Quantum Meets Fine-Grained Complexity: Sublinear Time Quantum Algorithms for String Problems

François Le Gall
Nagoya University, Japan

Saeed Seddighin
Toyota Technological Institute at Chicago, IL, USA

Abstract

Longest common substring (LCS), longest palindrome substring (LPS), and Ulam distance (UL) are three fundamental string problems that can be classically solved in near linear time. In this work, we present sublinear time quantum algorithms for these problems along with quantum lower bounds. Our results shed light on a very surprising fact: Although the classic solutions for LCS and LPS are almost identical (via suffix trees), their quantum computational complexities are different. While we give an exact[1] $\tilde{O}(\sqrt{n})$ time algorithm for LPS, we prove that LCS needs at least time $\tilde{\Omega}(n^{2/3})$ even for 0/1 strings.

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

Keywords and phrases Longest common substring, Longest palindrome substring, Quantum algorithms, Sublinear algorithms

Digital Object Identifier 10.4230/LIPIcs.ITCS.2022.97

Related Version *Full Version*: https://arxiv.org/abs/2010.12122

Funding *François Le Gall*: FLG was supported by the JSPS KAKENHI grants Nos. JP16H01705, JP19H04066, JP20H00579 and JP20H04139, and by the MEXT Quantum Leap Flagship Program (MEXT Q-LEAP) grant No. PMXS0120319794.
Saeed Seddighin: SS was supported by a Google Research Gift.

Acknowledgements The authors are grateful to Michael Saks and C. Seshadhri for helpful correspondence.

1 Introduction

Perhaps the earliest questions that were studied in computer science are the algorithmic aspects of string problems. *The edit distance*, *longest common substring*, and *longest palindrome substring* are some of the more famous problems in this category. Efforts to solve these problems led to the discovery of several fundamental techniques such as *dynamic programming*, *hashing algorithms*, and *suffix trees*. These algorithms have numerous applications in several fields including DNA-sequencing, social media, compiler design, anti-virus softwares, etc.

All of the above problems have received significant attention in the classical setting (see, e.g., [32, 12, 13, 7, 14, 19, 24, 6, 28]). Efficient classical algorithms for these problems emerged early on in the 1960s [22]. Moreover, thanks to a series of recent developments in fine-grained complexity [11, 2], we now seem to have a clear understanding of the classical

[1] Throughout the paper, when we say an exact solution, we mean a solution that does not lose anything in the approximation. However, it may be possible that our algorithm succeeds with probability less than 1 in which case we explicitly mention that.

lower bounds as well. Unless plausible conjectures such as SETH[2] are broken, we do not hope for a substantially better algorithm for edit distance. Failure to extend the fine-grained lower bounds to the quantum setting has left some very interesting open questions behind both in terms of quantum complexity and quantum lower bounds.

Despite a plethora of new quantum algorithms for various problems (e.g., [3, 18, 26, 10, 31, 39]), not much attention is given to string problems. Until recently, the only non-trivial quantum algorithm for such problems was the $\tilde{O}(\sqrt{n}+\sqrt{m})$ time algorithm of Ramesh and Vinay [36] for pattern matching where n and m are the sizes of the text and the pattern (we also mention the works [5, 21] that consider string problems with nonstandard queries in the quantum setting). Recently, Boroujeni, Ehsani, Ghodsi, Hajiaghayi, and Seddighin [14] made a clever use of the Grover's search algorithm [27] to obtain a constant factor approximation quantum algorithm for edit distance in truly subquadratic time. Shortly after, it was shown by Chakraborty, Das, Goldenberg, Koucky, and Saks [19] that a similar technique can be used to obtain a classical solution with the same approximation factor. Several improved classical algorithms have been given for edit distance in recent years [9, 29, 15] though it is still an open question if a non-trivial quantum algorithm can go beyond what we can do classically for edit distance.

In this work, we give novel sublinear time quantum algorithms and quantum lower bounds for LCS, LPS, and a special case of edit distance namely *Ulam distance* (UL). All these problems require $\tilde{\Omega}(n)$ time in the classical setting even if approximate solutions are desired. LCS and LPS can be solved in linear time via suffix trees [22] and there is an $O(n \log n)$ time algorithm for Ulam distance [22]. Our results shed light on a very surprising fact: Although the classical solutions for LCS and LPS are almost identical, their quantum computational complexities are different. While we give an exact $\tilde{O}(\sqrt{n})$ time quantum algorithm for LPS, we prove that any quantum algorithm for LCS needs at least time $\tilde{\Omega}(n^{2/3})$ even for 0/1 strings. We accompany this with several sublinear time quantum algorithm for LCS. A summary of our results is given in Tables 1 and 2.

Table 1 Our algorithms are shown in this table.

problem	solution	runtime	reference
longest common substring	exact	$\tilde{O}(n^{5/6})$	Theorem 1
longest palindrome substring	exact	$\tilde{O}(\sqrt{n})$	Theorem 2
longest common substring	$1-\epsilon$ approximation	$\tilde{O}(n^{3/4})$	Theorem 3
longest common substring for non-repetitive strings	exact	$\tilde{O}(n^{3/4})$	Theorem 4
longest common substring for non-repetitive strings	$1-\epsilon$ approximation	$\tilde{O}(n^{2/3})$	Theorem 5
Ulam distance	$1+\epsilon$ approximation	$\tilde{O}(\sqrt{n})$	Theorem 6

1.1 Related work

Our work is very similar in spirit to for instance the work of Ambainis, Balodis, Iraids, Kokainis, Prusis, and Vihrovs [4], where Grover's algorithm is cleverly combined with classical techniques to design more efficient quantum algorithms for dynamic programming problems.

[2] The *strong exponential time hypothesis* states that no algorithm can solve the satisfiability problem in time $2^{n(1-\epsilon)}$.

Table 2 This table includes the quantum lower bounds for the problems we study in this work.

problem	approximation factor	lower bound	reference
longest common substring	$1-\epsilon$ approximation	$\tilde{\Omega}(n^{2/3})$	full version of the paper
longest palindrome substring	$1-\epsilon$ approximation	$\tilde{\Omega}(\sqrt{n})$	full version of the paper
Ulam distance	$1+\epsilon$ approximation	$\tilde{\Omega}(\sqrt{n})$	full version of the paper

This is particularly similar to our approach since we obtain our main results by combining known quantum algorithms with new classical ideas. In the present work, however, we go beyond Grover's algorithm and make use of several other quantum techniques such as element distinctness, pattern matching, amplitude amplification, and amplitude estimation to obtain our improvements. In addition to this, we also develop quantum walks that improve our more general results for some special cases. In particular, our quantum walk for obtaining a $1-\epsilon$ approximate solution for LCS is tight up to logarithmic factors due to a lower-bound we give in the full version of the paper.

Another line of research which is closely related to our work is the study of quantum lower bounds for edit distance [37, 17]. While a SETH-based quadratic lower bound is known for the classical computation of edit distance, quantum lower bounds are not very strong. Recently, a quantum lower bound of $\Omega(n^{1.5})$ was given by Buhrman, Patro, and Speelman [17] under a mild assumption. Still no quantum algorithm better than the state of art classical solution (which runs in time $O(n^2/\log^2 n)$ [32]) is known for edit distance. The reader can find more details in [37].

While not directly related to string algorithms, another investigation of SETH in the quantum setting is the recent work by Aaronson, Chia, Lin, Wang, and Zhang [1] that focuses on the quantum complexity of the closest pair problem, a fundamental problem in computational geometry. Interestingly, the upper bounds obtained in that paper also use an approach based on element distinctness and quantum walks. Despite this, the high-level ideas of our work are substantially different from [1] as we utilize several novel properties of LCS and LPS to design our algorithms.

In the classical sequential setting, LCS and LPS can be solved in linear time [22]. The solutions are almost identical: we first construct suffix trees for the input strings and then find *the lowest common ancestors* for the tree nodes. Ulam distance can also be solved exactly in time $O(n \log n)$ [22] which is the best we can hope for via a comparison-based algorithm [25] or algebraic decision trees [35]. Approximation algorithms running in time $\tilde{O}(n/d+\sqrt{d})$ where d denotes the Ulam distance of the two strings have also been developed [8, 34].

1.2 Preliminaries

Description of LCS, LPS and UD. In the longest common substring problem (LCS), the input consists of two strings and our goal is to find the longest substring[3] which is shared between the two strings. We denote the two input strings by A and B. We assume that A and B have the same length, which we denote by n. We use Σ to denote the alphabet of the

[3] In a substring, the characters are next to each other. In other words, the positions of the characters of a substring should make an interval. This is in contrast to subsequence where the positions can be arbitrary.

strings. For any $\epsilon \in [0, 1)$, we say that an algorithm outputs a $(1-\epsilon)$-approximation of the longest common substring if for any input strings A and B, it outputs a common substring of length at least $(1-\epsilon)d$, where d is the length of the longest common substring of A and B.

In the longest palindrome substring problem (LPS), the goal is to find the longest substring of a given string A which reads the same both forward and backward. The length of A is also denoted by n and its alphabet by Σ. For any $\epsilon \in [0, 1)$, we say that an algorithm outputs a $(1-\epsilon)$-approximation of the longest palindrome substring if for any input string A, it outputs a palindrome substring of length at least $(1-\epsilon)d$, where d is the length of the longest palindrome substring of A.

We say that a string of length n over an alphabet Σ is *non-repetitive* if no character appears twice in the string (note that this can happen only if $|\Sigma| \geq n$). The Ulam distance is a special case of the edit distance in which the input strings are non-repetitive. Let us now define the problem more formally. In Ulam Distance (UD) we are given two non-repetitive strings A and B of length n, and consider how to transform one of them to the other one. For this purpose we allow two basic operations *character addition* and *character deletion*, each at a unit cost and our goal is to minimize the total cost of the transformation[4]. We denote by $\mathrm{ud}(A, B)$ the minimum number of such operations needed to transform A into B. The goal is to compute $\mathrm{ud}(A, B)$, either exactly or approximately. For any $\epsilon \in [0, 1]$, we say that an algorithm outputs a $(1+\epsilon)$-approximation of $\mathrm{ud}(A, B)$ if it outputs some value r such that the inequality $(1-\epsilon) \cdot \mathrm{ud}(A, B) \leq r \leq (1+\epsilon) \cdot \mathrm{ud}(A, B)$ holds.

