# Quantum Circuit Based Longest Common Substring

Domenico Cantone[1,†], Simone Faro[1,†], Arianna Pavone[2,‡] and Caterina Viola[1,*,†]

[1]Università di Catania, viale A. Doria n.6, 95125, Catania, Italy
[3]Università di Palermo, via Archirafi n.34, 90123, Palermo, Italy

**Abstract**

The Longest Common Substring (LCS) poses classical challenges in computer science, pivotal for string processing. Classically, the problem is tackled with linear time algorithms leveraging suffix trees. Recent breakthroughs in the quantum domain have unveiled sublinear solutions for LCS, demanding $\tilde{\mathcal{O}}(n^{2/3})$ quantum queries. Yet, these strides are tailored for the quantum query model, which treats input as a black box accessible via an oracle. In contrast, in this paper we delve into these challenges within the circuit model of computation. Here, circuit size gauges structural complexity, while depth identifies execution time on a quantum platform. As the query model complexity sets a baseline, any direct quantum circuit implementation yields a depth and size of at least $\tilde{\Omega}(n^{2/3})$ for LCS. The main result of this paper is the introduction of a quantum algorithm for LCS in the circuit model, which, despite its $\tilde{\mathcal{O}}(n^{3/2})$ size, achieves a groundbreaking $\tilde{\mathcal{O}}(\sqrt{n})$ depth, surpassing prior solutions. Notably, our algorithm is streamlined and readily translatable into quantum protocols. Furthermore, we demonstrate its practicality through a quantum circuit implementation operating in $\mathcal{O}(\sqrt{n}\log^5(n))$ time-steps.

**Keywords**

Quantum Computing, Text Processing, Sequence Analysis

## 1. Introduction

Quantum computing is a rapidly developing field within computer science that utilizes the principles of quantum mechanics to create more powerful computing systems operating in a markedly different way from classical computers. Unlike classical computers, which rely on bits (either 0 or 1) to process information, quantum computing leverages qubits, which can exist in multiple states simultaneously. Additionally, quantum entanglement, a physical phenomenon that allows two or more qubits to perform operations simultaneously, can be used to combine multiple qubits to perform faster and more efficient operations than classical bits. These unique features give quantum computers an advantage over classical ones, particularly in areas such as code-breaking and optimization, allowing them to perform certain calculations at an exceptional speed.

Quantum computing has had a significant impact on the development of algorithms, with some of the most notable advancements being Shor's algorithm [1] for factoring large numbers and Grover's algorithm [2] for unstructured search. These algorithms provide exponential and quadratic speed-ups over classical algorithms, respectively, serving as impressive demonstrations of the power of quantum

*ICTCS'24: Italian Conference on Theoretical Computer Science, September 11–13, 2024, Torino, Italy*
*Corresponding author.
†Supported by National Centre for HPC, Big Data and Quantum Computing, Project CN00000013, affiliated to Spoke 10, co-founded by the European Union - NextGenerationEU.
‡Supported by PNRR project ITSERR - Italian Strengthening of the ESFRI RI RESILIENCE
domenico.cantone@unict.it (D. Cantone); faro@dmi.unict.it (S. Faro); ariannamaria.pavone@unipa.it (A. Pavone); caterina.viola@unict.it (C. Viola)
0000-0002-1306-1166 (D. Cantone); 0000-0001-5937-5796 (S. Faro); 0000-0002-8840-6157 (A. Pavone); 0000-0001-7042-3912 (C. Viola)

computing and sparking a surge of interest in further research and development in the field. However, it is the recent demonstration of quantum supremacy that has unleashed a wave of interest in quantum computing, leading to the integration of these new technologies in various areas of computer science.

Only recently, text processing and string problems have become a topic of interest within the realm of quantum computation (see for instance [3, 4, 5]). This paper focuses on the fundamental Longest Common Substring (LCS) problem, which holds a crucial position in the field of string processing. The LCS problem asks for the longest substring that appears in two given input strings $x$ and $y$ of the same length $n$.

In the realm of classical computation, the LCS problem admits a linear time solution [6]. The solution to this problem involves the construction of the generalized suffix trees [7] for the input strings and the identification of the lowest common ancestors among the tree nodes.[1] It is reasonable to wonder about harnessing quantum technology to solve LCS more efficiently. In a recent paper [9], Le Gall and Seddighin proposed quantum solutions for the LCS problem based on a composition of other known quantum algorithms, such as Grover'search [2], string matching [3], element distinctness [10], and amplitude amplification and estimation [11]. Specifically, for the LCS they proposed a solution requiring $\tilde{\mathcal{O}}(n^{5/6})$ queries.

More interestingly, in [9], the authors proved that any quantum algorithm for LCS must take at least $\tilde{\Omega}(n^{2/3})$ time, even when binary strings are considered. After a while, Akmal and Jin reached in [12] the lower bound stated by Le Gall and Seddighin with a quantum algorithm in $\tilde{\mathcal{O}}(n^{2/3})$ time, improving the previous result, using the MNRS quantum walk framework [13], together with a careful combination of string synchronizing sets [14] and generalized difference covers [15].

The efficiency of previous solutions [9, 12] is measured in the *query complexity* model [3, 16], also known as the *quantum oracle model* (see [17]), where the input is presented as a *black box* that can be accessed by an *oracle* that, given a function $f$, returns the image of the input (or other variables depending on it) via $f$. The query complexity of an algorithm expressed in this model is defined as the number of *queries* that the algorithm makes to the oracle(s). However, while the query model presents an intriguing and abstract framework valuable for purely theoretical exploration, its practical relevance may be limited in the context of algorithm design for real hardware implementation. This limitation arises from the challenge of efficiently implementing an oracle, as the methodology for doing so is frequently unclear. Alternatively, there are different models of quantum computation that easily and almost directly translate to concrete implementations on quantum computers.[2]

The complexity of a quantum algorithm is *de facto* best expressed in the computational complexity model [18], where the input is encoded as a binary string and supplied to the algorithm, which computes an output string. In such a model, an algorithm is expressed using the quantum Turing machine model [19] or the quantum circuit model [20]. Perhaps, this latter is one of the most widespread among such models, considering that there are several programming languages featuring the circuit formalism, such as IBM's `Qiskit`, Microsoft's `Q#`, and Google's `Cirq`, just to cite a few. The computational complexity of a quantum circuit can be measured by its *size* (the number of gates) or by its *depth* (the number of layers). We refer to Section 2.1 for details on the computational model based on quantum circuits and a discussion about the measures used to calculate its complexity.

Thus, given that any quantum oracle comprises at least one gate, and gates are applied sequentially in an algorithm adhering to the query complexity model, the query complexity of solutions given

[1] We also mention an algorithm [8] working in the word RAM model of computation when the size $\sigma$ of the input alphabet is in $2^{o(\sqrt{\log(n+m)})}$. Such solution runs in $\mathcal{O}\big((n+m)\log\sigma/\sqrt{\log(n+m)}\big)$ time using $\mathcal{O}\big((n+m)\log\sigma/\log(n+m)\big)$ space.

[2] We observe that contemporary quantum computers do not have yet the memory capabilities to deal with the large number of qubits involved in our algorithm.

| Problem | Paper | Query Compl. | Circuit Size | Circuit Depth |
|---|---|---|---|---|
| exact LCS | [9] | $\tilde{\mathcal{O}}(n^{5/6})$ | $\tilde{\Omega}(n^{5/6})$ | $\tilde{\Omega}(n^{5/6})$ |
| exact LCS | [12] | $\tilde{\mathcal{O}}(n^{2/3})$ | $\tilde{\Omega}(n^{2/3})$ | $\tilde{\Omega}(n^{2/3})$ |
| exact LCS | This paper | - | $\tilde{\mathcal{O}}(n^{3/2})$ | $\tilde{\mathcal{O}}(\sqrt{n})$ |

**Table 1**
A comparison of quantum solutions on LCS. The time complexities for [9] are unknown due to their reliance on random-access oracles, which lack a circuit-level construction in the referenced paper.

in [9] and [12] establishes a lower bound for both the size and the depth of any quantum circuit implementing the same algorithms. Consequently, the optimal hypothetical circuit-based solution would necessitate a depth (and a size) of $\tilde{\Omega}(n^{2/3})$ (see Table 1).

