Theoretical Computer Science 1065 (2026) 115716

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Quantum algorithms for longest common and palindromic substrings in the circuit model

Domenico Cantone [a,1], Simone Faro [a,1,*], Arianna Pavone [b,2], Caterina Viola [a,1]

[a] University of Catania, Department of Mathematics and Computer Science, Italy
[b] University of Palermo, Department of Mathematics and Computer Science, Italy

ARTICLE INFO

Editor: Dr. Pinyan Lu

2000 MSC:
68W32
68Q12

Keywords:
Quantum computing
Text processing
Sequence analysis

ABSTRACT

The Longest Common Substring (LCS) and Longest Palindromic Substring (LPS) problems are fundamental challenges in string processing, traditionally solved in linear time using classical computation through suffix trees. Recent breakthroughs by Le Gall and Seddighin [1] introduced sublinear quantum query algorithms, while Akmal and Jin [2] further improved the LCS complexity to $\tilde{O}(n^{2/3})$. While these results are remarkable in the *quantum query model*, their practical implementation on real quantum hardware remains elusive. In this paper, we bridge this gap by presenting the first $\tilde{O}(\sqrt{n})$ quantum algorithms for both LCS and LPS in the *circuit model of computation*. Our circuits are explicitly constructed and analyzed in terms of size and depth, achieving polylogarithmic overheads while preserving the $\tilde{O}(\sqrt{n})$ depth bound. This provides, for the first time, concrete circuit-level blueprints and resource estimates for quantum solutions to LCS and LPS.

1. Introduction

Quantum computing is a rapidly developing area of computer science that leverages the properties of quantum mechanics to create computing systems that can tackle problems in a markedly different way than classical computers both from the software and the hardware perspectives. Unlike classical computers, which rely on bits (either 0 or 1) to encode data, the information unit in quantum computing is the *quantum bit* aka *qubit*, which can exist in a *superposition* of classical states. Additionally, when considering multiple systems of qubits they might be in an *entangled state*. Entanglement is a physical phenomenon that allows two or more qubits to have dependent behaviours even when they are separated by a great distance. These unique features give quantum machines an advantage over classical ones, particularly in areas such as code-breaking and optimization, allowing them to perform certain calculations at an exceptional speed.

The most notable examples of quantum algorithms that exhibit a computational advantage over their known classical counterparts are Shor's algorithm [4] for factoring large numbers and Grover's algorithm [5] for unstructured database searching. These algorithms provide exponential and quadratic speed-ups over classical algorithms, respectively, serving as impressive demonstrations of the

* Corresponding author.
E-mail addresses: domenico.cantone@unict.it (D. Cantone), simone.faro@unict.it (S. Faro), ariannamaria.pavone@unict.it (A. Pavone), caterina.viola@unict.it (C. Viola).

[1] These authors are supported by the National Centre for HPC, Big Data and Quantum Computing, Project CN00000013, affiliated to Spoke 10, co-founded by the European Union - NextGenerationEU.
[2] This author is supported by PNRR project ITSERR - Italian Strengthening of the ESFRI RI RESILIENCE

https://doi.org/10.1016/j.tcs.2025.115716
Received 28 February 2025; Received in revised form 6 November 2025; Accepted 15 December 2025
Available online 21 December 2025
0304-3975/© 2025 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

power of quantum computing and sparking a surge of interest in further research and development in the field. Moreover, the recent demonstration of quantum supremacy [6] unleashed a wave of interest in quantum computing, leading to the integration of these new technologies in various areas of computer science.

Only recently have text processing and string problems become a topic of interest for quantum computer scientists. The first notable quantum algorithm for such problems was the $\tilde{O}(\sqrt{n}+\sqrt{m})$ time algorithm by Ramesh and Vinay [7], which finds the first exact occurrence of a pattern of length m in a text of length n. Subsequently, Niroula and Nam [8] presented a more general algorithm for exact string matching in $\tilde{O}(\sqrt{n})$ time. More recently, the Niroula-Nam quantum algorithm has been extended [9,10] to swap matching and other variants of approximate string matching achieving $\tilde{O}(n)$ time in the case of short patterns, i.e., when $m = O(\log(n))$.

This paper focuses on the fundamental *Longest Common Substring (LCS)* problem and the *Longest Palindromic Substring (LPS)* problem, both of which play crucial roles in string processing. The LCS problem asks for the longest substring that is common to two given input strings x and y of the same length n both over a common alphabet Σ of size σ, while the objective of the LPS problem is to determine the longest substring of a string x of length n that remains unchanged when read forward or backward.

1.1. Models and existing results on LCS and LPS

In classical computer science, the most straightforward solutions to such two problems are those based on dynamic programming, which solves both problems in $O(n^2)$ time. However, the LCS and LPS problems admit linear-time solutions [11]. Remarkably, the solutions to these problems are nearly identical and involve the construction of the generalized suffix trees [12] for the input strings and the identification of the lowest common ancestors among the tree nodes.

The question of whether it is possible to exploit the properties of quantum information to solve LCS and LPS more efficiently naturally arises.

In a recent paper [1], Le Gall and Seddighin [1] presented quantum algorithms for both LCS and LPS that assume the quantum query model [7,13] and have sublinear-time complexity. Specifically, these solutions to LCS and LPS require $\tilde{O}(n^{5/6})$ and $\tilde{O}(n^{1/2})$ oracle queries, respectively, and are based on well-known quantum procedures, such as Grover search [5], quantum string matching [7], quantum element distinctness [14], and amplitude amplification and estimation [15].

More interestingly, in [1], 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. More recently, Akmal and Jin reached in [2] this lower bound with a quantum algorithm in $\tilde{O}(n^{2/3})$ time, improving the previous result, using the MNRS quantum walk framework [16], together with a careful combination of string synchronizing sets [17] and generalized difference covers [18].

The complexity of the aforementioned solutions [1,2] is analyzed in the *quantum oracle model* (also known as the *query complexity model* [7,13,19]), where the input is accessed via a black-box oracle. In this model, the *query complexity* of an algorithm is defined as the number of oracle invocations, while the overall *time complexity* also includes the size of the quantum circuit implementing the algorithm, not counting the internal cost of oracle evaluation.

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 and concretely constructing or simulating such an oracle.