General definitions and conventions. Throughout the paper, we use notations $\tilde{O}(\cdot)$ and $\tilde{\Omega}(\cdot)$ that hide the polylogarithmic factors in terms of n. We always assume that the size of Σ is polynomial in n and that each character is encoded using $O(\log n)$ bits. The size of Σ thus never appears explicitly in the complexity of our algorithms. We say that a randomized or a quantum algorithm solves a problem like LCS, LPS or UL with high probability if it solves the problem with probability at least $9/10$ (this success probability can be easily amplified to $1 - 1/\mathrm{poly}(n)$ with a logarithmic overhead in the complexity).

For convenience, we often only compute/approximate the size of the solution as opposed to explicitly giving the solution. However, it is not hard to see that for LCS and LPS, the same algorithms can also give an explicit solution with a logarithmic overhead in the runtime. (a solution can be specified by two integers pointing at the interval of the input.)

For a string X, we denote by $X[i, j]$ the substring of X that starts from the i-th character and ends at the j-th character. We say a string X is q-periodic if we have $X_i = X_{i+q}$ for all $1 \leq i \leq |X| - q$. Moreover, the periodicity of a string X is equal to the smallest number $q > 0$ such that X is q-periodic. We also call a non-repetitive string of length n over an alphabet of size n a *permutation* (it represents a permutation of the set Σ).

Quantum access to the inputs. In the quantum setting, we suppose that the input strings A and B can be accessed directly by a quantum algorithm. More precisely, we have an oracle O_A that, for any $i \in \{1, \ldots, n\}$, any $a \in \Sigma$, and any $z \in \{0, 1\}^*$, performs the unitary mapping $O_A \colon |i\rangle|a\rangle|z\rangle \mapsto |i\rangle|a \oplus A[i]\rangle|z\rangle$, where $\oplus$ denotes an appropriate binary operation defined on Σ (e.g., bit-wise parity on the binary encodings of a and $A[i]$). Similarly we have an oracle O_B that, for any $i \in \{1, \ldots, n\}$, any $b \in \Sigma$, and any $z \in \{0, 1\}^*$, performs the unitary mapping $O_B \colon |i\rangle|b\rangle|z\rangle \mapsto |i\rangle|b \oplus B[i]\rangle|z\rangle$. Each call to O_A or O_B can be implemented

[4] Another popular version allows character substitution as a third operation. These two definitions of the Ulam distance only differ by a factor at most 2.

at unit cost. This description corresponds to quantum random access ("QRAM access") to the input, which is the standard model to investigate the complexity of sublinear time quantum algorithms.

2 Results

We present sublinear time quantum algorithms along with quantum lower bounds for LCS, LPS, and UL. For the most part, the novelty of our work is to make use of existing quantum algorithms to solve our problems. For this purpose, we introduce new classical techniques that significantly differ from the conventional methods. However for a special case of LCS, we design a novel quantum walk that leads to an improvement over our more general solution. We give a brief explanation of this technique later in the section. For now, we start by stating the quantum tools that we use in our algorithms.

2.1 Quantum components

Grover's search ([26, 16]). Given a function $f\colon [n] \to \{0,1\}$, Grover's algorithm can find an element $x \in [n]$ such that $f(x) = 1$ or verify if $f(i) = 0$ for all $i \in [n]$. This quantum algorithm runs in time $\tilde{O}(\sqrt{n} \cdot T(n))$ and succeeds with probability $9/10$ (the success probability can be increased to $1 - 1/\text{poly}(n)$ with only a logarithmic overhead). Here, $T(n)$ represents the time complexity of computing $f(i)$ for one given element $i \in [n]$. Additionally, distinguishing between the case where $f(x) = 1$ holds for at least m elements (for some value $1 \leq m \leq n$) and the case where $f(i) = 0$ for all $i \in [n]$ can be done in time $\tilde{O}(\sqrt{n/m} \cdot T(n))$.

Pattern matching ([36]). Let P and S be a pattern and a text of lengths n and m respectively. One can either verify that P does not appear as a substring in S or find the leftmost (rightmost) occurrence of P in S in time $\tilde{O}(\sqrt{n} + \sqrt{m})$ via a quantum algorithm. The algorithm gives a correct solution with probability at least $9/10$.

Element distinctness ([3]). Let X and Y be two lists of size n and $f\colon (X \cup Y) \to \mathbb{N}$ be a function that is used to compare the elements of X and Y[5]. There is a quantum algorithm that finds (if any) an (x, y) pair such that $x \in X$, $y \in Y$ and $f(x) = f(y)$. The algorithm succeeds with probability at least $9/10$ and has running time $\tilde{O}(n^{2/3} \cdot T(n))$, where $T(n)$ represents the time needed to answer to the following question: Given $\alpha, \beta \in X \cup Y$ is $f(\alpha) = f(\beta)$ and if not which one is smaller?

Amplitude amplification ([16]). Let Q be a decision problem and $\mathcal{A}$ be a quantum algorithm that solves Q with one-sided error and success probability $0 < p < 1$ (i.e., on a yes-instance $\mathcal{A}$ always accepts, while on a no-instance $\mathcal{A}$ rejects with probability p). Let T be the runtime of $\mathcal{A}$. One can design a quantum algorithm for Q with runtime $O(T/\sqrt{p})$ that solves Q with one-sided error and success probability at least $9/10$.

[5] In the standard definition of element distinctness, we are given a single list of elements and the goal is to find out if two elements in the list are equal. The present definition, also known as claw finding, is slightly more general – for completeness we discuss how the upper bound $\tilde{O}(n^{2/3} \cdot T(n))$ is obtained for our version as well in Section 4.1.

Amplitude estimation ([16]). Let $\mathcal{A}$ be a quantum algorithm that outputs 1 with probability $0 < p < 1$ and returns 0 with probability $1 - p$. Let T be the time needed for $\mathcal{A}$ to generate its output. For any $\alpha > 0$, one can design a quantum algorithm with runtime $O(T/(\alpha\sqrt{p}))$ that outputs with probability at least 9/10 an estimate $\tilde{p}$ such that $(1-\alpha)p \leq \tilde{p} \leq (1+\alpha)p$.

2.2 LCS and LPS

In this section, we outline the ideas for obtaining sublinear time algorithms for LCS and LPS. We begin as a warm up by giving a simple exact algorithm for LCS that runs in sublinear time when the solution size is small. Next, we explain our techniques for the cases that the solution size is large (at a high-level, this part of the algorithm is very similar in both LCS and LPS). In our algorithms, we do a binary search on the size of the solution. We denote this value by d. Thus, by losing an $O(\log n)$ factor in the runtime, we reduce the problem to verifying if a solution of size at least d exists for our problem instance.

Exact quantum algorithm for LCS (small d). For small d, we use element distinctness to solve LCS. Let $|\Sigma|$ be the size of the alphabet and $v : \Sigma \to [0, |\Sigma| - 1]$ be a function that maps every element of the alphabet to a distinct number in range $0 \ldots |\Sigma| - 1$. In other words, v is a perfect hashing for the characters. We extend this definition to substrings of the two strings so that two substrings t and t' are equal if and only if we have $v(t) = v(t')$. From the two strings, we then make two sets of numbers S_A and S_B each having $n - d + 1$ elements. Element i of set S_A is a pair (A, i) whose value is equal to $v(A[i, i + d - 1])$ and similarly element i of S_B is a pair (B, i) whose value is equal to $v(B[i, i + d - 1])$. The two sets contain elements with equal values if and only if the two strings have a common substring of size d. Therefore, by solving element distinctness for S_A and S_B we can find out if the size of the solution is at least d. Although element distinctness can be solved in time $\tilde{O}(n^{2/3})$ when element comparison can be implemented in $\tilde{O}(1)$ time, our algorithm needs more runtime since comparing elements takes time $\omega(1)$. More precisely, each comparison can be implemented in time $\tilde{O}(\sqrt{d})$ in the following way: In order to compare two elements of the two sets, we can use Grover's search to find out if the two substrings are different in any position and if so we can find the leftmost position in time $\tilde{O}(\sqrt{d})$. Thus, it takes time $\tilde{O}(\sqrt{d})$ to compare two elements, which results in runtime $\tilde{O}(n^{2/3}\sqrt{d})$.

$1 - \epsilon$ approximation for LCS and LPS (large d). We use another technique to solve LCS and LPS when the solution is large. While for LPS, this new idea alone gives an optimal solution, for LCS we need to combine it with the previous algorithm to make sure the runtime is sublinear. Let us focus on LCS first. For a constant $0 < \epsilon < 1$, we define $d' = (1 - \epsilon)d$ and randomly draw a substring of length d' from A. We denote this substring by P. More precisely, we sample an $1 \leq i \leq n - d' + 1$ uniformly at random and set $P = A[i, i + d' - 1]$. Assuming the solution size is at least d, it follows that P is part of a solution with probability at least $\epsilon d/n$. Then, by searching this substring in B, we can find a solution of size d'.

We use the pattern matching quantum algorithm of Ramesh and Vinay [36] to search P in B. This takes time $\tilde{O}(\sqrt{n})$ since $|P| \leq |B| = n$. Moreover, the success probability of this algorithm is $\Omega(d/n)$ and therefore by amplitude amplification, we can improve the success rate to 9/10 by only losing a factor of $O(\sqrt{n/d})$ in the runtime. Thus, if the solution is at least d, we can obtain a solution of size at least $(1 - \epsilon)d$ in time $\tilde{O}(\sqrt{n/d} \cdot \sqrt{n}) = \tilde{O}(n/\sqrt{d})$. Notice that the runtime is sublinear when d is large.