In this paper we present the first quantum algorithm for the LCS problem in the circuit model of computation, providing an actual implementation of a quantum circuit that works in $\tilde{\mathcal{O}}(\sqrt{n})$ depth, despite its $\tilde{\mathcal{O}}(n^{3/2})$ size. Specifically, our proposed approach leads to the definition of an effective circuit that requires $\mathcal{O}(\sqrt{n}\log^4(n))$ depth in the case of binary strings, and $\mathcal{O}(\sqrt{n}\log^5(n))$ depth in the general case.

At first glance, our result might seem contradictory to that of Le Gall and Seddighin; however, in fact, the comparison of the two results shows how in quantum computation space efficiency can be traded off for time efficiency. While Le Gall and Seddighin access the input by querying a quantum oracle, we have direct access to the input, which is encoded on a circuit register. Instead of claiming the preferability of one model over the other, we aim to draw the reader's attention to the differences between them. The query model allows one to study and analyse quantum algorithms without worrying about the technicalities around the construction of any specific oracle, and has been the framework of the first outstanding attempts to design algorithms that exhibit a theoretical advantage against classical ones.

Another significant difference lies in the ways in which the two algorithms access any input string $s$. In [9], the authors assume a QRAM (Quantum Random Access Memory) model [21]. Specifically, it is assumed that the string $s$ can be accessed directly by a random access oracle that performs, at unit cost, unitary mappings of the kind $|i\rangle|a\rangle|z\rangle \rightarrow |i\rangle|a \otimes s[i]\rangle|z\rangle$, where $i$ is a string position such that $0 \leqslant i < n$, $a \in \Sigma$ is a character, $z \in \{0,1\}^*$, and $\otimes$ denotes an appropriate binary operation defined on $\Sigma$. However, we point out that the most efficient QRAM designs [22, 23] exhibit a polylogarithmic time complexity for accessing the memory with respect to its size. In our scenario, the memory size is $\mathcal{O}(n)$, which implies that QRAM queries will incur an additional multiplicative cost of at least $\mathcal{O}(\log^2(n))$. Moreover, we must consider the overhead of initializing the quantum memory, which requires $\mathcal{O}(n)$ operations [24].

In contrast, our algorithm does not rely on a random access oracle, but we assume, as in [4, 25], that the input registers are already stored in a quantum memory and do not need initialization.

Ultimately, our approach stands apart from the previous results due to its inherent simplicity, which enables us not only to provide a circuit-level blueprint, but also to assess the quantum resources required for its implementation.

The paper is organized as follows. In Section 2, we review some useful preliminaries. Next, in Section 3, we provide an abstract view of the algorithm for solving the LCS problem. Then, in Section 4, we describe an actual implementation of the same algorithm within the circuit-based model. Finally, in Section 5, we briefly draw our conclusions.

## 2. Preliminaries

We represent a string $x$ of length $n \geqslant 1$, over a finite alphabet $\Sigma$ of size $\sigma$, as a finite array $x[0\,..\,n-1]$, and denote the empty string by $\varepsilon$. We also denote by $x[i]$ the $(i+1)$-st character of $x$, for $0 \leqslant i < n$, and by $x[i\,..\,j]$ the substring of $x$ contained between the $(i+1)$-st and the $(j+1)$-st characters of $x$, for $0 \leqslant i \leqslant j < n$. A $k$-substring of a string $x$ is any substring of $x$ of length $k$. For ease of notation, the $(i+1)$-st character of the string $x$ will also be denoted by the symbol $x_i$, so that $x = x_0x_1 \ldots x_{n-1}$. A substring of $x$ beginning at position 0 is a *prefix* of $x$. We use the notation $x_{:i}$ to indicate the prefix of $x$ of length $i$.

For any two strings $x$ and $y$ of length $n$, we say that $x$ and $y$ have a common $k$-substring if there exist two indices $0 \leqslant i, j < n-k$ such that $x[i\,..\,i+k-1] = y[j\,..\,j+k-1]$. In particular, when the indices $i$ and $j$ coincide, we say that $x$ and $y$ *share* a $k$-substring at position $i$. The expression $x \cdot y$ denotes the concatenation of $x$ and $y$. Furthermore, given a string $x$ of length $n$ and a shift $0 \leqslant j < n$, we denote by $\overrightarrow{x}^j$ the cyclic rightward rotation of the characters of $x$ by $j$ positions. More formally, we have $\overrightarrow{x}^j \coloneqq x[n-j\,..\,n-1] \cdot x[0\,..\,n-j-1]$.

Due to space limitations, we assume the reader is familiar with essential concepts in quantum computation, including qubits, bra-ket notation, amplitudes, quantum entanglement, and measurement. Multiple qubits taken together are referred to as *quantum registers*. Specifically, a quantum register $|\psi\rangle = |q_0, q_1, \ldots, q_{n-1}\rangle$ of $n$ qubits is the tensor product $\bigotimes_{i=0}^{n-1} |q_i\rangle$ of its constituent qubits. If $k$ is an integer value that can be represented as a binary string of length $n$, we use the symbol $|k\rangle$ to denote the register $\bigotimes_{i=0}^{n-1} |k_i\rangle$ of $n$ qubits, where $|k_i\rangle$ takes the value of the $i$-th most significant binary digit of $k$. Thus, the quantum register $|8\rangle$ with 4 qubits is given by $|8\rangle = |1000\rangle$.

*Operators* in quantum computing are mathematical entities used to represent functional processes that result in the change of the state of a quantum register. Although there is no problem in realizing any quantum operator capable of working in constant time on a quantum register of fixed size, operators of variable size can only be implemented through the composition of elementary gates.

Given a function $f\colon \{0,1\}^n \to \{0,1\}$, any quantum operator that maps a register containing the value of a given input $x \in \{0,1\}^n$ into a register whose value depends on $f(x)$ is called a *quantum oracle*. A *Boolean oracle* $U_f$ maps a register $|x\rangle \otimes |0\rangle$, of size $n+1$, to the register $|x\rangle \otimes |f(x)\rangle$. More formally, $U_f|x, 0\rangle = |x, f(x)\rangle$. A *phase oracle* $P_f$ for a function $f\colon \{0,1\}^n \to \{0,1\}$ takes as input a quantum register $|x\rangle$, where $x \in \{0,1\}^n$, and leaves its value unchanged, while applying to it a negative global phase only when $f(x) = 1$, that is, only if $x$ is a solution for the function. More formally, $P_f|x\rangle = (-1)^{f(x)}|x\rangle$. Intuitively, a Boolean oracle is a black-box function that outputs a binary result (0 or 1) based on the input. A phase oracle, instead of giving a classical output, alters the phase of the quantum state if a certain condition is met, flipping its sign. Thus, while the Boolean oracle returns a bit, the phase oracle encodes the result directly into the quantum state's phase, important for algorithms like Grover's search.

### 2.1. The Quantum Circuit Model and Its Complexity Measures

In this paper we adopt the *circuit model of computation* [18]. David Deutsch was the first to formulate the idea of the quantum circuit model [26] to encapsulate quantum computations, although, in his original formalization, Deutsch uses the term *quantum network*.