On the other hand, there are a variety of different models of quantum computation that easily and almost directly translate to concrete implementations on quantum computers[1].

The complexity of a quantum algorithm is best expressed in the computational complexity model [20], 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 [21] or the quantum circuit model [22]. 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 for details on the quantum circuit model and a discussion about the measures of 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 in [1] and [2] establishes a lower bound for both size and depth of any quantum circuit implementing the same algorithms. Consequently, the optimal input-oracle-based solution would necessitate a depth (and a size) of $\tilde{\Omega}(n^{2/3})$ (see Table 1).

1.2. Our contributions

In this paper we present the first quantum algorithm for the LCS problem in the circuit model of computation, providing actual implementation of a quantum circuit that works in $\tilde{O}(\sqrt{n})$ depth, despite its $\tilde{O}(n)$ size. Specifically, the proposed approach leads to the definition of an effective circuit with $O(\sqrt{n}\log^4(n))$-depth in the case of binary strings, and $O(\sqrt{n}\log^5(n))$-depth in the general case. For the LPS problem, we obtain two circuits having depth $O(\sqrt{n}\log^3(n))$ and $O(\sqrt{n}\log^4(n))$ in the two corresponding cases, respectively.

[1] 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.

Table 1
A comparison of quantum solutions on LPS and LCS. The time complexities for [1] and [2] are unknown due to their reliance on random-access oracles, which lack a circuit-level construction in the referenced paper.

Problem	Paper	Case	# Query	Circ. Depth
exact LPS	[1]	general case	$\tilde{O}(\sqrt{n})$	–
exact LCS	[1]	general case	$\tilde{O}(n^{5/6})$	–
exact LCS	[1]	non-repetitive	$\tilde{O}(n^{3/4})$	–
exact LCS	[2]	general case	$\tilde{O}(n^{2/3})$	–
exact LPS	This paper	general case	–	$\tilde{O}(\sqrt{n})$
exact LCS	This paper	general case	–	$\tilde{O}(\sqrt{n})$

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 input 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 analyze 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 how the two algorithms access any input string s. In [1], the authors assume a QRAM (Quantum Random Access Memory) model [23]. 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 $|i\rangle|a\rangle|z\rangle \to |i\rangle|a \otimes s[i]\rangle|z\rangle$, 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 Σ. However, we point out that the most efficient QRAM designs [24,25] exhibit a polylogarithmic-time-complexity for accessing the memory with respect to its size. In our scenario, the memory size is $O(n)$, which implies that QRAM queries will incur an additional multiplicative cost of at least $O(\log^2(n))$. Moreover, we must consider the overhead of initializing the quantum memory, which requires $O(n)$ operations [26].

In contrast, our algorithm does not rely on a random access oracle, but we assume, as in [8,27], that the input registers are already stored in a quantum memory and do not need initialization[2].

Ultimately, our approach stands apart from the previous results due to its inherent simplicity, which enables us to provide a circuit-level blueprint and to assess the quantum resources required for its implementation.

Table 1 summarizes the complexities of the discussed quantum solutions for LCS and LPS.

The paper is organized as follows. In Section 2, we fix the notation and 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. In Section 5, we show how to adapt the given algorithm for LCS to a solution of the LPS problem. Finally, in Section 6, we briefly draw our conclusions.

1.3. Extensions beyond the conference versions

This extended version substantially improves upon the two previously published conference papers [3,27]. In particular, it introduces for the first time a quantum algorithm for the *Longest Palindromic Substring* (LPS) problem, together with its complete quantum circuit implementation and complexity analysis. Furthermore, the quantum algorithm for the *Longest Common Substring* (LCS) problem, originally sketched in [3], is here described in full detail and optimized in terms of depth and modular structure. The generalization to non-binary alphabets is also explicitly treated. Compared to [27], this version unifies and reuses the fixed-length substring matching procedures as subroutines within more complex circuit architectures. In addition, the description of the FSM circuit has been substantially extended: we provide rigorous proofs of the size and depth complexity of each component (reversal, conjunction, rotation, carry, and disjunction operators), as well as of the overall circuit, thereby consolidating the formal analysis of its efficiency. Finally, a deeper theoretical discussion is included, concerning the computational implications of working within the quantum circuit model, and potential separations with classical circuit complexity.

2. Preliminaries

Basic Notations. We denote by $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{C}$ the sets of natural, integer and complex numbers, respectively.

Notation for Strings. We represent a string x of length $n \geqslant 1$, over a finite alphabet Σ of size σ, as a finite array $x[0\,..\,n-1]$, and denote by $|x|$ the length of x. The empty string has zero length and is denoted by ε. We also denote by $x[i]$ the $(i+1)$st character

[2] Like the oracle-based model, our approach assumes an idealized setting: large quantum superpositions, entangled registers, and a number of qubits proportional to the input size. While clearly beyond current hardware capabilities, we believe such assumptions are acceptable in theoretical computer science, where the goal is to explore algorithmic structure and complexity. Notably, the oracle model itself also relies on features such as QRAM and coherent data access, which are equally unfeasible today. Our contribution should thus be seen as a concrete and transparent alternative model for analyzing quantum algorithms.

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 $x = x_0 x_1 \ldots x_{n-1}$. A substring of x starting at position 0 is a *prefix* of x.

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 $\vec{x}^j$ the cyclic rightward rotation of the characters of x by j positions. More formally, we have $\vec{x}^j := x[n-j\,.\,n-1] \cdot x[0\,.\,n-j-1]$.