The same technique can be used to approximate LPS. Similarly, we define $d' = (1 - \epsilon)d$ for some constant $0 < \epsilon < 1$ and draw a random substring of size d' from A. With the same

argument, provided that the solution size is at least d, the probability that P is part of an optimal solution is at least $\Omega(d/n)$. We show in the full version of the paper that by searching the reverse of P in its neighbourhood we are able to find a solution of at least d'. This step of the algorithm slightly differs from LCS in that we only search the reverse of P in the area at most d away from P. Thus, both the text and the pattern are of size $O(d)$ and therefore the search can be done in time $\tilde{O}(\sqrt{d})$. By utilizing amplitude amplification, we can obtain an algorithm with runtime $\tilde{O}(\sqrt{n/d} \cdot \sqrt{d}) = \tilde{O}(\sqrt{n})$ and approximation factor $1 - \epsilon$.

From $1 - \epsilon$ approximation to exact solution. We further develop a clever technique to obtain an exact solution with the above ideas. We first focus on LCS to illustrate this new technique. The high-level intuition is the following: After sampling P from A and searching P in B, if the pattern appears only once (or a small number of times) in B then by extending the matching parts of B from both ends we may find a common substring of size d (see Figure 1). Thus intuitively, the challenging case is when there are several occurrences of

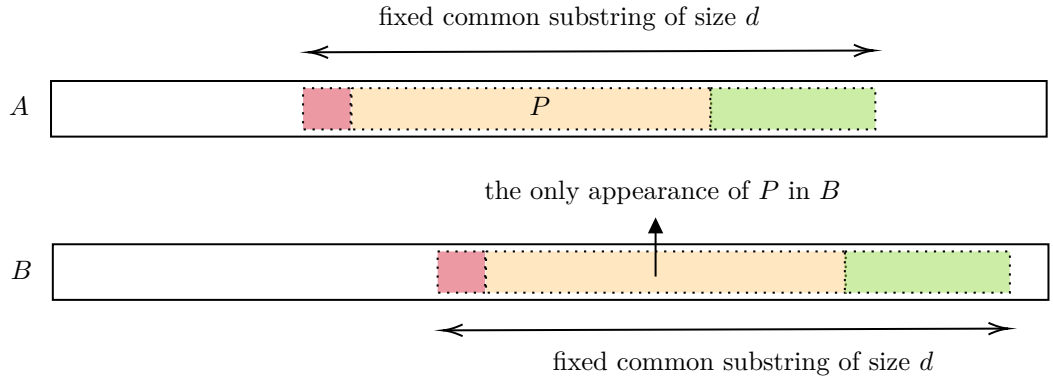


Figure 1 When P appears in B only once.

P in B. The key observation is that since $|P|$ is large and there are several places that P appears in B then they must overlap (see Figure 2). This gives a very convenient approach to tackle the problem. Assume that P appears at positions i and $j > i$ of B and these parts are overlapping. This implies that P is certainly $(j - i)$-periodic. To see how this enables us to solve the problem, assume that P is x-periodic and that a large continuous area of B is covered by occurrences of P. It follows that except for the boundary cases, P appears as parts of that interval that are exactly x units away. Therefore, by detecting such an interval and computing the periodicity of P, one can determine **all** occurrences of P in the interval almost as quickly as finding one occurrence of P.

Since the techniques are involved, we defer the details to Section 3 and here we just mention some intuition about its complexity analysis: It follows from the above observations that when several occurrences of P cover an entire interval I of B, then we only need to consider $O(1)$ places in I that may correspond to an optimal solution. Moreover, the length of every such interval is at least d. Thus, there are only n/d places of B that we need to take into account in our algorithm. Therefore at a high-level, when d is large (say $\Omega(n)$), the only non-negligible cost we pay is for the pattern matching which takes time $\tilde{O}(\sqrt{n})$. We show in Section 3 that as d becomes smaller the runtime increases; more precisely, when $d = n/\alpha$ the runtime increases by a factor of $O(\sqrt{\alpha})$ and thus the overall runtime of the algorithm is $\tilde{O}(\sqrt{n} \cdot \sqrt{\alpha}) = \tilde{O}(n/\sqrt{d})$. A very similar argument can be used to shave the $1 - \epsilon$ approximation factor from the LPS algorithm as well.

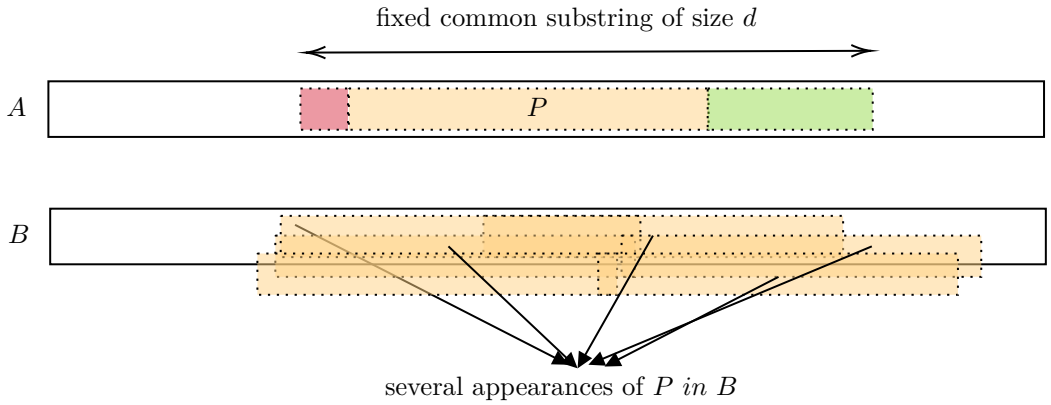


Figure 2 When P appears several times in B.

To summarize, we obtain the following theorems. Theorem 1 combines the two algorithms we just described: if d is larger than $n^{1/3}$ we use the second algorithm with runtime $\tilde{O}(n/\sqrt{d})$ otherwise we use the first algorithm with runtime $\tilde{O}(n^{2/3}\sqrt{d})$.

▶ **Theorem 1.** *The longest common substring of two strings of size n can be computed with high probability by a quantum algorithm in time $\tilde{O}(n^{5/6})$.*

▶ **Theorem 2.** *The longest palindrome substring of a string of size n can be computed with high probability by a quantum algorithm in time $\tilde{O}(\sqrt{n})$.*

We accompany Theorems 1 and 2 with quantum lower bounds (all the lower bounds are proven in the full version of the paper). Intuitively, obtaining a solution with time better than $\tilde{O}(\sqrt{n})$ is impossible for either problem due to a reduction to searching unordered sets. This makes our solution for LPS optimal up to subpolynomial factors. For LCS, an improved lower bound of $\tilde{\Omega}(n^{2/3})$ can be obtained via a reduction from element distinctness. However, the gap is still open between our upper bound of $\tilde{O}(n^{5/6})$ and lower bound of $\tilde{\Omega}(n^{2/3})$. Thus, we aim to improve our upper bound by considering approximate solutions and special cases. In the following, we briefly explain these results.

Improved $1-\epsilon$ approximation for LCS. One way to obtain a better algorithm for LCS is by considering $1-\epsilon$ approximation algorithms. Note that the quantum algorithm for element distinctness is based on quantum walks, and our solution for LCS is obtained via a reduction to element distinctness. The runtime of quantum walks can be improved when multiple solutions are present for a problem. If instead of a solution of size d which is exact, we resort to solutions of size $(1-\epsilon)d$, we are guaranteed to have at least ϵd solutions. Thus, intuitively this should help us improve the runtime of our algorithm.

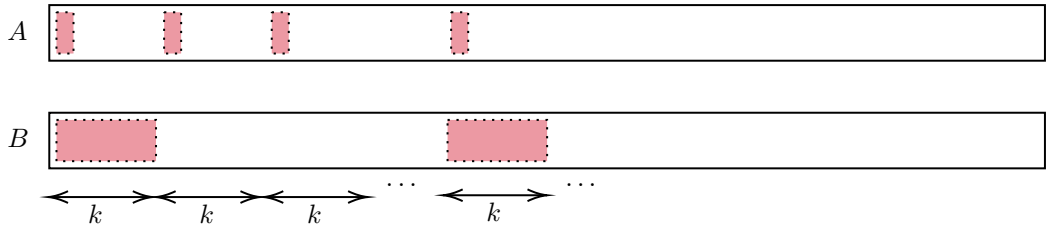


Figure 3 $k = \epsilon\sqrt{d}$. Only the elements colored in red are included in the two sets.

Although the intuition comes from the inner workings of the quantum techniques, we are actually able to improve the runtime with a completely combinatorial idea. For small d, instead of constructing two sets S_A and S_B with $n-d+1$ elements, we construct two sets of size $O(n/\sqrt{d})$ and prove that if the two sets have an element in common, then there is a solution of size at least $(1-\epsilon)d$. The construction of the two sets is exactly the same as the construction of S_A and S_B except that only some of the elements are present in the new subsets (see Figure 3 for an illustration of the construction).

We prove that since each set now has size $O(n/\sqrt{d})$, if the two strings have an LCS of size d then a solution of size $(1-\epsilon)d$ can be found in time $\tilde{O}((n/\sqrt{d})^{2/3}\sqrt{d}) = \tilde{O}(n^{2/3}d^{1/6})$. This combined with the $\tilde{O}(n/\sqrt{d})$ algorithm for large d gives us a $1-\epsilon$ approximation algorithm with runtime $\tilde{O}(n^{3/4})$.

▶ **Theorem 3.** *For any constant $0 < \epsilon < 1$, the longest common substring of two strings of size n can be approximated within a factor $1-\epsilon$ with high probability by a quantum algorithm in time $\tilde{O}(n^{3/4})$.*

LCS for non-repetitive strings. For all string problems, a special case which is of particular interest is when the characters are different. For instance, although DNA's consist of only 4 characters, one can make the representation more informative by assigning a symbol to every meaningful block of the sequence. This way, there is only little chance a character appears several times in the sequence. Similarly, in text recognition, one may rather represent every word with a symbol of the alphabet resulting in a huge alphabet and strings with low repetitions. These scenarios are motivating examples for the study of edit distance and longest common subsequence under this assumption, known respectively as Ulam distance [8, 20, 34] and longest increasing subsequence (which has been the target of significant research by the string algorithms community – see, e.g., [33] for references).