In fact, quantum circuits are networks composed of wires that carry qubit values to *gates* that perform elementary operations on qubits. The qubits move through the circuit in a linear fashion, where the input values are written onto the wires entering the circuit from the left side, while the output values are read off the wires leaving the circuit on the right side. At every time step, each

wire can enter at most one gate.

Formally, the quantum circuit model constitutes a broader framework than the classical circuit model. As in a classical circuit, the *size*, or number of gates involved in a quantum circuit, is a measure of its computational complexity, since it represents the number of elementary operations required to execute a given quantum algorithm: the more gates or operators are applied in a circuit, the more complex the operation the circuit is performing. This measure becomes even more meaningful within the quantum framework due to the inherent susceptibility of modern quantum computers to gate errors. As the number of gates increases, the reliability of the final outcome diminishes, necessitating the implementation of error correction methods.

On the other hand, it is imperative to acknowledge that, unlike classical computation, where only one gate can be executed at a time regardless of circuit structure, a quantum computer enjoys the remarkable advantage of concurrently executing two (or more) gates, provided they do not involve the same set of qubits [27, 28]. This characteristic of quantum computation, coupled with superposition and entanglement, underpins quantum supremacy.[3]

Hence, size does not consistently represent the most precise measure of complexity in quantum computation, often providing only a rough approximation of an algorithm's intricacy. The complexity of a quantum algorithm depends on various factors, including the types of gates used, qubit connectivity within the quantum processor, and especially the circuit depth — defined as the number of layers required for parallel execution, where a qubit participates in at most one interaction per layer [29]. It is important to note that the depth of a circuit does not necessarily correspond to its size, as gates acting on disjoint sets of qubits can often be applied in parallel.

As quantum gates necessitate implementation time, the depth of a circuit in modern quantum computers approximately correlates with the duration required for the quantum computer to execute the circuit. Consequently, circuit depth serves as a crucial metric to assess the feasibility of running a quantum circuit on a device.[4]

In addition, enabling the realization of quantum algorithms in the near future with existing technology appears contingent upon the development of shallow-depth quantum circuits [29]. Qubits are susceptible to decoherence, rendering them prone to spontaneous state fluctuations, thereby limiting the duration of feasible operations. Maximizing the utilization of these delicate qubits necessitates the circuit depth reduction [27] and, therefore, the parallelization of circuits.[5]

In recent years, the complexity of quantum states has emerged as a pivotal quantity of interest spanning various domains, ranging from quantum computing [30] to black hole theory [31]. Reflecting this burgeoning interest, Haferkamp *et al.* [32] recently validated the Brown and Susskind conjecture [33], asserting that the complexity of quantum circuits typically experiences linear growth with circuit depth over an exponentially protracted period, ultimately reaching saturation when the number of applied gates surpasses a threshold that scales exponentially with the number of qubits.

For the reasons stated above, in this paper, our objective is to provide a solution to the LCS problem, where the measure of complexity is more directed towards circuit depth rather than size.

[3] We observe that even in the absence of quantum entanglement, simultaneous operations on all qubits remain feasible. Utilizing $n$ qubits permits $n$ concurrent operations per time step. Nevertheless, in theory, a quantum computer comprising $n$ qubits could emulate the functionality of a classical computer outfitted with $n$ processors.

[4] The depth of a circuit is considered the measure of complexity in many quantum languages. See for example https://docs.quantum.ibm.com/api/qiskit/0.43/circuit

[5] It is noteworthy that achieving parallelism within the quantum circuit model mandates the capability to interact with spatially distant qubits. Various implementations may impose physical constraints on the extent of such interaction. Nonetheless, recent advancements in quantum computing [1–7] have demonstrated successful realization of long-range qubit interactions in several proposed schemes.

```
QUANTUM-LCS(x, y, n):
1.  ℓ ← 0; r ← n
2.  while ℓ < r do
3.      d ← ⌊(ℓ + r)/2⌋
4.      if [∃ i, j ∈ {0, ..., n − 1} : x⃗^j[i .. i + d − 1] = y[j .. j + d − 1]]   ← QUANTUM TEST
5.          then ℓ ← d
6.          else r ← d − 1
7.  return ℓ
```

**Figure 1:** The pseudocode of the algorithm for computing the LCS between two strings, $x$ and $y$, of length $n$. The quantum part of the algorithm reduces to the iterative test of line 4.

## 3. Quantum Longest Common Substring in the Circuit Model

In this section, we first describe our quantum algorithm for computing the LCS within an abstract model, in order to better understand its design, by defining the quantum oracles involved in the computation, but without giving their actual implementation. Later, we will show that our abstract algorithm requires $\tilde{\mathcal{O}}(\sqrt{n})$ queries to oracles.[6] Subsequently, we will present an actual implementation of our algorithm in the circuit-based computational model. Our algorithm, named QUANTUM-LCS, comprises a quantum computation-based and a classical computation-based components. Its simple underlying structure is summarized in the pseudocode shown in Figure 1.

The classical part of the computation involves a binary search to determine the length $d$ of the longest common substring of $x$ and $y$ (line 2). During each iteration, the algorithm checks for a common substring of length $d$ between $x$ and $y$. Let $[\ell \,..\, r]$ be the interval over which the binary search is restricted during an iteration of the algorithm, and let $d = \lfloor (\ell + r)/2 \rfloor$ be its median. The values of $\ell$ and $r$ are initialized to $0$ and $n$, respectively. If the iterative test returns a positive answer, then the interval is narrowed to $[d \,..\, r]$, otherwise it is narrowed to $[\ell \,..\, d-1]$. The search identifies the length $d$ of the longest common substring in $\mathcal{O}(\log(n))$ steps.

The quantum part of the algorithm implements the test of line $4$. In what follows, we will focus exclusively on the implementation of such iterative test. Before describing the details of the quantum procedure for the iterative test, we formalize some assumptions we make along the description.

Since a quantum register of dimension $\log(n)$ can take on all values between $0$ and $n-1$, like any binary sequence of the same dimension, for simplicity we will assume that both input strings $x$ and $y$ have length $n = 2^p$, for some $p > 0$. We also assume that $x$ and $y$ end with two different special characters, \$ and %, respectively, not belonging to the alphabet $\Sigma$. These assumptions can be made without any loss of generality, since it would suffice to take the smallest value $p$ for which we have $n < 2^p$ and concatenate the text with $2^p - n$ copies of the special character. For instance, if $x = \texttt{abaacbcbbca}$ is a text of length $11$, we silently concatenate it with $5$ copies of the character \$, i.e., we assume that $x = \texttt{abaacbcbbca\$\$\$\$\$}$. This assumption does not affect on the asymptotic complexity, as the resulting string is at most twice as long as the original.

Despite any substring of length $d$ can begin at any position $j$ of the text, for $0 \leqslant j \leqslant n - d$, in this paper we also admit values of $j$ between $0$ and $n-1$, thus assuming that a substring of the text can be obtained in a circular way. Even such an assumption can be made without loss of generality, since the last character of $x$ and $y$ are the special character \$ and %, respectively, and therefore no substring obtained circularly can ever be returned as LCS.

[6] We would like to point out that in counting the number of queries requested by our algorithm, we do not intend to compute its query complexity, since we work within the circuit-based computational approach that does not conform to the constraints of the query-based model.

For the sake of simplicity and due to space constraints, we restrict to circuits algorithms designed for processing binary strings. This further simplification, however, does not lead to any substantial change in our results since, assuming that each character can be represented with (at most) $\log(n)$ bits, it is easy to show that the quantum operators used in the construction of the algorithm would undergo an increase in their complexity at most equal to a factor of $\log(n)$.

### 3.1. The Quantum Iterative Test

Given two strings $x$ and $y$, both of length $n$, and a bound $d \leqslant n$, the quantum test checks for the presence of a common substring of length $d$ between $x$ and $y$.