Elements of Quantum Information. The fundamental unit in quantum computation is the *quantum bit*, also known as *qubit*. A qubit is an element of the two-dimensional Hilbert space $\mathcal{H}$ over $\mathbb{C}$, equipped with the usual inner product. Using Dirac's notation, we denote the elements $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ from $\mathcal{H}$, respectively, by the symbols $|0\rangle$, and $|1\rangle$. The set $\{|0\rangle, |1\rangle\}$ is a basis of $\mathcal{H}$, called the *(standard) computational basis*. Namely, the elements of this basis correspond to the classical states of a (classical) bit. Using again Dirac's notation, the mathematical expression of a generic qubit, $|\psi\rangle$, is a linear combination of the two computational basis states, i.e., $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β, called *amplitudes*, are complex numbers satisfying $|\alpha|^2 + |\beta|^2 = 1$. The squared magnitudes of α and β represent the probabilities of measuring $|\psi\rangle$ in the state $|0\rangle$ or $|1\rangle$, respectively[3]. A *quantum measurement* is the only operation through which we can gain some knowledge of the state of a qubit; however, this operation causes the qubit to collapse onto a classical bit in the classical measured state, which is 0 or 1 depending on the probability distribution described by the squares of its amplitudes before the measurement is performed. The symbols $|+\rangle$ and $|-\rangle$ denote the quantum states $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, respectively. Both are uniform superpositions of the two computational basis states $|0\rangle$ and $|1\rangle$; howbeit, $|+\rangle$ and $|-\rangle$ differ in their respective *relative phase*, which is positive for $|+\rangle$ and negative for $|-\rangle$. Multiple qubit systems are referred to as *quantum registers*. A quantum register $|\psi\rangle = |q_0, q_1, \ldots, q_{n-1}\rangle$ of size n is an element from $\mathcal{H}^{\otimes n}$, that is, the tensor product of n copies of $\mathcal{H}$, and is thus expressed as a linear combination of the 2^n states in $\{0,1\}^n$. Specifically, $|\psi\rangle = \sum_{k=0}^{2^n-1} \alpha_k |k\rangle$, where the values α_k represent the probability amplitudes of the corresponding classical states $|k\rangle$ and satisfy $\sum_{k=0}^{2^n-1} |\alpha_k|^2 = 1$. Sometimes, we use $|q\rangle^{\otimes n}$ to denote a quantum register of size n such that each of its constituent qubits is in the state $|q\rangle$.

The allowed operations on systems of qubits are those permitted by quantum mechanics and can be distinguished into two classes: measurements and transformations that change the state of a quantum register into another.

Quantum measurements are operations that cause the collapse of a quantum system into a classical state and result in the loss of the probability amplitude wave. Quantum measurements are needed not only to have access to the elaboration of the information stored in qubits but also for their state preparation [28] Section 1.10 .

The dynamics of quantum systems are described by *unitary transformations* that result in the change of the state of a quantum register into another. A unitary operator is a bounded linear operator U on a Hilbert space that satisfies $U^\dagger U = UU^\dagger = I$, where $U^\dagger$ is the conjugate transpose of U, and I is the identity operator. In other words, a unitary transformation is an automorphism U of the (tensor) state space that preserves the inner product, $\langle Ux, Uy\rangle = \langle x, y\rangle$, thus ensuring the conservation of the probability amplitudes. Formally, each unitary transformation on an n-qubit register can be described by a $2^n \times 2^n$ unitary matrix with complex entries. In particular, unitary transformations are reversible, meaning that no information is lost when performing them, and the composition of multiple unitary transformations is also a unitary transformation. Any unitary transformation on an n-qubit register can be implemented as a sequence of 2-qubit unitary transformations. For this reason, 2-qubit and single-qubit unitary transformations are also called *elementary quantum operations*.

A quantum algorithm on a n-dimensional input register consists of quantum unitary transformations and measurements. Expressing a quantum algorithm using elementary quantum operations is often a challenging task that is essential for the implementation of such algorithms in real quantum machines[4].

Quantum Circuits.

This paper adopts the *quantum circuits model of computation* [20]. David Deutsch was the first to formulate the idea of the quantum circuit model [29] to encapsulate quantum computations, although, in his original formalization, Deutsch used the term *quantum network*. General computational circuits are represented as directed acyclic graphs, where nodes denote gates operating on information carried by edges. Boolean and arithmetic circuits are examples of circuit models used for classical computation. In the case of quantum circuits, each gate models either a unitary transformation or a measurement. Unlike classical Boolean counterparts, quantum circuits must be reversible (except for measurement gates) and ensure the number of input edges matches the number of output edges. Consequently, quantum circuits cannot join (fan-in) multiple wires or copy/split (fan-out) information due to the No-Cloning Theorem [30]. Thus, the graph representation features parallel wires (qubits) passing through gates. To support classical data propagation and fan-out within quantum circuits (operations which are constrained by the No-Cloning Theorem for general quantum states) *ancillary* qubits are often used to replicate basis states such as $|0\rangle$ and $|1\rangle$ without violating the theorem.

A *basis* gate corresponds to an elementary operation, and every other gate is obtained as a combination of basis gates. For example, in Boolean circuits, the basis gates typically include the negation $\neg$, the logical AND, and the logical OR gates, from which every other finite Boolean operation can be constructed. In arithmetic circuits, the basis gates are the binary addition gate, $+$, and the binary multiplication gate, $\times$. A set of elementary quantum operations from which every other can be obtained must be uncountable.

[3] when measuring it with respect to the computational basis.

[4] which can perform only a fixed small set of elementary operations.

However, as established by the Solovay-Kitaev theorem [31], any quantum operation can be efficiently approximated to any desired degree using a finite set of universal quantum gates. Furthermore, as discussed in the previous paragraph every unitary transformation can be constructed as a sequence of 2-qubit quantum operations as well as a sequence of single-qubit quantum operations and CNOT operations [32]. For this reason, any gate implementing a unary or binary quantum operation is a basis quantum gate, also referred to as an *elementary (quantum) gate*. Hereafter, we list a few of the most well-known quantum elementary gates.

There is a wide variety of elementary quantum gates that can be combined to form non-elementary gates performing more complex quantum operations. We will list the two elementary gates used in this paper:

- The *Pauli*-X (or X or NOT) gate is the quantum equivalent of the NOT gate for classical computers. It operates on a single qubit, mapping the computational basis states $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$.

