d$                                  ▷ if $d$ is too small, try $2d$
    end while
    $p \leftarrow k$                                   ▷ now we know that $k + d/2 \le \ell < k + d$
    $q \leftarrow k + d$
    while $p < q$ do                                   ▷ so we can do the real binary search
        $s \leftarrow \lceil \frac{p+q}{2} \rceil$
        if Prev($Q_s, x$) $\prec (x, y)$ then
            $p \leftarrow s$
        else
            $q \leftarrow s - 1$
        end if
    end while
    return $p$
end procedure
```

The first part of the algorithm finds d for which $d/2 \le |\ell - k| < d$ (if $\ell = k$, then we assume $d = 1$). The second one is a binary search on the interval $[k, k + d)$. Therefore, the algorithm makes at most $2\log(d + 1) = 2 \cdot \log(lcis^{\rightarrow}(x, y) - k + 2)$ steps, each step invoking a queue operation once.

Let us now go back to the main algorithm, proving its correctness:

▶ **Lemma 21.** *After processing a symbol σ, the following two facts hold:*

(1) *If $(x, y) \in Q_k$, then (x, y) is a significant σ'-pair for some $\sigma' \le \sigma$ with $lcis^{\rightarrow}(x, y) = k$;*

(2) *For any $k \ge 1$, $\sigma' \le \sigma$ and for every σ'-pair (x, y) with $lcis^{\rightarrow}(x, y) \ge k$, there is some $(x^*, y^*) \in Q_k$ with $(x^*, y^*) \le (x, y)$.*

Proof. Let us use induction on σ. Observe that assuming induction hypothesis, we only need to prove two weaker facts:

(1') If $(x, y) \in Q_k$, then $lcis^{\rightarrow}(x, y) = k$;

(2') For any $k \ge 1$ and for every σ-pair (x, y) with $lcis^{\rightarrow}(x, y) = k$, there is some $(x^*, y^*) \in Q_k$ with $(x^*, y^*) \le (x, y)$.

Indeed, for any pair (x, y) with $lcis^{\rightarrow}(x, y) = k' > k$, (2) is true because of the induction hypothesis applied to the pair $(x', y') = \pi^{k'-k}(x, y)$. We know that (x', y') is a τ-pair for some $\tau < \sigma$, so induction hypothesis provides a pair $(x^*, y^*) \in Q_k$ with $(x^*, y^*) \le (x', y') \le (x, y)$. Next, note that (2) is also true for $\sigma' < \sigma$, again from induction hypothesis and the fact that once a pair is in Q, it can only be dislocated by another pair that dominates it.

Also, if we show (2) and (1'), this will automatically imply that every $(x, y) \in Q_k$ must be significant, and thus that (1) is true – let (x', y') be a pair with $(x', y') \le (x, y)$ and $lcis^{\rightarrow}(x', y') = lcis^{\rightarrow}(x, y) = k$. There is, by (2), another pair $(x'', y'') \le (x', y') \le (x, y)$, which is also in Q_k. But with Claim 20, (x, y) and (x'', y'') cannot both be in Q_k unless $(x, y) = (x', y') = (x'', y'')$, so (x, y) is significant.

To prove the remaining statements (1') and (2'), consider two cases:

Case 1: infrequent σ. Pick an arbitrary σ-pair (x, y) and let $lcis^{\rightarrow}(x, y) = k$. We will show that at some point during processing symbol σ the instruction Insert-Inv($Q_k, (x, y)$) is invoked.

For any integer i with $1 \leq i \leq k$ let $(x_i, y_i) = \pi^i(x, y)$. We know that $symbol(x_i, y_i) < \sigma$ and $lcis^{\rightarrow}(x_i, y_i) = k - i$, so by induction hypothesis (2) there was some $(x_i^*, y_i^*) \in Q_{k-i}$ with $(x_i^*, y_i^*) \leq (x_i, y_i)$, which implies $(x_i^*, y_i^*) \prec (x, y)$ (observe that for $i = k$, we need the artificial pair $(0, 0) \in Q_0$). But then when (x, y) is considered by ComputeNextPair(), for every $i \geq 1$ and for each of the predecessors $(x_i, y_i) = \pi^i(x, y)$ there is a pair $(x_i^*, y_i^*) \leq (x_i, y_i)$ in Q_{k-i}. This satisfies the conditions needed by ComputeNextPair(), so the binary search properly computes $lcis^{\rightarrow}(x, y)$ as k (note that it is not possible to find any larger candidate for predecessor – any $(u, v) \prec (x, y)$ found in $Q_{k'}$ for $k' \geq k$ would mean either that $lcis^{\rightarrow}(u, v)$ had been computed wrong, or that $lcis^{\rightarrow}(x, y) > k$). This shows that the algorithm tries to insert (x, y) into the proper queue Q_k, which immediately proves (1'). Also, (x, y) may remain in Q_k, be dislocated later, or even fail to be inserted because of some other pair dominating it. Either way, some pair $(x^*, y^*) \leq (x, y)$ will be present in Q_k to the very end of the algorithm. This completes the proof of (2').