The abstract procedure for the iterative test is outlined in Figure 2. It consists of three phases, each implemented by a quantum sub-procedure: (1) a search phase, (2) a verification phase, and (3) a final check. The output of the iterative test is the output of the final check. In this section we describe in detail the role, structure, and complexity of each of the phases and then discuss the overall complexity of the iterative test.

The *search phase* makes use of the Grover's search algorithm [2] for finding (with high probability) a solution (if any) to a black box function, making just $\mathcal{O}(\sqrt{n})$ queries to the function. Specifically, the input black box to the algorithm is accessed by a phase oracle $P_\psi$ implementing the function $\psi_{(x,y)} \colon \{0,\ldots,n-1\} \times \{0,\ldots,n-1\} \longrightarrow \{0,1\}$, which depends on the strings $x$ and $y$. Given the two input parameters $j$ and $d$, with $0 \leqslant j, d < n$, the phase oracle $P_\psi$ tests whether the two strings $\vec{x}^j$ and $y$ share a $d$-substring. The function $\psi_{(x,y)}$ is defined, for all $0 \leqslant j, d < n$, as

$$\psi_{(x,y)}(j,d) = \begin{cases} 1 & \text{if } \exists\, i \in \{0,\ldots,n-1\} : \vec{x}^j[i\,..\,i+d-1] = y[i\,..\,i+d-1] \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

When the values of $x$ and $y$ are clear from the context, for simplicity we will use the symbol $\psi$ instead of $\psi_{(x,y)}$. Using this convention, the phase oracle $P_\psi$ operates so as to achieve the transformation $P_\psi|j\rangle = (-1)^{\psi(j)}|j\rangle$, for all $j \in \{0,1\}^{\log(n)}$. In Section 4.1, we show how the phase oracle $P_\psi$ can be effectively implemented by means of a circuit having depth $\mathcal{O}(\log^3(n))$.

After $\mathcal{O}(\sqrt{n})$ iterations of Grover's algorithm, the procedure returns a potential solution $j$ to the problem, such that $\vec{x}^j$ and $y$ share a $d$-substring. However, since such a solution may not exist (a case in which the search would return a random state $0 \leqslant j < n$), it is necessary to run the subsequent verification procedures to check whether the returned state is an actual solution of the function.

Assuming $\vec{x}^j$ and $y$ share a $d$-substring, the *verification phase* again makes use of Grover's search algorithm in order to identify a position $i$ within the strings such that $\vec{x}^j[i\,..\,i+d-1] = y[i\,..\,i+d-1]$. In this case, the input black box to the algorithm is a phase oracle $P_\varphi$ implementing the function $\varphi_{(x,y)} \colon \{0,\ldots,n-1\} \times \{0,\ldots,n-1\} \times \{0,\ldots,n-1\} \longrightarrow \{0,1\}$, where, for all strings $x$ and $y$, and for all $0 \leqslant i, j, d < n$, we have:

$$\varphi_{(x,y)}(i,j,d) = \begin{cases} 1 & \text{if } \vec{x}^j[i\,..\,i+d-1] = y[i\,..\,i+d-1] \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Therefore, the oracle $P_\varphi$ operates so as to achieve the phase transformation $P_\varphi|i\rangle|j\rangle|d\rangle = (-1)^{\varphi(i,j,d)}|i\rangle|j\rangle$, for all $i, j, d \in \{0,1\}^{\log(n)}$. In Section 4.1, we provide an implementation of the phase oracle $P_\varphi$ that operates in $\mathcal{O}(\log^3(n))$ time.

Even in this case, after $\mathcal{O}(\sqrt{n})$ iterations of Grover's algorithm, the procedure returns a potential position $i$, such that $\vec{x}^j[i\,..\,i+d-1] = y[i\,..\,i+d-1]$.

QUANTUM TEST:

1. $j \leftarrow$ get a random $k$ such that $\vec{x}^k$ and $y$ (possibly) share a $d$-substring (SEARCH PHASE)
2. $i \leftarrow$ get a random $k$ such that $\vec{x}^j[k \,..\, k+d-1]$ is (possibly) equal to $y[k \,..\, k+d-1]$ (VERIFICATION)
3. check if $\vec{x}^j[i \,..\, i+d-1]$ is equal to $y[i \,..\, i+d-1]$ (FINAL CHECK)

**Figure 2:** The structure of the quantum test used in the algorithm in Figure 1. The test comprises three phases, each of which is implemented as a quantum procedure.

Ultimately, the quantum test ends by checking whether the two substrings of length $d$ beginning at position $i$ of the strings $\vec{x}^j$ and $y$ are indeed equal. Such a *final check* can be exactly computed through a single execution of the quantum oracle $U_\varphi$ implementing the function $\varphi$ defined in (2).

The whole structure of the quantum test is depicted in Figure 2. The first two phases require both $\mathcal{O}(\sqrt{n})$ queries to the oracles $P_\psi$ and $P_\varphi$, respectively, while the last check requires a single query to the Boolean oracle $U_\varphi$. Therefore, the quantum iterative test requires $\mathcal{O}(\sqrt{n})$ queries.

We point out that, when a solution exists, both the search and verification phases may fail with a probability $\mathcal{O}(1/n)$, due to the internal randomness of Grover's algorithm. When the search phase or the verification phase returns a value that is not a solution of the respective function, the final check fails by returning the value 0. In such a case, we can simply repeat the whole test an arbitrary constant number of times in order to suppress the probability of failure. The test terminates when a common substring is found, or when such an attempt fails an arbitrary number of times.

The overall number of queries needed to solve the problem is $\mathcal{O}(\sqrt{n}\log(n))$, since the execution of the quantum iterative test requires $\mathcal{O}(\sqrt{n})$ queries and the binary search for the length of the LCS requires $\mathcal{O}(\log(n))$ iterations.

## 4. A Circuit-Model Based Implementation

In this section we provide an actual implementation of the iterative test shown in Figure 2 within the circuit-based computational model. The purpose of this translation is to provide a direct implementation of the algorithm in a quantum computer and evaluate the actual resources required.

The three steps of the iterative test are implemented by the three circuits reported in Figure 5. Only three operators are used as building-blocks in the three circuits: the circular shift (ROT) operator, the shared fixed substring checking (SFC) operator, and the fixed prefix matching (FPM) operator (see Table 2). We observe that the oracles used in the actual circuits of Fig.5 are implemented as Boolean oracles rather than as phase oracles. These are denoted as $U_\psi$ and $U_\varphi$ instead of $P_\psi$ and $P_\varphi$, respectively. However, we recall that initializing the output register of a Boolean oracle to the value $|-\rangle$ allows it to behave like a phase oracle.

For lack of space, in this section we only provide a brief overview of how these operators are structured, referring the reader to the appropriate references.

A *circular shift operator* (or rotation operator) ROT applies a rightward shift of $s$ positions to a register of $n$ qubits for a fixed parameter $0 \leqslant s < n$. Thus, the element at position $i$ is moved to position $(i+s) \mod n$. Such an operator has been effectively used in other quantum text searching algorithms [4, 5]. The details of its construction have been detailed by Pavone and Viola in [34], where it is shown that the resulting operator can be executed in $\mathcal{O}(n\log(n))$ size and $\mathcal{O}(\log(n))$ depth.

In our implementation we make use of the controlled version of the circular shift operator, which applies a circular rotation of a number of positions, depending on an input value $j$ such that $0 \leqslant j < n$.

| Operator | Symbol | Size | Depth B.S. | Depth G.C. | Ref. |
|---|---|---|---|---|---|
| Controlled Circular Shift | ROT | $\mathcal{O}(n\log(n))$ | $\mathcal{O}(\log^2(n))$ | $\mathcal{O}(\log^2(n))$ | [34] |
| Shared Fixed Substring Check | SFC | $\mathcal{O}(n\log(n))$ | $\mathcal{O}(\log^3(n))$ | $\mathcal{O}(\log^4(n))$ | [35, 25] |
| Fixed Prefix Matching | FPM | $\mathcal{O}(n\log(n))$ | $\mathcal{O}(\log^3(n))$ | $\mathcal{O}(\log^4(n))$ | [35, 25] |

**Table 2**
The three operators used in the implementation of our circuits, with an indication of their size and their depth in the case of binary strings (B.S.) and in the general case (G.C.).