We thus consider LCS for input strings A and B that are non-repetitive (an important special case is when A and B are permutations). We show that there exists an $\tilde{O}(n^{3/4})$-time quantum algorithm for exact LCS and an $\tilde{O}(n^{2/3})$-time quantum algorithm for approximate LCS. This significantly improves the generic results of Theorems 1 and 3 and, for approximate LCS, this matches (up to possible polylogarithmic factors) the lower bound of the problem.

▶ **Theorem 4.** *The longest common substring of two non-repetitive strings of size n can be computed with high probability by a quantum algorithm in time $\tilde{O}(n^{3/4})$.*

▶ **Theorem 5.** *For any constant $0 < \epsilon < 1$, the longest common substring of two non-repetitive strings of size n can be approximated within a factor $1-\epsilon$ with high probability by a quantum algorithm in time $\tilde{O}(n^{2/3})$.*

The improvements for non-repetitive strings are obtained by improving the complexity of the first part of the algorithm used for general LCS from $\tilde{O}(n^{2/3}\sqrt{d})$ to $\tilde{O}(n^{2/3}+\sqrt{nd})$ for the case of exact LCS, and from $\tilde{O}(n^{2/3}d^{1/6})$ to $\tilde{O}(n^{2/3}/d^{1/3}+\sqrt{n})$ for approximate LCS. We now briefly describe how we achieve such improvements.

Let us first consider exact LCS. The $\tilde{O}(n^{2/3}\sqrt{d})$-time quantum algorithm described above was based on a "black-box reduction" to element distinctness, i.e., we used the quantum algorithm for element distinctness (which is based, as already mentioned, on a technique known as quantum walk) as a black-box. In comparison, our new algorithm is constructed by designing a quantum walk especially tailored for our problem. More precisely, we use the approach by Magniez, Nayak, Roland and Santha [30] to design a quantum walk over the Johnson graph (more precisely, we work with a graph defined as the direct product of two Johnson graphs, which is more convenient for our purpose).

We say that a pair $(i,j) \in [n-d+1] \times [n-d+1]$ is marked if there is a common substring of length d that starts at position i in A and position j in B, i.e., $A[i,i+d-1] = B[j,j+d-1]$. The goal of our quantum walk is to find a pair of subsets (R_1, R_2) where R_1 and R_2 are two subsets of $[n-d+1]$ of size r (for some parameter r) such that there exists a marked pair (i,j) in $R_1 \times R_2$. Note that since the strings are non-repetitive, for two random subsets R_1, R_2, the expected number of pairs $(i,j) \in R_1 \times R_2$ such that $A[i] = B[j]$ is roughly $\Theta(r^2/n)$. A simple but crucial observation is that only those pairs can be marked since marked pairs should agree on their first character. Thus for two random subsets R_1, R_2 we can check if there exists a marked pair in $R_1 \times R_2$ in expected time $\tilde{O}(\sqrt{r^2/n})$ using Grover search, since we have only $\Theta(r^2/n)$ candidates for marked pairs. This significantly improves over the upper bound $\tilde{O}(\sqrt{r^2})$ we would get without using the assumption that the strings are non-repetitive. This is how we can improve the complexity down to $\tilde{O}(n^{2/3} + \sqrt{nd})$. Note that a technical difficulty that we need to overcome is guaranteeing that the running time of the checking procedure is small not only for random subsets but for also all (R_1, R_2), i.e., we need a guarantee on the worst-case running time of the checking procedure. We solve this issue by disregarding the pairs of subsets (R_1, R_2) that contain too many candidates – we are able to prove that the impact is negligible by using concentration bounds.

The improvement for approximate LCS uses a very similar idea. The main difference is that now we consider a pair $(i,j) \in [n-d+1] \times [n-d+1]$ marked if there is a common substring of length $\lceil (1-\epsilon)d \rceil$ that starts at position i in A and position j in B. Since the fraction of marked pairs increases by a factor ϵd we obtain a further improvement. Analyzing the running time of the resulting quantum walk shows that we obtain overall time complexity $\tilde{O}(n^{2/3}/d^{1/3} + \sqrt{n})$.

2.3 Ulam distance

Finally, we present a sublinear time quantum algorithm that computes a $(1+\epsilon)$-approximation of the Ulam distance (i.e., the edit distance for non-repetitive strings).

▶ **Theorem 6.** *For any constant $\epsilon > 0$, there exists a quantum algorithm that computes with high probability a $(1+\epsilon)$-approximation of the Ulam distance of two non-repetitive strings in time $\tilde{O}(\sqrt{n})$.*

In comparison, classical algorithms require linear time even for computing a constant-factor approximation of the Ulam distance when the distance is small (see, e.g., [8] for a discussion of the classical lower bounds). Theorem 6 thus shows that while for general strings it is still unknown whether a quantum speed-up is achievable for the computation of the edit distance, we can obtain a quadratic speed-up for non-repetitive strings. In the full version of the paper, we show a quantum lower bound that matches (up to possible polylogarithmic factors) the upper bound of Theorem 6. Since it is easy to show that any quantum algorithm that computes the Ulam distance exactly requires $\Omega(n)$ time, our results are thus essentially optimal.

Let us now describe briefly how the quantum algorithm of Theorem 6 is obtained. Our approach is based on a prior work by Naumovitz, Saks, and Seshadhri [34] that showed how to construct, for any constant $\epsilon > 0$, a classical algorithm that computes a $(1+\epsilon)$-approximation of the Ulam distance and runs in sublinear time when $\mathrm{ud}(A,B)$ is large (the running time becomes linear when $\mathrm{ud}(A,B)$ is small). The core technique is a variant of the Saks-Seshadhri algorithm for estimating the longest increasing sequence from [38], which can be used to construct a binary "indicator" that outputs 1 with probability p and 0 with probability $1-p$,

for some value p that is related to the value of $\mathrm{ud}(A, B)$. Conceptually, the approach is based on estimating the value of $\mathrm{ud}(A, B)$ from this indicator using a hierarchy of gap tests and estimators, each with successively better run time. This results in a fairly complex algorithm.

At a high level, our strategy is applying quantum amplitude estimation on the classical indicator UlamIndic to estimate p and thus $\mathrm{ud}(A, B)$. Several technical difficulties nevertheless arise since the indicator requires a rough initial estimation of $\mathrm{ud}(A, B)$ to work efficiently. To solve these difficulties, we first construct a quantum gap test based on quantum amplitude estimation that enables to test efficiently if the success probability of an indicator is larger than some given threshold q or smaller than $(1-\eta)q$ for some given gap parameter. We then show how to apply this gap test to the indicator UlamIndic with successively better initial estimates of $\mathrm{ud}(A, B)$ in order to obtain a $(1+\epsilon)$-approximation of $\mathrm{ud}(A, B)$ in $\tilde{O}(\sqrt{n})$ time.

Organization of the paper. The details of our quantum algorithms solving LCS for general strings (Theorems 1 and 3) are given in Section 3. Section 4 presents our quantum algorithms solving LCS for non-repetitive strings (Theorems 4 and 5). Due to space constraints, the details of our other results (the quantum algorithms for LPS and the Ulam distance, and our lower bounds) are omitted from this version.

3 Longest Common Substring

Recall that in LCS, we are given two strings A and B of size n and the goal is to find the largest string t which is a substring of both A and B. This problem can be solved in time $O(n)$ via suffix tree in the classical setting. In this section, we give upper bounds for the quantum complexity of this problem (lower bounds are discussed in the full version of the paper). We first begin by giving an algorithm for exact LCS that runs in time $\tilde{O}(n^{5/6})$ in Section 3.1. We then give an algorithm for approximate LCS that runs in time $\tilde{O}(n^{3/4})$ in Section 3.2.

3.1 Quantum algorithm for exact LCS

In our algorithm, we do a binary search on the size of the solution. We denote this value by d. Thus, by losing an $O(\log n)$ factor in the runtime, we reduce the problem to verifying if a substring of size d is shared between the two strings. Our approach is twofold; we design separate algorithms for small and large d.

3.1.1 Quantum algorithm for small d

Our algorithm for small d is based on a reduction to element distinctness. Let $|\Sigma|$ be the size of the alphabet and $v : \Sigma \to [0, |\Sigma| - 1]$ be a function that maps every element of the alphabet to a distinct number in range $0 \dots |\Sigma| - 1$. In other words, v is a perfect hashing for the characters. We extend this definition to substrings of the two strings. For a string t, we define $v(t)$ as follows:

$$v(t) = \sum_{i=1}^{|t|} v(t_i)|\Sigma|^{i-1}.$$

It follows from the definition that two strings t and t' are equal if and only if we have $v(t) = v(t')$.

From the two strings, we make two sets of numbers S_A and S_B each having $n - d + 1$ elements. Element i of set S_A is a pair (A, i) whose value is equal to $v(A[i, i + d - 1])$ and similarly element i of S_B is a pair (B, i) whose value is equal to $v(B[i, i + d - 1])$. The

two sets contain elements with equal values if and only if the two strings have a common substring of size d. Therefore, by solving element distinctness for S_A and S_B we can find out if the size of the solution is at least d. Although element distinctness can be solved in time $\tilde{O}(n^{2/3})$ when element comparison can be implemented in $\tilde{O}(1)$ time, our algorithm needs more runtime since comparing elements takes time $\omega(1)$. More precisely, each comparison can be implemented in time $\tilde{O}(\sqrt{d})$ in the following way: In order to compare two elements of the two sets, we can use Grover's search to find out if the two substrings are different in any position and if so we can find the leftmost position in time $\tilde{O}(\sqrt{d})$. Thus, it takes time $\tilde{O}(\sqrt{d})$ to compare two elements which results in runtime $\tilde{O}(n^{2/3}\sqrt{d})$. We summarize this result in the following lemma.