Case 2: frequent σ. Let us first prove (2') for any k and for any σ-pair (x, y) with $lcis^{\rightarrow}(x, y) = k$. We can assume that (x, y) is significant (if not, we replace it with a significant σ-pair $(x', y') \leq (x, y)$). As in Case 1, from the induction hypothesis we know that some pair $(x', y') \prec (x, y)$ is present in Q_{k-1}. The algorithm must at some point consider (x', y'). If the next σ symbol in A after x' is not x but some z, then $lcis^{\rightarrow}(z, y) \geq k$, so (x, y) could not be significant. By the same argument, the next σ symbol in B after y' must be y. Therefore (x, y) is a candidate to be inserted into Q_k, so either it remains there itself, or is dominated by other pair $(x^*, y^*) \in Q_k$. Either way, (2') is shown.

For (1) we need to rule out a possibility that a σ-pair (x, y) with $lcis(x, y) = k$ will be inserted, besides Q_k, into some other $Q_{k'}$ with $k' \neq k$, as a frequent pair could be theoretically considered multiple times by the algorithm. But for $k' > k$ this would have been caused by another pair $(x^*, y^*) \in Q_{k'-1}$ with $(x^*, y^*) \prec (x, y)$. This contradicts $lcis^{\rightarrow}(x, y) = k$, as $k' - 1 \geq k$. For $k' < k$ observe that by induction hypothesis (2) applied to $\pi^{k-k'}(x, y)$ we already have a pair in $Q_{k'}$ which dominates (x, y), so the insertion must fail. ◀

▶ **Corollary 22.** *The algorithm correctly returns the length of longest common increasing subsequence of A and B.*

Proof. Let k be the value of LCIS and let (x, y) be a significant pair with $lcis^{\rightarrow}(x, y) = k$. From statement (2) of Lemma 21 we know that some pair $(x', y') \leq (x, y)$ must be in Q_k at the end of the algorithm. Therefore the algorithm returns at least k. On the other hand, for every $k' > k$, $Q_{k'} = \varnothing$, as any pair in it would contradict statement (1) from Lemma 21. ◀

▶ **Lemma 23.** *Let $x \leq |A|$ and (x, y_1), (x, y_2), ..., (x, y_m) be all σ-pairs formed by x, for an infrequent σ. Then, all calls to* ComputeNextPair$(x, \cdot, \cdot)$ *work in $\mathcal{O}\left(\frac{n(\log\log n)^2}{\sqrt{t}}\right)$ total time complexity.*

Proof. We know that $m \leq \frac{n}{\sqrt{t}}$, as σ is infrequent. Recall also that $\log t = \Theta(\log\log n)$. Let $\ell_i = lcis^{\rightarrow}(x, y_i) - lcis^{\rightarrow}(x, y_{i-1}) + 2$ for $i = 1, 2, \ldots, m$, assuming $y_0 = 0$. The i-th call to ComputeNextPair() requires $\mathcal{O}(\log \ell_i)$ steps of binary search, with every step having

$\mathcal{O}(\log\log n)$ complexity from queue operations. Therefore, the whole procedure works in $\mathcal{O}\left(\log\log n \cdot \sum_{i=1}^{m} \log \ell_i\right)$. Consider two cases:

- If $m \leq \frac{n}{\sqrt{t}\log n}$, then $\sum_{i=1}^{m} \log \ell_i \leq m \log n = \mathcal{O}\left(n/\sqrt{t}\right)$.