  Formally, its action can be described by the following unitary matrix:

  $$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

  This matrix, when applied to a qubit state, exchanges the amplitudes of the $|0\rangle$ and $|1\rangle$ components, effectively performing a bit-flip operation.
- The *Hadamard* (or H) gate is a single-qubit operation that maps the basis states $|0\rangle$ and $|1\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and to $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, respectively, thus creating a superposition of the two basis states with equal amplitudes. This makes the Hadamard gate a fundamental tool for creating quantum superpositions. Its action is described by the unitary matrix:

  $$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$
- The *Pauli*-Z (or Z or *phase-flip gate*) is the third single-qubit operator of our interest. It leaves the basis state $|0\rangle$ unchanged while mapping $|1\rangle$ to $-|1\rangle$, that is, applying a negative phase to it. Based on the equivalence $Z = HXH$, the Z operator can be obtained from the previous two operators. Its matrix representation is:

  $$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$
- The *controlled-NOT gate* (or CNOT) is a two-qubit gate that flips the second (target) qubit if and only if the first (control) qubit is set to 1. Formally, it maps a register $|q_0, q_1\rangle$ onto $|q_0, q_0 \oplus q_1\rangle$. The matrix representation of the CNOT gate, expressed in the standard computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$, is given by:

  $$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$
- The *Toffoli gate* (also known as CCNOT gate) is a works on 3 qubits: if the first two qubits are both set to 1, it inverts the third qubit, otherwise, all qubits stay the same. Formally, it maps a register $|q_0, q_1, q_2\rangle$ onto $|q_0, q_1, q_0 q_1 \oplus q_2\rangle$. Its matrix representation in the computational basis (of dimension 8×8 for three qubits) is:

  $$\text{CCNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$
- The *swap gate* (SWAP) implements the 2-qubit operation that swaps the state of the two qubits defined by $\text{SWAP}(|q_0, q_1\rangle) = |q_1, q_0\rangle$. Interestingly, the swap gate can be achieved by the application of three CNOT operators. The matrix representation of the SWAP gate in the computational basis (dimension 4×4) is:

  $$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Graphical representations of the listed gates can be found in Fig. 1.

Generalizations of the CNOT gate are the n-ary *fanout* operator and the *multiple*-CNOT. Given a quantum register $|\psi\rangle = |q_0, q_1, \ldots, q_{n-1}\rangle$ of n qubits, a *fanout* operator simultaneously copies the control qubit $|q_0\rangle$ onto the $n-1$ target qubits $|q_i\rangle$, for $i = 1, \ldots, n-1$. Formally, the fanout operator applies the following mapping $|q_0, q_1, q_2, \ldots, q_{n-1}\rangle \mapsto |q_0, q_0 \oplus q_1, q_0 \oplus q_2, \ldots, q_0 \oplus q_{n-1}\rangle$. Although a constant-time fanout can be obtained as the product of n controlled-not gates, the no-cloning theorem [33, Box 12.1]

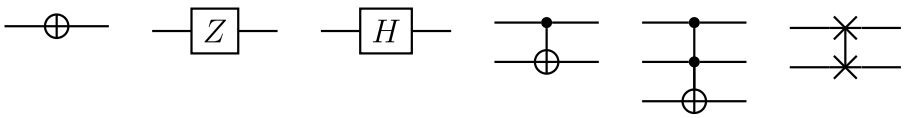

Fig. 1. The representation of the following basic gates (from left to right): Pauli-X, Pauli-Z, Hadamard, CNOT, Toffoli, and Swap.

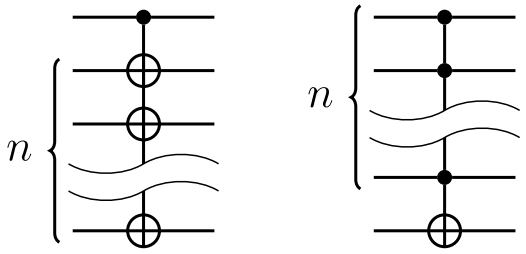

Fig. 2. The representation of n-ary *fanout* operator and multiple-CNOT with n control qubits.

makes it difficult to directly fan out qubits in constant depth [34]. However, assuming that the target qubits are all initialized to $|0\rangle$, it is easy to see that, by a divide-and-conquer strategy, we can compute fanout in depth $\Theta(\log(n))$, using controlled-not gates requiring no ancillary qubits [35][5].

A *multiple*-CNOT (or M-CNOT) operator flips the unique target qubit $|q_{n-1}\rangle$ if all the $n-1$ control qubits $|q_i\rangle$ (for $i = 0, ., n-2$) are set to $|1\rangle$. Formally, the M-CNOT operator maps $|q_0, q_1, q_2, ., q_{n-1}\rangle \mapsto |q_0, q_1, \ldots, q_{n-2}, (q_0 \cdot q_1 \cdots q_{n-2}) \oplus q_{n-1}\rangle$. The most direct way to implement an M-CNOT operator is to use the concepts of classical Boolean logic as shown in [36] for classical circuits, and later in [32] in the field of quantum computing. However, this implementation has a linear depth with respect to the number of control qubits and requires $n-2$ ancillary qubits[6]. In more recent papers [37,38], it was shown that a construction rearranging the circuit gates to achieve logarithmic depth by exploiting parallelism is possible, using $n-2$ and $n/2$ ancillary qubits, respectively[7]. Fig. 2 shows the graphical representations of a general n-ary *fanout* operator and multiple-CNOT with n control qubits.

Given a function $f : \{0,1\}^n \to \{0,1\}$, 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*. We distinguish two main classes of quantum oracles: *boolean oracles* and *phase oracles*. 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 $f : \{0,1\}^n \to \{0,1\}$ 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 relative phase of the quantum state if a 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 relative phase, which is important for algorithms like Grover's search[8].

Computational Complexity of Quantum Circuits.

There are two major measures of computational complexity for circuit models. One is the total number of basis gates, referred to as the *size* of the circuit. The other is the *depth* of the directed acyclic graph that represents the circuit. Clearly, the depth and the size of a basis gate equals $\Theta(1)$.

The size of a circuit does not consistently represent the most precise measure of complexity in quantum computation and typically offers only a rough approximation of the algorithm's intricacy.

The depth of a circuit equals the number of layers that are required for the parallel execution of the circuit [40], where a qubit can be involved in at most one interaction per layer. We observe that the depth of a circuit is, in general, not the same as the size of the circuit itself, since gates that act on disjoint sets of qubits can often be applied in parallel. As quantum gates necessitate

[5] One possible way to implement a multiple–CNOT (with one control and n targets) in logarithmic depth is to use a balanced binary tree of ancillæ, often called a "cat state" construction. The control qubit is coherently copied onto $O(n)$ ancillæ in $O(\log n)$ layers of CNOTs, so that each replica can act in parallel on a distinct target. After applying the CNOTs to all targets in one constant-depth layer, the ancillæ are uncomputed by reversing the tree. This yields depth $O(\log n)$ and size $\Theta(n)$. Variants of this construction exist, but all rely on the same tree-based replication principle.