More formally, for all $x \in \{0,1\}^n$ and all $j \in \{0,1\}^{\log(n)}$, the controlled circular shift operator performs the mapping $\text{ROT}|j\rangle|x\rangle = |j\rangle|\vec{x}^j\rangle$.

The controlled variant of the circular shift operator can be implemented by means of a well-known technique [5] that involves the use of $\log(n)$ ancillæ qubits for the application of all parallel operators controlled by the same qubit, with an overhead of $\mathcal{O}(\log(n))$ in both size the depth. Thus, the operator achieves $\mathcal{O}(n\log^2(n))$ size and $\mathcal{O}(\log^2(n))$ depth.

The *shared fixed substring checking* (SFC) operator addresses the following simple string matching problem, in which, given two strings $x$ and $y$, both of length $n$, and a bound $d \geqslant 0$, one wants to check whether $x$ and $y$ share a common $d$-substring, i.e., if there exists a position $i$, with $0 \leqslant i < n-d$, such that $x[i\,..\,i+d-1] = y[i\,..\,i+d-1]$. In other words, the SFC operator computes the function $\psi_{(x,y)}(j,d)$ for the special case where $j=0$, that is, $x$ does not undergo any cyclic rotation. More formally, given a bound $d \leqslant n$, the SFC operator, for all $x, y \in \{0,1\}^n$ and $d \in \{0,1\}^{\log(n)}$, is defined by $\text{SFC}|x\rangle|y\rangle|d\rangle|0\rangle = |x\rangle|y\rangle|d\rangle|\psi_{(x,y)}(0,d)\rangle$.

The construction of a quantum circuit implementing the SFC operator has been recently proposed in [35], where the authors provide a circuit with a $\mathcal{O}(\log^3(n))$ depth in the case of binary input strings and a circuit with a $\mathcal{O}(\log^4(n))$ depth in the general case. The size of the circuit is $O(n\log(n))$ in both cases. We do not provide here further details on the construction of the operator but refer to [35] for any structural aspects of the corresponding circuit.

Given two strings $x$ and $y$, both of length $n$, and a bound $d \leqslant n$, the *fixed prefix matching* (FPM) operator performs a simple check to determine whether the first $d$ characters of the string $x$ match their counterparts in the string $y$. Roughly speaking, the FPM operator checks if $x_{:d} = y_{:d}$. More formally, for all $x, y \in \{0,1\}^n$, and all $d \in \{0,1\}^{\log(n)}$, the FPM operator is defined by

$$\text{FPM}|x\rangle|y\rangle|d\rangle|0\rangle = |x\rangle|y\rangle|d\rangle|\varphi_{(x,y)}(0,0,d)\rangle.$$

The construction of a quantum circuit implementing the FPM operator has also been recently proposed in [35], where the authors give a circuit with a depth of $\mathcal{O}(\log^3(n))$ for the case of binary input strings and a circuit with a depth of $\mathcal{O}(\log^4(n))$ for the general case. The size of the corresponding circuit is $O(n\log(n))$ in both cases.

We are now ready to describe the quantum circuits that implement the three phases of the quantum iterative test. All circuits make use of two registers $|x\rangle$ and $|y\rangle$, both of size $n$, which we assume to contain the characters of the two input strings $x$ and $y$, respectively. We also assume that these registers are already stored in a quantum memory and do not need initialization. All circuits involve the presence of an input register $|d\rangle$, of size $\lceil\log(d)\rceil \leqslant \log(n)$, containing the binary representation of the bound $d \leqslant n$. The initialization of such input register requires $\mathcal{O}(\log(n))$ time. The output of the computation, for all circuits, is stored in the $|out\rangle$ register consisting of a single qubit.

| $\|j\rangle$ | $\|x\rangle$ | $\|y\rangle$ | $\|d\rangle$ | $\|r\rangle$ | $\|out\rangle$ | |
|---|---|---|---|---|---|---|
| $\|j\rangle$ | $\|x\rangle$ | $\|y\rangle$ | $\|d\rangle$ | $\|0\rangle$ | $\|0\rangle$ | ← INITIALIZATION |
| $\|j\rangle$ | $\|\vec{x}^j\rangle$ | $\|y\rangle$ | $\|d\rangle$ | $\|0\rangle$ | $\|0\rangle$ | ← APPLICATION OF ROT$\|j\rangle\|x\rangle$ |
| $\|j\rangle$ | $\|\vec{x}^j\rangle$ | $\|y\rangle$ | $\|d\rangle$ | $\|\psi_{(\vec{x}^j,y)}(0,d)\rangle$ | $\|0\rangle$ | ← APPLICATION OF SFC$\|x\rangle\|y\rangle\|d\rangle\|r\rangle$ |
| $\|j\rangle$ | $\|\vec{x}^j\rangle$ | $\|y\rangle$ | $\|d\rangle$ | $\|\psi_{(\vec{x}^j,y)}(0,d)\rangle$ | $\|\psi_{(\vec{x}^j,y)}(0,d)\rangle$ | ← APPLICATION OF CX$\|r\rangle\|out\rangle$ |
| $\|j\rangle$ | $\|\vec{x}^j\rangle$ | $\|y\rangle$ | $\|d\rangle$ | $\|0\rangle$ | $\|\psi_{(\vec{x}^j,y)}(0,d)\rangle$ | ← APPLICATION OF SFC$^\dagger\|x\rangle\|y\rangle\|d\rangle\|r\rangle$ |
| $\|j\rangle$ | $\|x\rangle$ | $\|y\rangle$ | $\|d\rangle$ | $\|0\rangle$ | $\|\psi_{(\vec{x}^j,y)}(0,d)\rangle$ | ← APPLICATION OF ROT$^\dagger\|j\rangle\|x\rangle$ |

**Figure 3:** The evolution of the six registers ($|j\rangle,\ |x\rangle,\ |y\rangle,\ |d\rangle,\ |r\rangle$, and $|out\rangle$) involved in the computation of the Boolean oracle $U_\psi$, whose circuit is depicted in Figure 5

.