▶ **Lemma 7.** *There exists a quantum algorithm that runs in time $\tilde{O}(n^{2/3}\sqrt{d})$ and verifies with probability $9/10$ if there is a common substring of length d between the two strings.*

3.1.2 Quantum algorithm for large d

We now present a quantum algorithm that runs in time $\tilde{O}(n/\sqrt{d})$ and verifies if there is a common substring of length d between the two strings.

General description of the algorithm. We say that a character of A is marked if it appears among the first $\lfloor d/3 \rfloor$ characters of some substring of length d shared between A and B. For example, If there is exactly one common substring of length d, there are precisely $\lfloor d/3 \rfloor$ marked characters but we may have more marked characters as the number of common subsequences of length d between A and B increases. In our algorithm, we sample a substring of length $2\lfloor d/3 \rfloor$ of A and a substring of length d of B. We call the substring sampled from A the pattern and denote it by P and denote the substring sampled from B by S. We denote the intervals of A and B that correspond to P and S by $[\ell_P, r_P]$ and $[\ell_S, r_S]$ respectively. That is, $A[\ell_P, r_P] = P$ and $B[\ell_S, r_S] = S$. We say that the pair (P, S) is good if the following conditions hold (see Figure 4 for an illustration):

- There exists a pair (i, j) such that $A[i, i+d-1] = B[j, j+d-1]$ is a common subsequence of size d between the two strings.
- $i \leq \ell_P$ and $\ell_P - i < \lfloor d/3 \rfloor$.
- $\ell_S \leq j - i + \ell_P \leq j - i + r_P \leq r_S$.

It directly follows that if (P, S) is a good pair, ℓ_P has to be a marked character. When we fix a good pair (P, S) and we refer to the optimal solution, we mean the common subsequence of size d made by $A[i, i + d - 1]$ and $B[j, j + d - 1]$.

If there is no substring of length d shared between A and B, then obviously no good pair exists. Let us now compute the probability of sampling such P and S under the assumption that there is a common substring of length d shared between A and B. There are at least $\lfloor d/3 \rfloor$ marked characters in A and thus with probability $\Omega(d/n)$ we sample the right pattern. Moreover, the corresponding substring of solution in B has at least $\lfloor d/3 \rfloor$ positions such that if we sample S starting from those positions the pair (P, S) is good. Thus, with probability $\Omega(d^2/n^2)$ we sample a good pair (P, S).

We describe below a quantum procedure that given a good pair (P, S), constructs with probability at least $1 - 1/\text{poly}(n)$ a common substring of length d in time $\tilde{O}(\sqrt{d})$. By first sampling (P, S) and then running this procedure, we get a one-sided procedure that verifies if there exists a common substring of length d shared between A and B with probability $\Omega(d^2/n^2)$ in time $\tilde{O}(\sqrt{d})$. With amplitude amplification, we can thus construct a quantum algorithm that verifies if there exists a common substring of length d with high probability in time $\tilde{O}(\sqrt{n^2/d^2} \cdot \sqrt{d}) = \tilde{O}(n/\sqrt{d})$. We summarize this result in the following lemma.

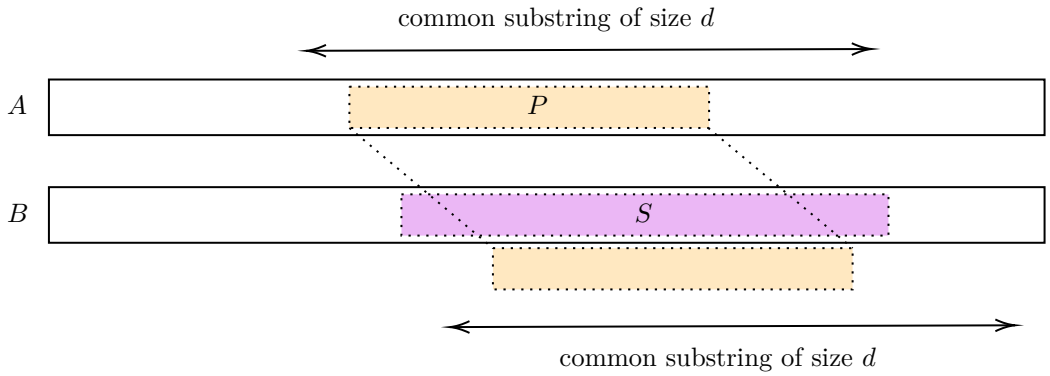


Figure 4 An example of a good pair (P, Q). The orange part of S shows the part of S that matches with P.

▶ **Lemma 8.** *There exists a quantum algorithm that runs in time $\tilde{O}(n/\sqrt{d})$ and verifies with probability 9/10 if there is a common substring of length d between the two strings.*

Constructing a common substring from a good pair. From here on, we assume that (P, S) is good and describe how to construct a common substring of length d in time $\tilde{O}(\sqrt{d})$.

We aim to find positions of S that match with P. For this purpose, we use the string matching algorithm of Ramesh and Vinay [36] that searches P in S in time $\tilde{O}(\sqrt{|S|} + \sqrt{|P|})$. If there is no match, then we can conclude that S and P do not meet our property and therefore this should not happen. If there is one such position, then we can detect in time $\tilde{O}(\sqrt{d})$ if by extending this matching from two ends we can obtain a common substring of size d as follows: We use Grover's search to find the left-most (up to d characters away) position where the two substrings differ when extending them from right. We do the same from the left and it tells us if this gives a common substring of size at least d. Each search takes time $\tilde{O}(\sqrt{d})$. We refer the reader to Figure 5 for a pictorial illustration.

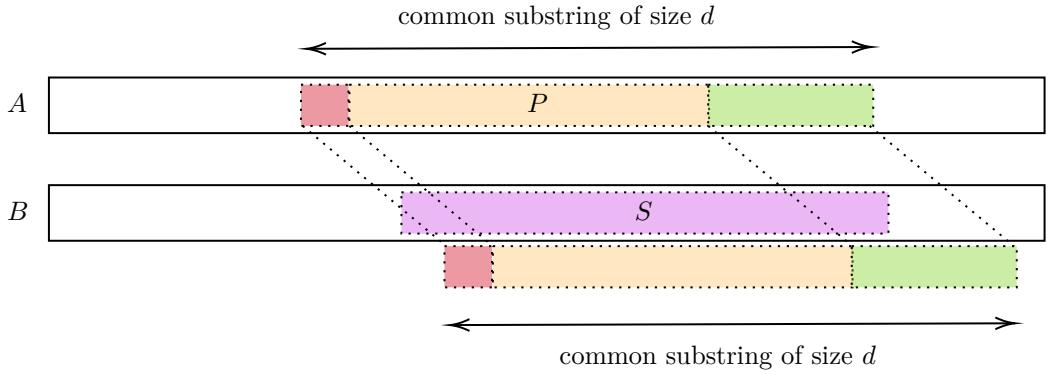


Figure 5 If P matches S in only one position, by extending the two ends in the two strings we can find a common substring of length d. The orange part of S shows the part of S that matches with P. Also, the red and green parts and extensions from left and right for the matched parts.

More generally, the above idea works when there are only $O(1)$ many positions of S that match with P. However, we should address the case that P appears many times in S. In the following we discuss how this can be handled. In time $\tilde{O}(\sqrt{d})$ we first find the leftmost and rightmost positions of S that match with P. Let the starting index of the

leftmost match be ℓ and the starting index of the rightmost match be r. In other words, $P = S[\ell, \ell + |P| - 1] = S[r, r + |P| - 1]$. Since $|S| = d$ and we have $|P| = 2\lfloor d/3 \rfloor$ then the two substrings overlap. Therefore, P as well as the entire string $S[\ell, r + |P| - 1]$ is $(r - \ell)$-periodic. This is the key property on which our algorithm will be based.

Let us denote the substring $S[\ell, r + |P| - 1]$ by T. We extend both P and T from left and right up to a distance of $2d$ in the following way: We increase the index of the ending point so long as the substring remains $(r - \ell)$-periodic. We also stop if we move the ending point by more than $2d$. We do the same for the starting point; we decrease the starting point so long as the substring remains $(r - \ell)$-periodic and up to at most a distance of $2d$. Since we bound the maximum change by $2d$, then this can be done in time $\tilde{O}(\sqrt{d})$ via Grover's search. Let us denote the resulting substrings by $A[\alpha, \beta]$ and $B[\alpha', \beta']$. The following observation enables us to find the solution in time $\tilde{O}(\sqrt{d})$: one of the following three cases holds for the optimal solution.

1. The starting index of the corresponding optimal solution in A is smaller than α. Since the matched parts of the solution are both in the periodic segments, this implies that the first non-periodic indices (when going backwards from the matched parts) in the two solution intervals are $A[\alpha - 1]$ and $B[\alpha' - 1]$. As a consequence the corresponding character of $A[\alpha]$ in B is $B[\alpha']$ (Figure 6a).
2. The ending index of the corresponding optimal solution in A is larger than β and thus (with a similar argument as above) the corresponding character of $A[\beta]$ in B is $B[\beta']$ (Figure 6b).
3. The corresponding optimal solution in A is in the interval $A[\alpha, \beta]$ and thus it is $(r - \ell)$-periodic (Figure 6c).

In all three cases, we can find the optimal solution in time $\tilde{O}(\sqrt{d})$. For the first two cases, we know one correspondence between the two common substrings of length d. More precisely, in Case 1 we know that $A[\alpha]$ is part of the solution and this character corresponds to $B[\alpha']$. Thus, it suffices to do a Grover's search from two ends to extend the matching in the two directions. Similarly in Case 2 we know that $A[\beta]$ corresponds to $B[\beta']$ and thus via a Grover's search we find a common substring of length d in the two strings.