[6] There exist different constructions for realizing an M-CNOT gate with logarithmic depth. A standard approach relies on a tree-based scheme: the $n-1$ control qubits are combined in pairs using Toffoli gates, producing $\lceil (n-1)/2 \rceil$ intermediate results stored in ancillæ. This process is iterated in parallel across $O(\log n)$ levels until a single ancilla encodes the logical AND of all controls. A single CNOT from this ancilla flips the target qubit, after which the tree is uncomputed to restore all ancillæ to $|0\rangle$. This yields a circuit of depth $O(\log n)$ and size $\Theta(n)$, at the expense of $\Theta(n)$ additional ancillæ.

[7] For the sake of completeness, we also mention a recent result [39] that enables the implementation of multi-controlled NOT gates in constant time in architectures with trapped ions and neutral atoms.

[8] It is worth noting that a phase oracle can be emulated from a Boolean oracle by introducing an ancillary qubit and exploiting the phase kickback technique.

QUANTUM-LCS(x, y, n):
1. $\ell \leftarrow 0$; $r \leftarrow n$
2. while $\ell < r$ do
3. $\quad d \leftarrow \lfloor(\ell + r)/2\rfloor$
4. $\quad$ if $\boxed{\exists\, i, j \in \{0, \ldots, n-1\} \;:\; \vec{x}^{j}[i \,..\, i+d-1] = y[j \,..\, j+d-1]}$ $\leftarrow$ QUANTUM TEST
5. $\quad\quad$ then $\ell \leftarrow d$
6. $\quad\quad$ else $r \leftarrow d - 1$
7. return ℓ

Fig. 3. 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.

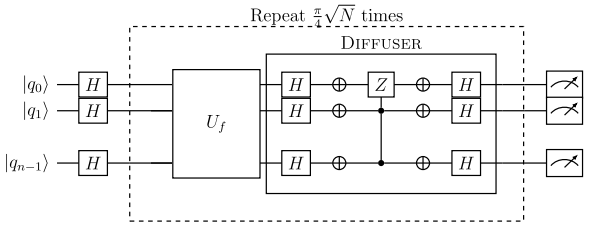

Fig. 4. The circuit implementing Grover's algorithm.

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[9].

We close this paragraph by noting that in classical computation, there is a strong relationship between space-bounded Turing Machines and depth-bounded Boolean circuits. Specifically, space-bounded computations can be efficiently simulated by bounded-depth circuits, and bounded-depth circuits can be efficiently simulated by space-bounded computations through depth-first explorations of the circuits (see [41,42] for details). Such a relationship is not known for quantum computations. While space-bounded computations on Quantum Turing Machines can be efficiently simulated by bounded-depth quantum circuits, the reverse is not known and is considered very unlikely. Beyond theoretical considerations (see [43]), practical quantum computations are limited by short coherence times, indicating that highly parallelized quantum circuits are preferable to other quantum models of computation,[10] especially from the perspective of near-term quantum computing [40,44].

For the reasons stated above, the objective of the present paper is to provide solutions to the LCS and the LPS problems in the form of quantum circuits that are efficient with respect to circuit depth rather than size.

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, that is, in order to better understand the algorithm's design, we define the quantum oracles involved in the computation without giving their actual implementation. Later, we will show that our abstract algorithm requires $\tilde{O}(\sqrt{n})$ queries to the oracles[11]. Subsequently, we will present an actual implementation of our algorithm in the circuit-based computational model. Our algorithm, named QUANTUM-LCS, consists of a quantum-based computation component and a classical-based computation one. Its simple underlying structure is summarized in the pseudocode shown in Fig. 3.

The classical part of the computation consists of a binary search for identifying the length d of the longest common substring of x and y (line 2).

During each iteration, we check for the presence of 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 ℓ and r are initialized to 0 and n, respectively. At each iteration, it checks for the presence of a common substring of length d between x and y. The efficient resolution of such a problem, which we call the *bounded substring matching* (BMS) problem, is the basis of our algorithm.

If the iterative test returns a positive answer, then the interval is narrowed to $[d \,.\, r]$ and otherwise is narrowed to $[\ell \,.\, d-1]$. The search identifies the length d of the longest common substring in $O(\log(n))$ steps.

[9] 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

[10] 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 have demonstrated the possibility of realizing long-range qubit interactions in several proposed schemes.

[11] 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.

The quantum part of the algorithm implements the test of line 4. In what follows, we will focus exclusively on the implementation of such an 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 any value 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 distinct special characters, \$ and %, respectively, not belonging to the alphabet Σ. 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 has no effect on the complexity, since the resulting string has at most twice the length of the original string.

Although 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 allow for 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, because the last characters of x and y are the special characters \$ and %, respectively, and therefore no substring obtained circularly can ever be returned as LCS.

For the sake of simplicity, we restrict our study to circuit 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) $\log(\sigma)$ bits, it is easy to show that the quantum operators used in the construction of the algorithm would undergo an increase in their depth by at most a factor of $\log(\sigma)$.

In the next two sections, we introduce the quantum operators used within the quantum circuit that implements the iteration test of the algorithm. Basically, these are the circular shift operator (Section 3.2) and the bounded match operator (Section 3.3). Finally, in Section 3.4 we describe the quantum procedure for the iterative test and discuss the overall depth of the algorithm.

3.1. Grover's search algorithm

Grover's search algorithm [5] is one of the most famous results proving theoretical quantum supremacy, together with Shor's algorithm [4]. Grover's algorithm searches a desired item within an unstructured dataset of n items in $\tilde{\mathcal{O}}(\sqrt{n})$ time [45], which is quadratically lower than the lower-bound to the runtime of an algorithm solving the same problem in a classical model of computation. The algorithm is inherently bounded-error, but we know there is no quantum algorithm making less than n queries that solves the problem with certainty for an arbitrary dataset [46].