| $\|i\rangle$ | $\|j\rangle$ | $\|x\rangle$ | $\|y\rangle$ | $\|d\rangle$ | $\|r\rangle$ | $\|out\rangle$ | |
|---|---|---|---|---|---|---|---|
| $\|i\rangle$ | $\|j\rangle$ | $\|x\rangle$ | $\|y\rangle$ | $\|d\rangle$ | $\|0\rangle$ | $\|0\rangle$ | INITIALIZATION |
| $\|i\rangle$ | $\|j\rangle$ | $\|\vec{x}^j\rangle$ | $\|y\rangle$ | $\|d\rangle$ | $\|0\rangle$ | $\|0\rangle$ | APPLICATION OF ROT$\|j\rangle\|x\rangle$ |
| $\|i\rangle$ | $\|j\rangle$ | $\|\vec{x}^{j+i}\rangle$ | $\|y\rangle$ | $\|d\rangle$ | $\|0\rangle$ | $\|0\rangle$ | APPLICATION OF ROT$\|i\rangle\|x\rangle$ |
| $\|i\rangle$ | $\|j\rangle$ | $\|\vec{x}^{j+i}\rangle$ | $\|\vec{y}^i\rangle$ | $\|d\rangle$ | $\|0\rangle$ | $\|0\rangle$ | APPLICATION OF ROT$\|i\rangle\|y\rangle$ |
| $\|i\rangle$ | $\|j\rangle$ | $\|\vec{x}^{j+i}\rangle$ | $\|\vec{y}^i\rangle$ | $\|d\rangle$ | $\|\varphi_{(\vec{x}^{j+i},\vec{y}^i)}(0,0,d)\rangle$ | $\|0\rangle$ | APPLICATION OF FPM$\|x\rangle\|y\rangle\|d\rangle\|r\rangle$ |
| $\|i\rangle$ | $\|j\rangle$ | $\|\vec{x}^{j+i}\rangle$ | $\|\vec{y}^i\rangle$ | $\|d\rangle$ | $\|\varphi_{(\vec{x}^{j+i},\vec{y}^i)}(0,0,d)\rangle$ | $\|\varphi_{(\vec{x}^{j+i},\vec{y}^i)}(0,0,d)\rangle$ | APPLICATION OF CX$\|r\rangle\|out\rangle$ |
| $\|i\rangle$ | $\|j\rangle$ | $\|\vec{x}^{j+i}\rangle$ | $\|\vec{y}^i\rangle$ | $\|d\rangle$ | $\|0\rangle$ | $\|\varphi_{(\vec{x}^{j+i},\vec{y}^i)}(0,0,d)\rangle$ | APPLICATION OF FPM$^\dagger\|x\rangle\|y\rangle\|d\rangle\|r\rangle$ |
| $\|i\rangle$ | $\|j\rangle$ | $\|\vec{x}^{j+i}\rangle$ | $\|y\rangle$ | $\|d\rangle$ | $\|0\rangle$ | $\|\varphi_{(\vec{x}^{j+i},\vec{y}^i)}(0,0,d)\rangle$ | APPLICATION OF ROT$^\dagger\|i\rangle\|y\rangle$ |
| $\|i\rangle$ | $\|j\rangle$ | $\|\vec{x}^j\rangle$ | $\|y\rangle$ | $\|d\rangle$ | $\|0\rangle$ | $\|\varphi_{(\vec{x}^{j+i},\vec{y}^i)}(0,0,d)\rangle$ | APPLICATION OF ROT$^\dagger\|i\rangle\|x\rangle$ |
| $\|i\rangle$ | $\|j\rangle$ | $\|x\rangle$ | $\|y\rangle$ | $\|d\rangle$ | $\|0\rangle$ | $\|\varphi_{(\vec{x}^{j+i},\vec{y}^i)}(0,0,d)\rangle$ | APPLICATION OF ROT$^\dagger\|j\rangle\|x\rangle$ |

**Figure 4:** The evolution of the seven registers ($|i\rangle,\ |j\rangle,\ |x\rangle,\ |y\rangle,\ |d\rangle,\ |r\rangle,$ and $|out\rangle$) involved in the computation of the Boolean oracle $U_\varphi$, as shown in Figure 5.

## 4.1. Implementing the Circuits for the Three Phases

The circuit for the search phase is depicted on the top of Figure 5. It makes use of the additional $|j\rangle$ register, of size $\log(n)$, which holds the rotation values of the string $x$. It is initialized to $|+\rangle^{\log(n)}$, in order to maintain, at the initial stage, the superposition of all possible rotation values between 0 and $n-1$. The $|r\rangle$ register, of a single qubit initialized to $|0\rangle$, stores the output of the SFC operator.

The core of the quantum procedure involves applying Grover's search algorithm on the phase oracle, $U_\psi$, of the $\psi_{(x,y)}$ function, as defined in (1). The Boolean oracle $U_\psi$ takes the two registers $|j\rangle$ and $|d\rangle$ as input, and is implemented through the ROT and SFC operators. The output of the SFC operator is stored in the qubit $|r\rangle$, while the output of $U_\psi$ is stored on the $|out\rangle$ register, which is initialized to $|-\rangle$ to make $U_\psi$ to behave as a phase oracle.

In Figure 3, we show the evolution of the 6 registers involved in the computation of the Boolean oracle $U_\psi$, namely $|j\rangle|x\rangle|y\rangle|d\rangle|r\rangle$ and $|out\rangle$ (see also Figure 5).

Specifically, the application on the register $|x\rangle$ of the ROT operator, controlled by the register $|j\rangle$, allows $|x\rangle$ to be modified so that it contains the superposition of all its possible cyclic rotations. Next, the application of the SFC operator on the registers $|d\rangle$, $|x\rangle$, and $|y\rangle$ allows the procedure to identify a possible position $i$ (if any) for which $\vec{x}^j[i\,..\,i+d-1] = y[i\,..\,i+d-1]$. Note that the application of the SFC operator is done in parallel, for all possible rotations of the register $|x\rangle$. The oracle completes its computation by saving the output of the SFC operator into the $|out\rangle$ register and uncomputing the entire process by applying the inverse operators in their reverse order.

Regarding the depth of the circuit for the search phase, we can observe that the ROT and the

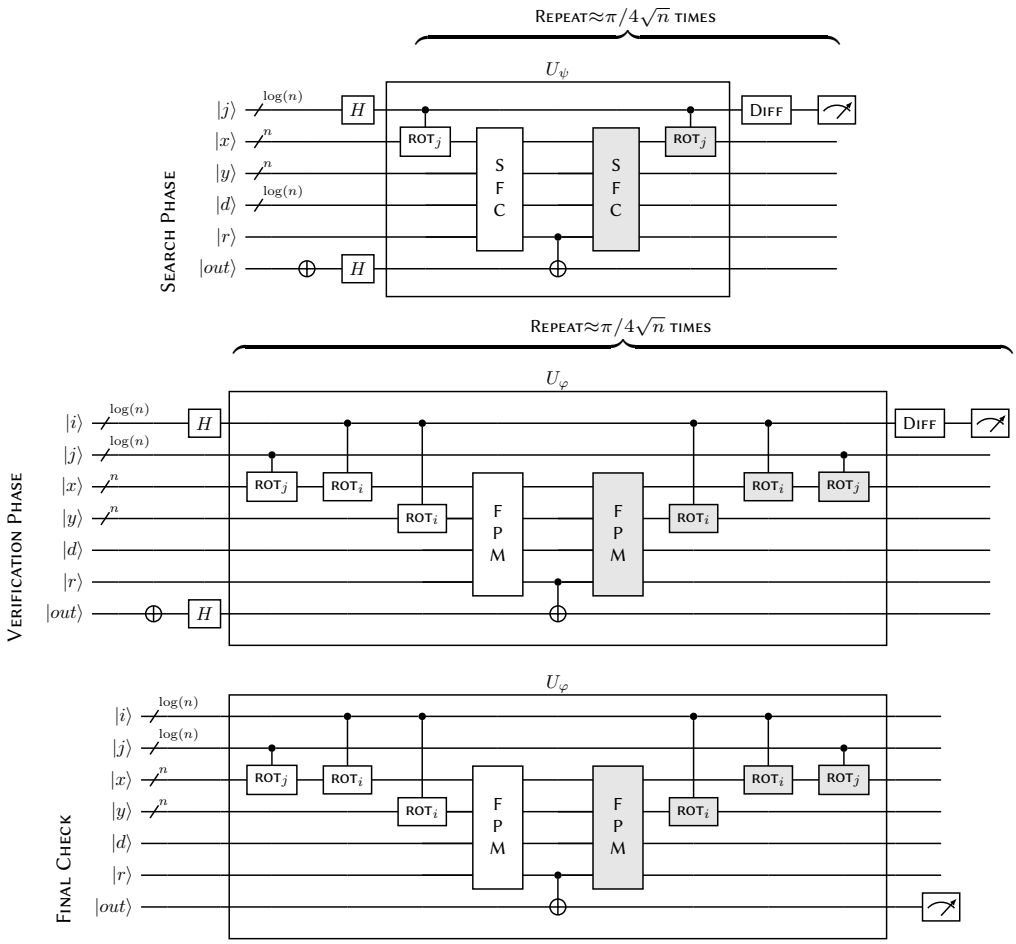


**Figure 5:** The quantum circuits implementing the three phases of the iterative test of the Quantum-LCS algorithm. For simplicity, the circuits do not contain all the ancillæ qubits. The operators in gray color represent the inverse operators.