Finally, for Case 3 we point out that since the entire solution lies in intervals $A[\alpha, \beta]$ and $B[\alpha', \beta']$ then we have $\beta \geq \alpha + d - 1$ and $\beta' \geq \alpha' + d - 1$. Notice that both intervals $A[\alpha, \beta]$ and $B[\alpha', \beta']$ are $(r - \ell)$-periodic. Let the starting position of P in A be x. For a fixed position y in the interval $[\alpha', \beta']$ that matches with P, if we extend this matching from the two ends we obtain a solution of size $\min\{x - \alpha, y - \alpha'\} + \min\{\beta - x + 1, \beta' - y + 1\}$. This means that extending one of the following matchings between P and S gives us a common subsequence of size d:

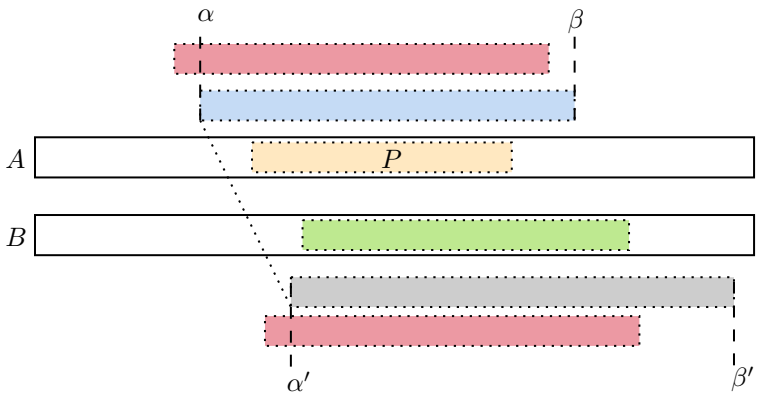
- An optimal solution when $x - \alpha \geq y - \alpha'$: the rightmost matching in the interval $B[\alpha', \alpha' + |P| + (x - \alpha)]$, or
- An optimal solution when $x - \alpha \leq y - \alpha'$: the leftmost matching in the interval $B[\alpha' + (x - \alpha), \beta']$.

Each matching can be found in time $\tilde{O}(\sqrt{d})$ and similar to the ideas explained above we can extend each matching to verify if it gives us a common substring of size d in time $\tilde{O}(\sqrt{d})$.

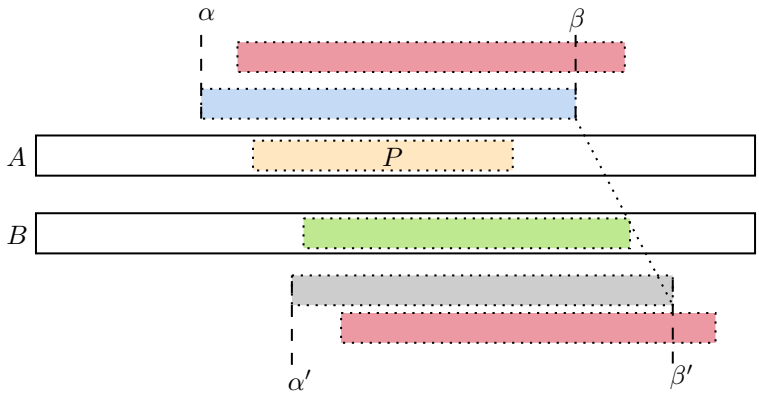
3.1.3 Combining the two algorithms

Combining Lemmas 7 and 8 gives us a solution in time $\tilde{O}(n^{5/6})$.

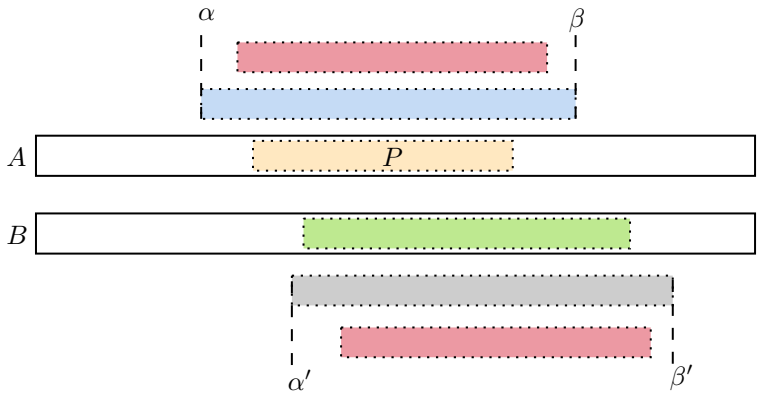
▶ **Theorem 1** (repeated). *The longest common substring of two strings of size n can be computed with high probability by a quantum algorithm in time $\tilde{O}(n^{5/6})$.*



(a) First case.



(b) Second case.



(c) Third case.

Figure 6 The three cases considered. The red intervals correspond to the longest common substring. The yellow interval is P and the green interval is $S[\ell, r + |P| - 1]$. The blue and grey intervals are extensions of the yellow and green intervals so long as they remain $(r - \ell)$-periodic.

Proof. We do a binary search on d (size of the solution). To verify a given d, we consider two cases: if $d < n^{1/3}$ we run the algorithm of Lemma 7 and otherwise we run the algorithm of Lemma 8. Thus, the overall runtime is bounded by $\tilde{O}(n^{5/6})$. ◀

3.2 Quantum algorithm for approximate LCS

In the following, we show that if an approximate solution is desired then we can improve the runtime down to $\tilde{O}(n^{3/4})$. Similar to the above discussion, we use two algorithms for small d and large d. Our algorithm for large d is the same as the one we use for exact solution. For small d we modify the algorithm of Lemma 7 to improve its runtime down to $\tilde{O}(n^{2/3}d^{1/6})$. This is explained in Lemma 9.

▶ **Lemma 9.** *For any constant $0 < \epsilon < 1$, there exists a quantum algorithm that runs in time $\tilde{O}(n^{2/3}d^{1/6})$ and if the two strings share a common substring of length d, finds a common substring of length $(1-\epsilon)d$.*

Proof. Similar to Lemma 7, we use element distinctness for this algorithm. We make two sets S_A and S_B and prove that they share two equal elements if and only if their corresponding substrings of size $(1-\epsilon)d$ are equal. The difference between this algorithm and the algorithm of Lemma 7 is that here S_A and S_B contain $O(n/\sqrt{\epsilon d})$ elements instead of $n-d+1$ elements.

Let $k = \sqrt{\epsilon d}$. We break both strings into blocks of size k and make the two sets in the following way (see Figure 7 for an illustration):

- For each i such that $i \bmod k = 1$, we put element (A, i) in set S_A.
- For each i such that $\lceil i/k \rceil \bmod k = 1$ we put element (B, i) in set S_B.

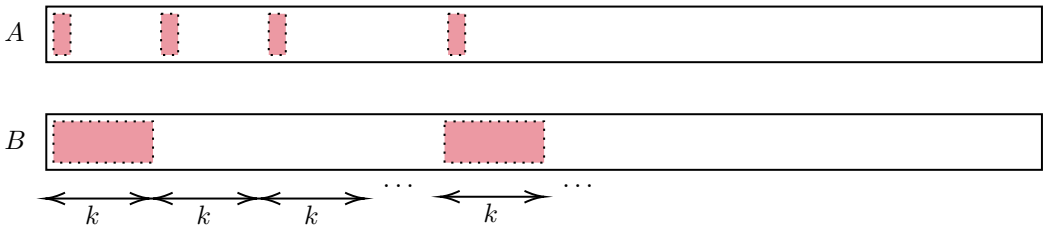


Figure 7 $k = \sqrt{\epsilon d}$. Only the elements colored in red are included in the two sets.

It follows that among the ϵd first characters of the solution, there is one position which is included in S_A such that its corresponding position in B is also included in S_B. Thus, if we define the equality between two elements (A, i) and (B, j) as $A[i, i + (1-\epsilon)d - 1] = B[j, j + (1-\epsilon)d - 1]$ then by solving element distinctness for S_A and S_B we can find a desired solution. Since we have $O(n/\sqrt{d})$ elements in each set then the overall runtime is bounded by $\tilde{O}((n/\sqrt{d})^{2/3}\sqrt{d}) = \tilde{O}(n^{2/3}d^{1/6})$. ◀

Lemmas 8 and 9 lead to a quantum algorithm for LCS with approximation factor $1-\epsilon$ and runtime $\tilde{O}(n^{3/4})$.

▶ **Theorem 3** (repeated). *For any constant $0 < \epsilon < 1$, the longest common substring of two strings of size n can be approximated within a factor $1-\epsilon$ with high probability by a quantum algorithm in time $\tilde{O}(n^{3/4})$.*

Proof. We do a binary search on d. To verify a given d, we consider two cases: if $d < n^{1/2}$ we run the algorithm of Lemma 9 and otherwise we run the algorithm of Lemma 8. Thus, the overall runtime is bounded by $\tilde{O}(n^{3/4})$. ◀

4 Longest Common Substring for Non-Repetitive Strings

In this section, we consider LCS for input strings A and B that are non-repetitive (e.g., permutations). We show that there exists an $\tilde{O}(n^{3/4})$-time quantum algorithm that computes an exact LCS and an $\tilde{O}(n^{2/3})$-time quantum algorithm that computes a $(1-\epsilon)$-approximation of LCS. This significantly improves the generic results of the previous section.

4.1 Quantum walks

We first review the quantum walk approach that solves element distinctness and several similar problems. We actually consider a variant of the standard approach that is especially well suited for our purpose: instead of considering a quantum walk on a Johnson graph we consider a quantum walk on the direct product of two Johnson graphs. While quantum walks on such graphs have also been used by Buhrman and Špalek in [18], their implementation used Szegegy's version of quantum walks [40]. One noteworthy aspect of our approach is that we rely on the stronger version of quantum walks developed by Magniez, Nayak, Roland and Santha [30] (this is crucial for the walk we present in Section 4.2 and 4.3; using Szegegy's version does not lead to an efficient algorithm because the checking cost of our quantum walk is too high).