Given a function $f : \{0,1\}^{\log(n)} \to \{0,1\}$ with a unique solution $w \in \{0,1\}^{\log(n)}$ such that $f(w) = 1$, the algorithm uses a rotation of the quantum register representing the superposition of all inputs $x \in \{0,1\}^{\log(n)}$ to increase the probability of getting the desired solution w when measuring the register.

More in detail, the algorithm starts with a register encoding the uniform superposition of all possible inputs $x \in \{0,1\}^{\log(n)}$. Note that the objective register, in which the unique solution w has amplitude 1 and all other items have amplitudes 0, is nearly orthogonal to the starting register. Thus the goal of the algorithm is to apply to the initial register a rotation as close as possible to $\pi/2$ to bring it closer to the objective register. A single iterative step of the algorithm consists of two rotations.

The first one consists in a *Phase Oracle gate* U_f for the function f, which flips the amplitude of the qubit $|w\rangle$ corresponding to the searched item, leaving the amplitudes of all other inputs unchanged. The second rotation is performed by the *Grover's Diffusion operator* (or *Diffuser*), which operates a reflection across $|+\rangle^{\otimes n}$. The overall rotation performed during a single Grover iteration is approximately equal to $2/\sqrt{n}$ radiants. Thus, $\pi/4\sqrt{n}$ iterations are necessary and sufficient. Fig. 4 represents the circuital translation of Grover's algorithm, in which details on the circuit implementing the Diffuser used by the algorithm are provided.

The Diffuser requires a multicontrolled Z gate of size n; therefore, its depth is logarithmic in n. Thus, Grover's algorithm has depth $\mathcal{O}\big(\sqrt{n}(T(n)+\log(n))\big)$,[12] where $T(n)$ is the depth of the phase oracle gate.

In the case where the function f has r solutions, with $r > 1$, $\mathcal{O}(\sqrt{n})$ iterations are still enough to find a solution, but it can be shown that roughly $\frac{\pi}{4}\sqrt{\frac{n}{r}}$ iterations are required [15], and that this bound is optimal [47]. However, in general, the value r is not known a priori and can only be obtained through a complex procedure based on the Quantum Phase Estimation. Finally, If we have r solutions and we would like to find all of them, $\Theta(\sqrt{nr})$ iterations are sufficient and necessary [48].

Observe that, when the number r of solutions is unknown, the standard version of Grover's algorithm cannot guarantee optimal success probability, which may drop to 1/2 if the number of iterations is not properly calibrated [47]. In such cases, one can rely on amplitude amplification techniques, such as the BBHT algorithm [47] or fixed-point variants [49], which preserve the $\tilde{O}(\sqrt{n})$ complexity up to logarithmic factors in the parameter.

3.2. Circular shift operator

A *circular shift operator* (or cyclic 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 $(i+s) \bmod n$. Such an operator has been effectively used

[12] If we assume to be able to implement the multicontrolled Z gate in constant time [39], a Grover's search on a dataset of n items achieves $O\big(\sqrt{n}T(n)\big)$ time.

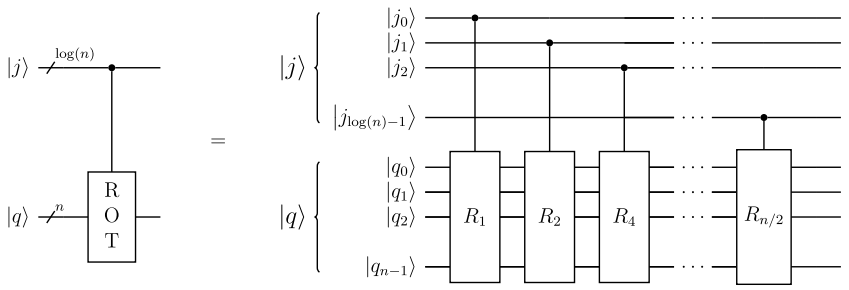

Fig. 5. A circular shift operator on a n-qubit register $|q\rangle$ controlled by a $\log(n)$-qubit register $|j\rangle$. The circuit makes use of $\log(n)$ ancilla qubits which is not shown in this graphical representation. On the left of the figure, we show the succinct graphical representation used below in this paper.

in other quantum text searching algorithms [8–10], which use a $O(\log(n))$-depth circuit implementing it. However, it is possible to perform a circular shift more efficiently via a constant-depth quantum circuit [44]. The details for constructing such a constant-depth circuit have been detailed in [50], where it is shown that the resulting operator can be executed in linear size and constant depth by two layers each made of 3 CNOT gates, both in the case of binary strings and in the general case (see Algorithm 1).

ALGORITHM 1: Algorithm constructing a quantum circuit for the rotation of a register of size n by s positions [50].

Input: n, s
Initialize n input wires (qubits) $|q_0\rangle, \ldots, |q_{n-1}\rangle$;
for $i = 1$ *to* $\lceil \frac{n}{2} \rceil - 1$ **do**
  $\texttt{Swap}|q_i, q_{n-i}\rangle$
end
for $j = 1$ *to* $\lceil \frac{n}{2} \rceil - 1$ **do**
  $\texttt{Swap}|q_{\lceil \frac{s}{2} \rceil - j}, q_{\lfloor \frac{s}{2} \rfloor + j}\rangle$
end

Suppose we operate on a circuit containing two quantum registers: the first register $|j\rangle$, of size $\lceil \log(n) \rceil$, encoding the input value related to the shift amount, and the second register $|q\rangle$, of size n, representing the register to be rotated. We denote such a controlled circuit by ROT, and its action on a register $|q\rangle$ of size n controlled by a register $|j\rangle$ (of size $\log(n)$) is formalized by

$$\mathrm{ROT}(|j\rangle \otimes |q_0, q_1, \ldots, q_{n-1}\rangle) = |j\rangle \otimes |q_{n-j}, q_{n-j+1} \ldots, q_{n-j-1}\rangle = |j\rangle \otimes |R_j(q)\rangle.$$