SFC operators have a depth equal to $\mathcal{O}(\log^2(n))$ and $\mathcal{O}(\log^3(n))$, respectively. The same is true for their inverse, while the Grover's diffuser is executed in $\mathcal{O}(\log(\log(n)))$ time. Since Grover's search requires iterating the phase oracle and diffuser a number of times equal to $\mathcal{O}(\sqrt{n})$, we state that the depth of circuit implementing the search phase is $\mathcal{O}(\sqrt{n}\log^3(n))$.

Once the value $j$ has been returned by the search phase, the circuit for the verification phase again uses Grover's search algorithm to identify the position $i$, with $0 \leqslant i < n$, for which $\vec{x}^j[i\,..\,i+d-1] = y[i\,..\,i+d-1]$ holds. The circuit, depicted in the middle of Figure 5, uses the $|j\rangle$ register containing the output of the search phase, and the $|i\rangle$ register, holding the position values of the two strings, initialized to $|+\rangle^{\log(n)}$, in order to maintain, at the initial stage, the superposition of all possible position values between 0 and $n-1$. The qubit $|r\rangle$, initialized to $|0\rangle$, stores the output of the operator.

At the heart of the quantum circuit lies the application of Grover's search algorithm to the function $\varphi_{(x,y,d)}$, as outlined in equation (2). The quantum phase oracle, $U_\varphi$, for the function $\varphi_{(x,y,d)}$ operates on the input register $|i\rangle$, and is executed using the ROT and the FPM operators. The output from the FPM operator is stored in the register $|r\rangle$, while the output from the oracle $U_\varphi$ gets stored in the $|out\rangle$ register, a single qubit that is initially set to $|-\rangle$ in order to make $U_\varphi$ to behave as a phase oracle

within the Grover's search procedure.

In Figure 4, we outline the evolution of the seven registers ($|i\rangle$, $|j\rangle$, $|x\rangle$, $|y\rangle$, $|d\rangle$, $|r\rangle$, and $|out\rangle$) involved in the computation of the Boolean oracle $U_\varphi$, as depicted in Figure 5.

Specifically, we apply the ROT operator on the register $|x\rangle$, controlled by the register $|j\rangle$, allowing $|x\rangle$ to be modified in order to contain the cyclic rotation of $j$ positions. Next, an application of the rotation operator controlled by register $|i\rangle$ to both registers $|x\rangle$ and $|y\rangle$ allows the two strings to be rotated by a shift of the same value. Note that, after the application of these operators, the register $|x\rangle$ contains the superposition of all possible rotations of $\vec{x}^j$, while $|y\rangle$ contains the superposition of all its possible rotations. Formally, the application of the FPM operator on the registers $|x\rangle$ and $|y\rangle$ allows the procedure to check if $\vec{x}^j[0\,..\,d-1] = y[0\,..\,d-1]$, for all possible rotations of $\vec{x}^j$ and $y$. The oracle completes its computation by saving the output into the $|out\rangle$ register and uncomputing the entire process by applying the inverse operators in reverse order.

Regarding the depth of the circuit for the verification phase, we observe that three ROT operators have a depth equal to $\mathcal{O}(\log^2(n))$, as well as their inverse. The FPM operator is executed in $\mathcal{O}(\log^3(n))$ time-steps, while the Grover's diffuser on the register $|i\rangle$ requires $\mathcal{O}(\log(\log(n)))$ time-steps. Since Grover's search requires $\mathcal{O}(\sqrt{n})$ iteration, we can conclude that the depth of the circuit implementing the verification phase is equal to $\mathcal{O}(\sqrt{n}\log^3(n))$.

The circuit for the final check takes as input two registers, $|i\rangle$ and $|j\rangle$, containing the output of the search phase and the verification phase, respectively, and checks whether $\vec{x}^j[i\,..\,i+d-1] = y[i\,..\,i+d-1]$. Such a circuit is obtained by means of the Boolean oracle $U_\varphi$. Thus, the resulting circuit implementing the final check has a depth equal to $\mathcal{O}(\log^3(n))$.

Ultimately, the three phases achieve $\mathcal{O}(\sqrt{n}\log^3(n))$, $\mathcal{O}(\sqrt{n}\log^3(n))$, and $\mathcal{O}(\log^3(n))$ time-steps, respectively. This allows us to state that the quantum iterative test has a $\mathcal{O}(\sqrt{n}\log^3(n))$ overall depth and that, therefore, the Quantum-LCS algorithm admits an effective implementation that achieves a $\mathcal{O}(\sqrt{n}\log^4(n))$ depth in the case of binary strings. This complexity grows by a logarithmic factor, reaching $\mathcal{O}(\sqrt{n}\log^5(n))$ in the general case.

## 5. Conclusions

In this paper we provided a concrete implementation of the first quantum algorithm for the LCS problem within the circuit model, achieving a significant milestone in the development of practical quantum algorithms. While previous works in the query model offer valuable theoretical insights, our approach emphasizes the importance of circuit-based implementations for real-world applications, where practical considerations such as quantum resource requirements and hardware constraints play a crucial role. By directly encoding the input into the circuit register, we achieve a time complexity of $\mathcal{O}(\sqrt{n}\log^4(n))$, highlighting the trade-offs between time and space efficiency in quantum computation. Our results not only advance the state of quantum algorithms for LCS but also highlight the broader implications for circuit-based quantum computing, where optimizing resources is essential. Future work will continue to explore these trade-offs further and aim to refine quantum algorithms for practical deployment on quantum hardware.

We are confident that a similar approach could be used to develop a quantum solution for the Longest Palindromic Substring (LPS) problem with comparable computational complexity. In future work, we will focus on this goal, aiming to extend our findings to address the LPS problem with equivalent efficiency.

## References


[1] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Comp. 26 (1997) 1484–1509. doi:10.1137/s0097539795293172.

[2] L. K. Grover, A fast quantum mechanical algorithm for database search, in: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, STOC '96, ACM, New York, NY, USA, 1996, pp. 212–219. URL: https://doi.org/10.1145/237814.237866. doi:10.1145/237814.237866.

[3] H. Ramesh, V. Vinay, String matching in $\tilde{o}(\sqrt{n}+\sqrt{m})$ quantum time, Journal of Discrete Algorithms 1 (2003) 103–110. URL: https://www.sciencedirect.com/science/article/pii/S1570866703000108. doi:https://doi.org/10.1016/S1570-8667(03)00010-8.

[4] P. Niroula, Y. Nam, A quantum algorithm for string matching, npj Quantum Information 7 (2021). doi:10.1038/s41534-021-00369-3.

[5] D. Cantone, S. Faro, A. Pavone, Quantum string matching unfolded and extended, in: M. Kutrib, U. Meyer (Eds.), Reversible Computation - 15th International Conference, RC 2023, Giessen, Germany, July 18-19, 2023, Proceedings, volume 13960 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 117–133. URL: https://doi.org/10.1007/978-3-031-38100-3_9. doi:10.1007/978-3-031-38100-3_9.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 2nd ed., The MIT Press, 2001.

[7] Bieganski, Riedl, Cartis, Retzel, Generalized suffix trees for biological sequence data: applications and implementation, in: 1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences, volume 5, 1994, pp. 35–44. doi:10.1109/HICSS.1994.323593.