Let S be a finite set and r be any integer such that $r \leq |S|$. Let us denote by $\mathcal{T}_r$ the set of all couples (R_1, R_2) where both R_1 and R_2 are subsets of S of size r. The elements of $\mathcal{T}_r$ are called the states of the walk. Let $\mathcal{T}_r^* \subseteq \mathcal{T}_r$ be some subset. The elements of $\mathcal{T}_r^*$ are called the marked states. The quantum walk approach associates to each $(R_1, R_2) \in \mathcal{T}_r$ a data structure $D(R_1, R_2)$. Three types of costs are associated with D. The setup cost $s(r)$ is the cost to set up the data structure $D(R_1, R_2)$ for any $(R_1, R_2) \in \mathcal{T}_r$. The update cost $u(r)$ is the cost to update $D(R_1, R_2)$ for any $(R_1, R_2) \in \mathcal{T}_r$: converting $D(R_1, R_2)$ to $D((R_1 \setminus a) \cup \{a'\}, (R_2 \setminus b) \cup \{b'\})$ for any $a \in R_1$, $a' \in S \setminus R_1$, $b \in R_2$ and $b' \in S \setminus R_2$. The checking cost $c(r)$ is the cost of checking, given $D(R_1, R_2)$ for any $(R_1, R_2) \in \mathcal{T}_r$, if $(R_1, R_2) \in \mathcal{T}_r^*$. By combining amplitude amplification and quantum walks, we can find with high probability one marked state (R_1, R_2). We will use in this paper the following statement (we refer to [30] for details) of this approach.

▶ **Proposition 10.** *Consider any value $r \in \{1, \ldots, |S|\}$ and any constant $\gamma > 0$. Assume that either $\mathcal{T}_r^* = \emptyset$ or $\frac{|\mathcal{T}_r^*|}{|\mathcal{T}_r|} \geq \delta$ holds for some known value $\delta > 0$. There exists a quantum algorithm that always rejects if $\mathcal{T}_r^* = \emptyset$, outputs a set $(R_1, R_2) \in \mathcal{T}_r^*$ with probability at least $1 - \frac{1}{n^\gamma}$ otherwise, and has complexity*

$$\tilde{O}\left(s(r) + \sqrt{1/\delta}\left(\sqrt{r} \times u(r) + c(r)\right)\right).$$

Let us now explain how to solve element distinctness using this framework. Let $x_1, \ldots, x_n$ denote the n elements of the first list, and $y_1, \ldots, y_n$ denote the n elements of the second list of the input of element distinctness. Let us first discuss the complexity in the standard access model, where we assume that each element is encoded using $O(\log n)$ bits and that given $i \in [n]$ we can obtain all the bits of x_i and y_i in $O(\log n)$ time. We set $S = \{1, \ldots, n\}$ and say that a state $(R_1, R_2) \in \mathcal{T}_r$ is marked if there exists a pair $(i, j) \in R_1 \times R_2$ such that $x_i = y_j$. Easy calculations show that the fraction of marked states is $\Omega(r^2/n^2)$. By using an appropriate data structure that allows insertion, deletion and lookup operations in polylogarithmic time (see Section 6.2 of [3] for details about how to construct such a data structure) to store the elements x_i for all $i \in R_1$, and using another instance of this data structure to store the elements y_j for all $j \in R_2$, we can perform the setup operation in

time $s(r) = \tilde{O}(r)$ and perform each update operation in time $u(r) = \tilde{O}(1)$. Each checking operation can be implemented in time $c(r) = \tilde{O}(\sqrt{r})$ as follows: perform a Grover search over R_1 to check if there exists $i \in R_1$ for which $x_i = y_j$ for some $j \in R_2$ (once i is fixed the later property can be checked in polylogarithmic time using the data structure representing R_2).[6] From Proposition 10, the overall time complexity of the quantum walk is thus

$$\tilde{O}\left(r + \frac{n}{r}\left(\sqrt{r} \times 1 + \sqrt{r}\right)\right).$$

By taking $r = n^{2/3}$ we get time complexity $\tilde{O}(n^{2/3})$. As discussed in Section 6.1 of [3], the above data structure, and thus the whole quantum walk can also be implemented in the same way in the (weaker) comparison model: if given any pair of indices $(i, j) \in [n] \times [n]$ we can decide in $T(n)$ time whether $x_i < y_j$ or $x_i \geq y_j$, then the time complexity of the implementation is $\tilde{O}(n^{2/3}T(n))$.

4.2 Quantum algorithm for exact LCS of non-repetitive strings

We set $S = \{1, \ldots, n - d + 1\}$. Let us write $\alpha = 54 \log n$. For any state $(R_1, R_2) \in \mathcal{T}_r$ we define the data structure $D(R_1, R_2)$ as follows:

- $D(R_1, R_2)$ records all the values $A[i]$ and $B[j]$ for all $i \in R_1$ and $j \in R_2$ using the same data structure as the data structure considered above for element distinctness;
- additionally, $D(R_1, R_2)$ records the number of pairs $(i, j) \in R_1 \times R_2$ such that $A[i] = B[j]$, and stores explicitly all these pairs (using a history-independent data structure updatable in polylogarithmic time, which can be constructed based on the data structure from [3]).

We define the set of marked states $\mathcal{T}_r^*$ as follows. We say that (R_1, R_2) is marked if the following two conditions hold:

(i) there exists a pair $(i, j) \in R_1 \times R_2$ such that $A[i, i + d - 1] = B[j, j + d - 1]$;
(ii) the number of pairs $(i, j) \in R_1 \times R_2$ such that $A[i] = B[j]$ is at most $\lceil \alpha(r^2/n + 1) \rceil$.

The following lemma is easy to show.

▶ **Lemma 11.** *If the two non-repetitive strings A and B have a common substring of length d, then the fraction of marked states is $\Omega\left(r^2/n^2\right)$.*

Proof. The fraction of states (R_1, R_2) for which there exists a pair $(i, j) \in R_1 \times R_2$ such that $A[i, i + d - 1] = B[j, j + d - 1]$ is at least

$$\frac{\binom{n-2}{r-2}}{\binom{n}{r}} = \frac{r(r-1)}{n(n-1)}.$$

The fraction of (R_1, R_2) such that Condition (i) holds is thus $\Omega(r^2/n^2)$.

Let X be the random variable representing the number of pairs $(i, j) \in R_1 \times R_2$ such that $A[i] = B[j]$ when (R_1, R_2) is taken uniformly at random in $\mathcal{T}_r$. We will show that $X \leq \lceil \alpha(r^2/n + 1) \rceil$ with high probability. Let us prove this upper bound for the worst possible case: input strings for which the number of pairs $(i, j) \in S \times S$ such that $A[i] = B[j]$ is $|S|$.

Assume that we have fixed R_1. For each $i \in R_1$ there exists a unique index $j \in S$ such that $A[i] = B[j]$. Let Y be the random variable representing the total number of pairs $(i, j) \in R_1 \times R_2$ such that $A[i] = B[j]$ when choosing R_2. Note that Y follows an

[6] The checking operation can actually be implemented more efficiently but the cost $c(r) = \tilde{O}(\sqrt{r})$ is enough for our purpose.

hypergeometric distribution of mean r^2/n. From standard extensions of Chernoff's bound (see, e.g., Section 1.6 of [23]), we get

$$\Pr\left[Y \geq \alpha\left(\frac{r^2}{n}+1\right)\right] \leq \exp\left(-(3\log n)\cdot\left(\frac{r^2}{n}+1\right)\right) \leq \frac{1}{n^3}.$$

Thus the fraction of (R_1, R_2) such that Condition (ii) does not hold is at most $1/n^3$.

The statement of the lemma then follows from the union bound. ◀

Let us now analyze the costs corresponding to this quantum walk. The setup and update operations are similar to the corresponding operations of the quantum walk for element distinctness described in Section 4.1. The only difference is that we need to keep track of the pairs $(i,j) \in R_1 \times R_2$ such that $A[i] = B[j]$. Note that for each $i \in [n]$, there is at most one index $j \in R_2$ such that $A[i] = B[j]$, since the strings are non-repetitive. Moreover this index can be found in polylogarithmic time using the data structure associated with R_2. Similarly, for each $j \in [n]$, there is at most one index $i \in R_1$ such that $A[i] = B[j]$, which can be also found in polylogarithmic time. Thus the time complexities of the setup and update operations are $s(r) = \tilde{O}(r)$ and $u(r) = \tilde{O}(1)$, respectively, as in the quantum walk for element distinctness described in Section 4.1. The checking operation first checks if the number of pairs $(i,j) \in R_1 \times R_2$ such that $A[i] = B[j]$ is at most $\lceil \alpha(r^2/n+1)\rceil$ and then, if this condition is satisfied, performs a Grover search on all pairs $(i,j) \in R_1 \times R_2$ such that $A[i] = B[j]$ stored in $D(R_1, R_2)$, in order to check if there exists a pair $(i,j) \in R_1 \times R_2$ such that $A[i, i+d-1] = B[j, j+d-1]$. The time complexity is

$$c(r) = \tilde{O}(1 + \sqrt{\lceil \alpha(r^2/n+1)\rceil}\cdot\sqrt{d}).$$

Using Proposition 10, we get that the overall time complexity of the quantum walk is

$$\tilde{O}\left(r + \frac{n}{r}(\sqrt{r}\times 1 + \sqrt{\lceil \alpha(r^2/n+1)\rceil}\cdot\sqrt{d})\right) = \tilde{O}\left(r + \frac{n}{\sqrt{r}} + \frac{n\sqrt{d}}{r} + \sqrt{nd}\right).$$

For $d \leq n^{1/3}$, this expression is minimized for $r = n^{2/3}$ and gives complexity $\tilde{O}(n^{2/3})$. For $d \geq n^{1/3}$, the complexity is dominated by the last term $\tilde{O}(\sqrt{nd})$.

By combining the above algorithm with the $\tilde{O}(n/\sqrt{d})$-time algorithm of Lemma 8, we get overall complexity $\tilde{O}(n^{3/4})$.