Because the controlled cyclic shift gate does not change the control register, we abuse the notation writing ROT_j for $\mathrm{ROT}(|j\rangle, \cdot)$. Since a register of size n can be rotated by a number of positions between 0 and $n-1$, the $|j\rangle$ register can be implemented using $\log(n)$ qubits. We have then $|j\rangle = \bigotimes_{i=0}^{\log(n)-1} |j_i\rangle$, where $|j_i\rangle$ is initialized with the value of the ith least significant bit of the binary representation of the value j. It will then be enough, for any value of i, such that $0 \leqslant i < \log(n)$, to apply to register $|q\rangle$ the rotation operator ROT_{2^i} controlled by the qubit $|j_i\rangle$. Note that the rotation operator ROT_{2^i} consists of a sequence of at most $\log(n)$ layers, each containing at most, $\frac{n}{2}$ parallel gates, controlled by the same qubit. Thus, to keep the parallelism of the cyclic shift operator in its controlled version, we apply the technique described in Section 2 that involves the use of $\frac{n}{2}\log(n)$ ancilla qubits for the application of all parallel operators controlled by the same qubit, that is, for each $0 \leqslant i \leqslant \log(n)$, the cyclic shift controlled by 2^i positions we need $\frac{n}{2}$ ancilla qubits. The controlled cyclic shift by 2^i positions requires depth $\mathcal{O}(\log(n))$ to fan the information on the control qubit out to the ancillæ plus $\mathcal{O}(1)$ to perform the cyclic rotation. Therefore, the overall controlled circular shift operator ROT_j over a register of size n and dependent on the value of an input quantum register of size $\log(n)$ can be implemented by a quantum circuit with depth equal to $\mathcal{O}(\log(n))$. Fig. 5 contains an illustration of such a circuit.

3.3. Fixed substring matching operator

In this section, we focus on the *Fixed Substring Matching* (FSM) Operator, a quantum circuit originally introduced in [27] as a unified construction for addressing different variants of fixed-length substring matching. In that work, the circuit was shown to solve the following three related combinatorial problems:

1. the *Fixed Prefix Matching* (FPM) problem, which checks whether the two input strings share a common prefix of length d;
2. the *Fixed Factor Matching* (FFM) problem, which verifies whether the two strings coincide on a substring of length d starting at a prescribed position j; and
3. the *Shared Fixed Substring Checking* (SFC) problem, which generalizes the task by asking whether such a common substring of length d exists at *any* position j.

All these problems belong to the family of *Fixed Substring Matching* problems, where the parameter d is fixed in advance. Within this framework, the FPM problem can be seen as a special case of FFM (with $j = 0$), while SFC extends FFM by ranging over all valid positions. The FSM provides a single quantum primitive capable of supporting each of these formulations.

In the following, we adopt the name FSM to emphasize its use in the Fixed Substring Matching setting; however, throughout the literature and depending on the specific problem under consideration, the same circuit may be referred to by different names (e.g., FPM, FFM, or SFC). For clarity, we will treat these as instances of the same underlying operator.

3.3.1. Definitions of core components of FSM

We first give the notion of *matching substring vectors*, originally introduced in [27]. Let x and y be two strings of length n. For any integer $i \geqslant 0$, the matching substring vector λ^i of x and y is defined as the bit vector of length n such that $\lambda^i[j] = 1$ whenever the substrings $x[j \,..\, j + 2^i - 1]$ and $y[j \,..\, j + 2^i - 1]$ are identical, and $\lambda^i[j] = 0$ otherwise. Such vectors can be efficiently computed in a recursive fashion, starting from λ^0, the vector whose entries $\lambda^0[j]$ are the character-wise comparisons of x and y, and combining the results to handle increasing substring lengths. It is easy to check that $\lambda^0[j] = 1$ only if $x[j] = y[j]$ and that, in general, $\lambda^i[j]$ is equal to the conjunction of two entries in λ^{i-1}, corresponding to adjacent blocks of length 2^{i-1}. We exploit this recursive structure to identify equal substrings of any power-of-two length. For example, given the strings $x = \texttt{cgaactta}$ and $y = \texttt{ctacctta}$, we obtain $\lambda^0 = [1\ 0\ 1\ 0\ 1\ 1\ 1\ 1]$, $\lambda^1 = [0\ 0\ 0\ 0\ 1\ 1\ 1\ 0]$, and so on.

Combining the two notions given above, we get the following result [27] stating that a substring of length d starting at position j in both x and y is equal if and only if, for every power 2^i in the decomposition of d, the corresponding bit in λ^i is set to 1 at the appropriate shifted position.

Lemma 3.1 (from [27]). *Let $d > 0$ and let $\bar{d}$ denote its reverse binary representation. Then, for any $j < n - d$:*

$$x[j \,..\, j + d - 1] = y[j \,..\, j + d - 1] \iff \bigwedge_{i \in S} \lambda^i \left[j + \bar{d}[0 : i - 1] \right] = 1,$$

where S is the set of bit positions i such that $\bar{d}[i] = 1$.

Note that $\bar{d}[0 : i - 1]$ is not a binary array, but rather the integer value represented by the first i bits of the reversed binary form of d.

We refer the reader to [27] for the full derivation and additional properties of this technique. Following the idea underlying Lemma 3.1, we employ a family of auxiliary bit-vectors D^i, indexed by i, that characterize the presence of common substrings between x and y. Each vector D^i encodes the information on matching substrings of length $\bar{d}[0 : i]$, according to the power-based decomposition of the target length d. The bit at position $j + \bar{d}[0 : i]$ is set to 1 if and only if a common substring of length $\bar{d}[0 : i]$ begins at position j of both strings.

Such vectors are built incrementally: D^0 corresponds to matching substrings of the shortest relevant length, and the subsequent vectors D^i combine and extend this information to longer substrings, aligned with the binary expansion of d.

To simplify the base case of the construction, we assume that the empty string is trivially shared between x and y at every position. As a result, the initial vector is set to $D^{-1} = 1^{n+1}$[13].

3.3.2. Circuit logic for the FSM procedure

In this section, we discuss the ideas underlying the workflow of the Fixed Substring Matching (FSM) operator and describe its quantum circuit implementation in terms of elementary at-most-ternary gates that compose it.

Fig. 6 contains the pseudocode of the algorithm designed to solve the general FSM problem. The algorithm proceeds in three distinct stages and takes as input the strings x and y, the matching length d, and an auxiliary bit vector D^{-1}, which is initialized according to the specific application [14].