[8] P. Charalampopoulos, T. Kociumaka, S. Pissis, J. Radoszewski, Faster algorithms for longest common substring, in: P. Mutzel, R. Pagh, G. Herman (Eds.), 29th Annual European Symposium on Algorithms (ESA 2021), Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik GmbH, Dagstuhl Publishing, 2021, pp. 1–17. doi:10.4230/LIPIcs.ESA.2021.30.

[9] F. L. Gall, S. Seddighin, Quantum meets fine-grained complexity: Sublinear time quantum algorithms for string problems, Algorithmica 85 (2023) 1251–1286. URL: https://doi.org/10.1007/s00453-022-01066-z. doi:10.1007/s00453-022-01066-z.

[10] A. Ambainis, Quantum walk algorithm for element distinctness, SIAM J. Comput. 37 (2007) 210–239. URL: https://doi.org/10.1137/S0097539705447311. doi:10.1137/S0097539705447311.

[11] G. Brassard, P. Høyer, M. Mosca, A. Tapp, Quantum amplitude amplification and estimation, in: S. G. Lo Monaco, H. E. Brandt (Eds.), Quantum Computation and Information, volume 305 of *Contemporary Mathematics*, American Mathematical Society, 2002, pp. 53–74. URL: https://doi.org/10.1090%2Fconm%2F305%2F05215. doi:10.1090/conm/305/05215.

[12] S. Akmal, C. Jin, Near-optimal quantum algorithms for string problems, Algorithmica 85 (2023) 2260–2317. URL: https://doi.org/10.1007/s00453-022-01092-x. doi:10.1007/S00453-022-01092-X.

[13] F. Magniez, A. Nayak, J. Roland, M. Santha, Search via quantum walk, SIAM J. Comput. 40 (2011) 142–164. URL: https://doi.org/10.1137/090745854. doi:10.1137/090745854.

[14] D. Kempa, T. Kociumaka, String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure, in: M. Charikar, E. Cohen (Eds.), Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019, ACM, 2019, pp. 756–767. URL: https://doi.org/10.1145/3313276.3316368. doi:10.1145/3313276.3316368.

[15] S. Burkhardt, J. Kärkkäinen, Fast lightweight suffix array construction and checking, in: R. A. Baeza-Yates, E. Chávez, M. Crochemore (Eds.), Combinatorial Pattern Matching, 14th Annual Symposium, CPM 2003, Morelia, Michocán, Mexico, June 25-27, 2003, Proceedings, volume 2676 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 55–69. URL: https://doi.org/10.1007/3-540-44888-8_5. doi:10.1007/3-540-44888-8_5.

[16] A. Montanaro, Quantum pattern matching fast on average, Algorithmica 77 (2017) 16–39. URL: https://doi.org/10.1007/s00453-015-0060-4. doi:10.1007/s00453-015-0060-4.

[17] A. S. Arora, A. Coladangelo, M. Coudron, A. Gheorghiu, U. Singh, H. Waldner, Quantum depth in the random oracle model, in: Proceedings of the 55th Annual ACM Symposium on Theory of Computing, ACM, 2023. URL: https://doi.org/10.1145%2F3564246.3585153. doi:10.1145/3564246.3585153.

[18] R. Cleve, An introduction to quantum complexity theory, in: Quantum Computation and Quantum Information Theory, WORLD SCIENTIFIC, 2001, pp. 103–127. URL: https://doi.org/10.1142%2F9789810248185_0004. doi:10.1142/9789810248185_0004.

[19] E. Bernstein, U. V. Vazirani, Quantum complexity theory, SIAM J. Comput. 26 (1997) 1411–1473. URL: https://doi.org/10.1137/S0097539796300921. doi:10.1137/S0097539796300921.

[20] A. C. Yao, Quantum circuit complexity, in: 34th Annual Symposium on Foundations of Computer Science, Palo Alto, California, USA, 3-5 November 1993, IEEE Computer Society, 1993, pp. 352–361. URL: https://doi.org/10.1109/SFCS.1993.366852. doi:10.1109/SFCS.1993.366852.

[21] K. Phalak, A. Chatterjee, S. Ghosh, Quantum random access memory for dummies, 2023. arXiv:2305.01178.

[22] V. Giovannetti, S. Lloyd, L. Maccone, Quantum random access memory, Physical Review Letters 100 (2008). URL: https://doi.org/10.1103%2Fphysrevlett.100.160501. doi:10.1103/physrevlett.100.160501.

[23] V. Giovannetti, S. Lloyd, L. Maccone, Architectures for a quantum random access memory, Physical Review A 78 (2008). URL: https://doi.org/10.1103%2Fphysreva.78.052310. doi:10.1103/physreva.78.052310.

[24] D. K. Park, F. Petruccione, J.-K. K. Rhee, Circuit-based quantum random access memory for classical data, Scientific Reports 9 (2019). URL: https://doi.org/10.1038%2Fs41598-019-40439-3. doi:10.1038/s41598-019-40439-3.

[25] D. Cantone, S. Faro, A. Pavone, C. Viola, Quantum circuits for fixed substring matching problems, in: To appear in Proceedings of the 12th Computing Conference, London, United Kingdom, 11-12 July 2024, 2024. To appear.

[26] D. Deutsch, Quantum computational networks, Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences 425 (1989) 73 – 90. URL: https://api.semanticscholar.org/CorpusID:123073680.

[27] C. Moore, M. Nilsson, Parallel quantum computation and quantum codes, SIAM J. Comput. 31 (2001) 799–815. URL: https://doi.org/10.1137/S0097539799355053. doi:10.1137/S0097539799355053.

[28] F. Green, S. Homer, C. Moore, C. Pollett, Counting, fanout and the complexity of quantum ACC, Quantum Inf. Comput. 2 (2002) 35–65. URL: https://doi.org/10.26421/QIC2.1-3. doi:10.26421/QIC2.1-3.

[29] A. Broadbent, E. Kashefi, Parallelizing quantum circuits, Theor. Comput. Sci. 410 (2009) 2489–2510. URL: https://doi.org/10.1016/j.tcs.2008.12.046. doi:10.1016/J.TCS.2008.12.046.

[30] F. G. Brandão, W. Chemissany, N. Hunter-Jones, R. Kueng, J. Preskill, Models of quantum complexity growth, PRX Quantum 2 (2021) 030316. URL: https://link.aps.org/doi/10.1103/PRXQuantum.2.030316. doi:10.1103/PRXQuantum.2.030316.

[31] L. Susskind, Computational Complexity and Black Hole Horizons, Fortsch. Phys. 64 (2016) 24–43. doi:10.1002/prop.201500092. arXiv:1403.5695, [Addendum: Fortsch.Phys. 64, 44–48 (2016)].

[32] J. Haferkamp, P. Faist, N. B. T. Kothakonda, J. Eisert, N. Yunger Halpern, Linear growth of quantum circuit complexity, Nature Physics 18 (2022) 528–532. doi:10.1038/s41567-022-01539-6. arXiv:2106.05305.

[33] A. R. Brown, L. Susskind, Second law of quantum complexity, Phys. Rev. D 97 (2018) 086015. URL: https://link.aps.org/doi/10.1103/PhysRevD.97.086015. doi:10.1103/PhysRevD.97.086015.

[34] A. Pavone, C. Viola, The quantum cyclic rotation gate, in: G. Castiglione, M. Sciortino (Eds.), Proceedings of the 24th Italian Conference on Theoretical Computer Science, Palermo, Italy, September 13-15, 2023, volume 3587 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 206–218. URL: https://ceur-ws.org/Vol-3587/4071.pdf.

[35] D. Cantone, S. Faro, A. Pavone, C. Viola, Quantum circuits for fixed substring matching problems, 2023. arXiv:2308.11758, arXiv: 2308.11758.