▶ **Theorem 4** (repeated). *The longest common substring of two non-repetitive strings of size n can be computed with high probability by a quantum algorithm in time $\tilde{O}(n^{3/4})$.*

Proof. We do a binary search on d. To verify a given d, we consider two cases: if $d < \sqrt{n}$ we run the $\tilde{O}(\sqrt{nd})$-time quantum algorithm we just described and otherwise we run the algorithm of Lemma 8. Thus, the overall runtime is bounded by $\tilde{O}(n^{3/4})$. ◀

4.3 Quantum algorithm for approximate LCS of non-repetitive strings

We set again $S = \{1, \ldots, n-d+1\}$ and, for any $(R_1, R_2) \in \mathcal{T}_r$, we define the data structure $D(R_1, R_2)$ exactly as above. This time, we say that (R_1, R_2) is marked if the following two conditions hold:

- **(i)** there exists a pair $(i,j) \in R_1 \times R_2$ such that $A[i, i+\lceil(1-\epsilon)d\rceil-1] = B[j, j+\lceil(1-\epsilon)d\rceil-1]$;
- **(ii)** the number of pairs $(i,j) \in R_1 \times R_2$ such that $A[i] = B[j]$ is at most $\lceil \alpha r^2/n\rceil$;

where we use the same value $\alpha = 54\log n$ as above. We now show the following lemma.

▶ **Lemma 12.** *Assume that $r \leq n/d$. If A and B have a common subsequence of length d, then the fraction of marked sets is $\Omega\left(\frac{dr^2}{n^2}\right)$.*

Proof. Let us write $m = d - \lceil(1-\epsilon)d\rceil$ and say that a pair $(i,j) \in S \times S$ is good if $A[i, i + \lceil(1-\epsilon)d\rceil - 1] = B[j, j + \lceil(1-\epsilon)d\rceil - 1]$. There are at least m good pairs in $S \times S$. The probability that R_1 does not contain any element involved in a good pair is

$$\begin{aligned}\frac{\binom{n-m}{r}}{\binom{n}{r}} &= \frac{(n-m)!r!(n-r)!}{r!(n-m-r)!n!} \\ &= \frac{(n-m)(n-m-1)\cdots(n-m-r+1)}{n(n-1)\cdots(n-r+1)} \\ &\leq \left(\frac{n-m}{n}\right)^r = \left(1-\frac{m}{n}\right)^r \leq \exp\left(-\frac{rm}{n}\right) \\ &\leq 1 - \frac{rm}{2n},\end{aligned}$$

where we used the fact that $\exp(-x) \leq 1 - x/2$ on $x \in [0,1]$ to derive the last inequality. Thus with probability at least $rm/(2n)$, the set R_1 contains at least one element involved in a good pair.

Assuming that R_1 contains at least one element involved in a good pair, when choosing R_2 at random with probability at least r/n it contains a good pair. Thus the overall probability that $R_1 \times R_2$ contains a good pair (and thus satisfies Condition (i)) is at least $\Omega(dr^2/n^2)$.

The fraction of (R_1, R_2) such that Condition (ii) does not hold is at most $1/n^3$ (the proof of this claim is exactly the same as for Lemma 11). The statement of the lemma then follows from the union bound. ◀

We thus get a quantum walk with running time

$$\tilde{O}\left(r + \frac{n}{r\sqrt{d}}(\sqrt{r} \times 1 + \sqrt{\lceil\alpha(r^2/n+1)\rceil} \cdot \sqrt{d})\right) = \tilde{O}\left(r + \frac{n}{\sqrt{rd}} + \frac{n}{r} + \sqrt{n}\right).$$

If $d \leq \sqrt{n}$ then the expression is minimized for $r = n^{2/3}/d^{1/3}$ and the complexity is $\tilde{O}(n^{2/3}/d^{1/3})$. (Note that with this value of r the condition of Lemma 12 is satisfied.) If $d > \sqrt{n}$ we obtain complexity $\tilde{O}(d)$ by taking $r = n/d$.[7]

By combining the above algorithm with the $\tilde{O}(n/\sqrt{d})$-time algorithm of Lemma 8, we get overall complexity $\tilde{O}(n^{2/3})$.

▶ **Theorem 5** (repeated). *For any constant $0 < \epsilon < 1$, the longest common substring of two non-repetitive strings of size n can be approximated within a factor $1-\epsilon$ with high probability by a quantum algorithm in time $\tilde{O}(n^{2/3})$.*

Proof. We do a binary search on d. To verify a given d, we consider two cases: if $d < n^{2/3}$ we run the $\tilde{O}(n^{2/3}/d^{1/3} + d)$-time quantum algorithm we just described and otherwise we run the algorithm of Lemma 8. Thus, the overall runtime is bounded by $\tilde{O}(n^{2/3})$. ◀

[7] For $d > \sqrt{n}$, we can actually obtain the better upper bound $\tilde{O}(\sqrt{n})$ by slightly modifying the algorithm. The bound $\tilde{O}(d)$ is nevertheless sufficient for our purpose.

References

1 Scott Aaronson, Nai-Hui Chia, Han-Hsuan Lin, Chunhao Wang, and Ruizhe Zhang. On the quantum complexity of closest pair and related problems. In *Proceedings of the 35th Computational Complexity Conference*, pages 16:1–16:43, 2020. doi:10.4230/LIPIcs.CCC.2020.16.

2 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proceedings of the 56th Annual IEEE Symposium on the Foundations of Computer Science*, pages 59–78, 2015.

3 Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007.

4 Andris Ambainis, Kaspars Balodis, Jānis Iraids, Martins Kokainis, Krišjānis Prūsis, and Jevgēnijs Vihrovs. Quantum speedups for exponential-time dynamic programming algorithms. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1783–1793, 2019.

5 Andris Ambainis and Ashley Montanaro. Quantum algorithms for search with wildcards and combinatorial group testing. *Quantum Information and Computation*, 14(5-6):439–453, 2014. doi:10.26421/QIC14.5-6-4.

6 Amihood Amir, Panagiotis Charalampopoulos, Solon P Pissis, and Jakub Radoszewski. Longest common substring made fully dynamic. In *Proceedings of the 27th Annual European Symposium on Algorithms*, pages 6:1–6:17, 2018.

7 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the 51st Annual IEEE Symposium on the Foundations of Computer Science*, pages 377–386, 2010.

8 Alexandr Andoni and Huy L. Nguyen. Near-optimal sublinear time algorithms for Ulam distance. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 76–86, 2010. doi:10.1137/1.9781611973075.8.

9 Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science*, 2020.

10 Avinatan Hassidim Aram W. Harrow and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103:150502, 2006.

11 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM Journal on Computing*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.

12 Ziv Bar-Yossef, TS Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of the 45th Annual IEEE Symposium on the Foundations of Computer Science*, pages 550–559, 2004.

13 Tuğkan Batu, Funda Ergun, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 792–801, 2006.

14 Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi HajiAghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and MapReduce. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1170–1189, 2018.

15 Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 685–698, 2020.

16 Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

17 Harry Buhrman, Subhasree Patro, and Florian Speelman. A framework of quantum strong exponential-time hypotheses. In *Proceedings of the 38th International Symposium on Theoretical*

Aspects of Computer Science, volume 187 of *LIPIcs*, pages 19:1–19:19, 2021. doi:10.4230/LIPIcs.STACS.2021.19.

18 Harry Buhrman and Robert Špalek. Quantum verification of matrix products. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 880–889, 2006.

19 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucky, and Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *Proceedings of the 59th Annual IEEE Symposium on the Foundations of Computer Science*, pages 979–990, 2018.

20 Moses Charikar and Robert Krauthgamer. Embedding the Ulam metric into ℓ_1. *Theory of Computing*, 2(11):207–224, 2006.

21 Richard Cleve, Kazuo Iwama, François Le Gall, Harumichi Nishimura, Seiichiro Tani, Junichi Teruyama, and Shigeru Yamashita. Reconstructing strings from substrings with quantum queries. In *Proceedings of the 13th Scandinavian Symposium and Workshops on Algorithm Theory*, pages 388–397, 2012. doi:10.1007/978-3-642-31155-0_34.

22 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

23 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

24 Martin Farach. Optimal suffix tree construction with large alphabets. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 137–143, 1997.

25 Michael L Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975.

26 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, pages 212–219, 1996. doi:10.1145/237814.237866.

27 Phillip Kaye, Raymond Laflamme, Michele Mosca, et al. *An introduction to quantum computing*. Oxford university press, 2007.

28 Tomasz Kociumaka, Tatiana Starikovskaya, and Hjalte Wedel Vildhøj. Sublinear space algorithms for the longest common substring problem. In *Proceedings of the 22th Annual European Symposium on Algorithms*, pages 605–617, 2014.

29 Michal Koucký and Michael Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 699–712, 2020.

30 Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011. doi:10.1137/090745854.

31 Frédéric Magniez, Miklos Santha, and Mario Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2):413–424, 2007.

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

33 Michael Mitzenmacher and Saeed Seddighin. Dynamic algorithms for LIS and distance to monotonicity. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 671–684, 2020. doi:10.1145/3357713.3384240.

34 Timothy Naumovitz, Michael E. Saks, and C. Seshadhri. Accurate and nearly optimal sublinear approximations to Ulam distance. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2012–2031, 2017. Revised version available at https://users.soe.ucsc.edu/~sesh/publication.html. doi:10.1137/1.9781611974782.131.

35 Prakash Ramanan. Tight $\Omega(n \lg n)$ lower bound for finding a longest increasing subsequence. *International journal of computer mathematics*, 65(3-4):161–164, 1997.

36 Hariharan Ramesh and V Vinay. String matching in $\tilde{O}(\sqrt{n} + \sqrt{m})$ quantum time. *Journal of Discrete Algorithms*, 1(1):103–110, 2003.

37 Aviad Rubinstein. Quantum DNA sequencing and the ultimate hardness hypothesis. `https://theorydish.blog/2019/12/09/quantum-dna-sequencing-the-ultimate-hardness-hypothesis/`, 2019.

38 Michael E. Saks and C. Seshadhri. Estimating the longest increasing sequence in polylogarithmic time. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science*, pages 458–467, 2010. `doi:10.1109/FOCS.2010.51`.

39 Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. `doi:10.1137/S0097539795293172`.

40 Mario Szegedy. Quantum speed-up of markov chain based algorithms. In *Proceedings of the 45th Annual IEEE symposium on foundations of computer science*, pages 32–41, 2004.