- If $m > \frac{n}{\sqrt{t}\log n}$, then the Jensen equality yields:
  $\sum_{i=1}^{m} \log \ell_i = m\frac{\sum_{i=1}^{m} \log \ell_i}{m} \leq m \log\left(\frac{\sum_{i=1}^{m} \ell_i}{m}\right)$,
  and as $\sum \ell_i = 2m + lcis^{\rightarrow}(x, y_m) - lcis^{\rightarrow}(x, y_0) \leq 3n$ we have:
  $\sum_{i=1}^{m} \log \ell_i \leq m \log \frac{3n}{m} < m \log(3\sqrt{t} \cdot \log n) = \mathcal{O}(m \log\log n)$.

With $m \leq n/\sqrt{t}$, the total complexity in both cases is $\mathcal{O}\left(\frac{n(\log\log n)^2}{\sqrt{t}}\right)$. ◀

Proof of Theorem 12. We already know that the algorithm correctly computes $lcis(A, B)$. It remains to determine its complexity.

There are $\mathcal{O}(n)$ van Emde Boas queues. Each of them is initialized in constant time if we use randomized version [22], or in $\mathcal{O}(n/\log^c n)$ with some c large enough if we choose the version described in the appendix of the full version of the paper . Either way, total complexity is $\mathcal{O}\left(n^2/\log^c n\right)$, which is fast enough.

We first analyze the cost for infrequent symbols – it is dominated by the calls of ComputeNextPair(). By Lemma 23 the cost is $\mathcal{O}\left(\frac{n(\log\log n)^2}{\sqrt{t}}\right)$ for a fixed x, which yields $\mathcal{O}\left(\frac{n^2(\log\log n)^2}{\sqrt{t}}\right)$ total complexity.

Now let us move on to frequent symbols. For every such symbol, we iterate over all Q. But from Lemma 21 we know Q only contains significant pairs, therefore $|Q| = \mathcal{O}\left(\frac{n^2}{t}\right)$. As every iteration needs only a constant number of queue operations, the total cost for a single symbol is $\mathcal{O}\left(\frac{n^2 \log\log n}{t}\right)$.

Finally, observe that there are at most $\sqrt{t}$ frequent symbols (otherwise there would be $|B| > n$), so the final complexity in this case is $\mathcal{O}\left(\frac{n^2 \log\log n}{\sqrt{t}}\right)$.

It is also worth noting that we can replace the threshold between frequent and infrequent symbols $\frac{n}{\sqrt{t}}$ with $\frac{n}{\sqrt{t \log\log n}}$, and all the proofs would essentially work in the same way, with only minor changes needed. This way we could show the complexity of the algorithm to be in fact $\mathcal{O}\left(\frac{n^2(\log\log n)^{3/2}}{\sqrt{t}}\right)$. The current analysis seems, however, a bit easier to read. ◀

5 Final remarks and open problems related to LCIS

We have shown an algorithm for LCIS that breaks the $\mathcal{O}\left(n^2\right)$ barrier, but there still appears to be plenty of room for improvement and further work on this matter. First, the bound in Theorem 11 for the number of significant pairs may not be tight. In the full version of the paper we give an example of two sequences A and B having $\Omega\left(\frac{n^2}{\log n}\right)$ significant pairs, but this still leaves a gap between $\Omega\left(\frac{n^2}{\log n}\right)$ and $\mathcal{O}\left(\frac{n^2}{(\log n)^{1/3}}\right)$. Also, the algorithm itself might be improved to work in $\mathcal{O}\left(s \cdot (\log\log n)^k\right)$, where s is the number of significant pairs and k is a constant. Taking all this into account, we conjecture that there is an $\mathcal{O}\left(\frac{n^2(\log\log n)^k}{\log n}\right)$ algorithm which uses the significant pairs technique.

The second question related to LCIS stems from the papers [3] and [2]: we know that there is a constant $c \leq 7$ such that an $\mathcal{O}\left(\frac{n^2}{\log^c n}\right)$ algorithm for LCS would lead to unexpected breakthroughs in circuit complexity. Can a similar statement be made for LCIS?

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Quadratic-time hardness of LCS and other sequence similarity measures. In *Proc. 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*, pages 59–78, 2015.

2 Amir Abboud and Karl Bringmann. Tighter connections between formula-sat and shaving logs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 8:1–8:18, 2018. doi:10.4230/LIPIcs.ICALP.2018.8.

3 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: Or: a polylog shaved is a lower bound made. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 375–388, New York, NY, USA, 2016. ACM. doi:10.1145/2897518.2897653.

4 Karl Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, December 1987. doi:10.1137/0216067.

5 Moshe Lewenstein Amihood Amir and Ely Porat. Faster algorithms for string matching with k mismatches. In *J. of Algorithms*, pages 794–803, 2000.

6 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proc. 47th Annual ACM Symposium on Theory of Computing (STOC'15)*, pages 51–58, 2015.

7 Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3sum. In *Proceedings of the 9th International Conference on Algorithms and Data Structures*, WADS'05, pages 409–421, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11534273_36.

8 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, pages 1216–1235, 2018.

9 Timothy M. Chan. The art of shaving logs. In Frank Dehne, Roberto Solis-Oba, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, pages 231–231, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

10 Timothy M. Chan. More logarithmic-factor speedups for 3sum, (median,+)-convolution, and some geometric 3sum-hard problems. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 881–897, Philadelphia, PA, USA, 2018. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=3174304.3175327.

11 Wun-Tat Chan, Yong Zhang, Stanley P. Y. Fung, Deshi Ye, and Hong Zhu. Efficient algorithms for finding a longest common increasing subsequence. *Journal of Combinatorial Optimization*, 13(3):277–288, 2007.

12 Lech Duraj. A linear algorithm for 3-letter longest common weakly increasing subsequence. *Information Processing Letters*, 113(3):94–99, 2013.

13 Lech Duraj, Marvin Künnemann, and Adam Polak. Tight conditional lower bounds for longest common increasing subsequence. *Algorithmica*, 81(10):3968–3992, October 2019. doi:10.1007/s00453-018-0485-7.

14 Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. *Discrete Applied Mathematics*, 212:96–103, 2016. doi:10.1016/j.dam.2015.10.040.

15 Yijie Han and Tadao Takaoka. An $o(n^3 \log\log n/\log^2 n)$ time algorithm for all pairs shortest paths. In Fedor V. Fomin and Petteri Kaski, editors, *Algorithm Theory – SWAT 2012*, pages 131–141, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

16 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

17 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

18 Guy Jacobson and Kiem-Phong Vo. Heaviest increasing/common subsequence problems. In *Combinatorial Pattern Matching, Third Annual Symposium, CPM 92, Tucson, Arizona, USA, April 29 - May 1, 1992, Proceedings*, pages 52–66, 1992.

19 Martin Kutz, Gerth Stølting Brodal, Kanela Kaligosi, and Irit Katriel. Faster algorithms for computing longest common increasing subsequences. *J. Discrete Algorithms*, 9(4):314–325, 2011. doi:10.1016/j.jda.2011.03.013.

20 Kasper Green Larsen and Ryan Williams. Faster online matrix-vector multiplication. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 2182–2189, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=3039686.3039828.

21 William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

22 Kurt Mehlhorn and Stefan Näher. Bounded ordered dictionaries in o(log log N) time and o(n) space. *Inf. Process. Lett.*, 35(4):183–189, 1990. doi:10.1016/0020-0190(90)90022-P.

23 Adam Polak. Why is it hard to beat $O(n^2)$ for longest common weakly increasing subsequence? *Information Processing Letters*, 132:1–5, 2018.

24 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proc. 45th Annual ACM Symposium on Symposium on Theory of Computing (STOC'13)*, pages 515–524, 2013.

25 Yoshifumi Sakai. A linear space algorithm for computing a longest common increasing subsequence. *Inf. Process. Lett.*, 99(5):203–207, 2006. doi:10.1016/j.ipl.2006.05.005.

26 P. van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, SFCS '75, pages 75–84, Washington, DC, USA, 1975. IEEE Computer Society. doi:10.1109/SFCS.1975.26.

27 Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

28 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

29 Ryan Williams. Matrix-vector multiplication in sub-quadratic time: (some preprocessing required). In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 995–1001, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1283383.1283490.

30 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 664–673, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591811.

31 I-Hsuan Yang, Chien-Pin Huang, and Kun-Mao Chao. A fast algorithm for computing a longest common increasing subsequence. *Information Processing Letters*, 93(5):249–253, 2005.

32 Daxin Zhu and Xiaodong Wang. A space efficient algorithm for lcis problem. In Guojun Wang, Mohammed Atiquzzaman, Zheng Yan, and Kim-Kwang Raymond Choo, editors, *Security, Privacy, and Anonymity in Computation, Communication, and Storage*, pages 70–77, Cham, 2017. Springer International Publishing.