Let $L = \lfloor \log(d) \rfloor$. In its first stage (*initialization stage*), the algorithm initializes the matching vector λ^0 (line 1) and the corresponding vector D^0 (line 2). This stage is followed by a loop (*extension stage*) of L iterations (lines 3-6) that compute the vectors λ^i and D^i, by exploiting the recursive relations described in the previous section. Upon completion of the loop, the algorithm computes a bitwise OR over the entries of the final vector D^L, and, finally, the resulting bitwise OR value is output (line 7), indicating the presence or absence of a fixed-length match.

We provide now a detailed description of the quantum circuit implementing the algorithm presented above. Such a quantum circuit is depicted in Fig. 7, and our description will closely follow its structure, presenting each operator that composes it, and analyzing its complexity in terms of both size and depth.

3.3.3. Registers of the FSM quantum circuit

The circuit implementing the FSM operator utilizes two registers, $|x\rangle$ and $|y\rangle$, each of size n, which store the characters of the input strings x and y, respectively. These registers are assumed to be preloaded in quantum memory and do not require initialization.

[13] Here, 1^{n+1} denotes the binary vector of length $n + 1$ with all bits set to one.

[14] The vector D^{-1} serves as an initialization step, tailored to the specific problem being addressed. In particular, if the goal is to check the equality of a specific substring starting at position i, the corresponding bit in the vector must be initialized to 1. Conversely, if the objective is to verify all possible positions, the vector should be initialized with all bits set to 1.

$\mathrm{FSM}(d, x, y, D^{-1})$:
1. $\lambda^0 \leftarrow \mathsf{M}(x, y)$
2. if $(\bar{d}[0] = 1)$ then $D^0 \leftarrow (D^{-1} \wedge \lambda^0) \gg 2^0$
3. $L \leftarrow \lfloor \log(d) \rfloor$
4. for $i \leftarrow 1$ to L do
5. $\quad \lambda^i \leftarrow \mathrm{EXT}_i(\lambda^{i-1})$
6. $\quad$ if $(\bar{d}[i] = 1)$ then $D^i \leftarrow (D^{i-1} \wedge \lambda^i) \gg 2^i$
7. $\quad$ else $D^i \leftarrow D^{i-1}$
8. $r \leftarrow \bigvee_{k=d}^{n} D^L[k]$
9. return r

Fig. 6. The pseudocode of the FSM algorithm.

A separate register, $|d\rangle$, of size L, holds the fixed-length value d. Initializing this register requires $O(\log(n))$ time. For simplicity, in Fig. 7, we assume that d is within the range $1 \leqslant d < 8$, which allows it to be represented using 3 bits. In our quantum implementation, each vector D^i is stored in a quantum register $|D^i\rangle$ of size $n+1$, and each vector λ^i is stored in a quantum register $|\lambda^i\rangle$ of size n. Here the index i ranges over $0 \leqslant i \leqslant L$, with $L \leqslant \log(n)$, so that representing i requires at most $O(\log n)$ qubits. The final result of the computation is stored in the single-qubit register $|r\rangle$.

3.3.4. Initialization of the FSM circuit

The initialization step involves the application of the *reversal* operator to the register $|d\rangle = |d_{m-1} \dots d_1 d_0\rangle$, where $m = \lceil \log_2(d+1) \rceil$ is the number of qubits required to encode d. This operator maps $|d\rangle$ onto $|\bar{d}\rangle = |d_0 d_1 \dots d_{m-1}\rangle$, i.e., it reverses the order of the qubits. Formally, the reversal operator R_m is defined by

$$R_m \, |d_{m-1} \dots d_1 d_0\rangle = |d_0 d_1 \dots d_{m-1}\rangle, \tag{1}$$

and it is implemented by swapping qubit i with qubit $m-1-i$, for all $0 \leqslant i < \lfloor m/2 \rfloor$.

Lemma 3.2 (Complexity of the reversal operator)**.** *The m-qubit reversal operator R_m admits an implementation of size $\Theta(m)$ and constant depth:*

- *if SWAP is primitive, the circuit requires $\lfloor m/2 \rfloor$ SWAP gates in parallel (depth 1);*
- *if SWAP is decomposed into three CNOTs, the depth is 3 and the two-qubit gate count is $3\lfloor m/2 \rfloor$.*

Proof. The operator R_m consists of $\lfloor m/2 \rfloor$ disjoint transpositions. All swaps can be applied in parallel since they act on disjoint qubit pairs. If SWAP is primitive, the depth is 1; otherwise each SWAP is replaced by its standard three-CNOT decomposition, scheduled in three parallel layers, giving depth 3. In both cases the gate count is $\Theta(m)$, hence the size is linear and the depth constant. □

Remark 3.3. The assumption that the pattern and text registers are preloaded with input data is standard in both classical and quantum string-matching algorithms. The only initialization cost considered here is the reversal of $|d\rangle$, whose complexity is as stated in Lemma 3.2.

3.3.5. Computing the $|\lambda\rangle$ quantum registers

The registers holding the matching substring vectors λ^i are computed in the first phase of the algorithm through the application of the matching operator M (for the initialization of λ^0) and the extension operators EXT_i for subsequent iterations. Formally, the M operator performs, for all $x, y \in \{0,1\}^n$, the following mapping

$$\mathrm{M}|x\rangle|y\rangle|0^n\rangle = |x\rangle|y\rangle|\lambda^0\rangle$$

and is based on the following simple logical relation

$$\lambda^0[j] = (x_j \wedge y_j) \vee (\neg x_j \wedge \neg y_j).$$

Thus, the matching gate M can be implemented by two sets of n parallel Toffoli gates, the second of which is squeezed by two layers of parallel X gates. Fig. 8 (left) illustrates the quantum circuit implementing the M match operator, acting on two input registers x and y, each consisting of 8 qubits. The output register λ^0, of the same size, stores a 1 in position j if and only if $x_j = y_j$.

The extension operator EXT_i performs, for all $0 < i \leqslant \log(n)$, the following mapping

$$\mathrm{EXT}_i|\lambda^{i-1}\rangle|0^n\rangle = |\lambda^{i-1}\rangle|\lambda^i\rangle$$

and its computation is based on the following simple logical relation

$$\lambda^i[j] = \lambda^{i-1}[j] \wedge \lambda^{i-1}[j + 2^{i-1}].$$

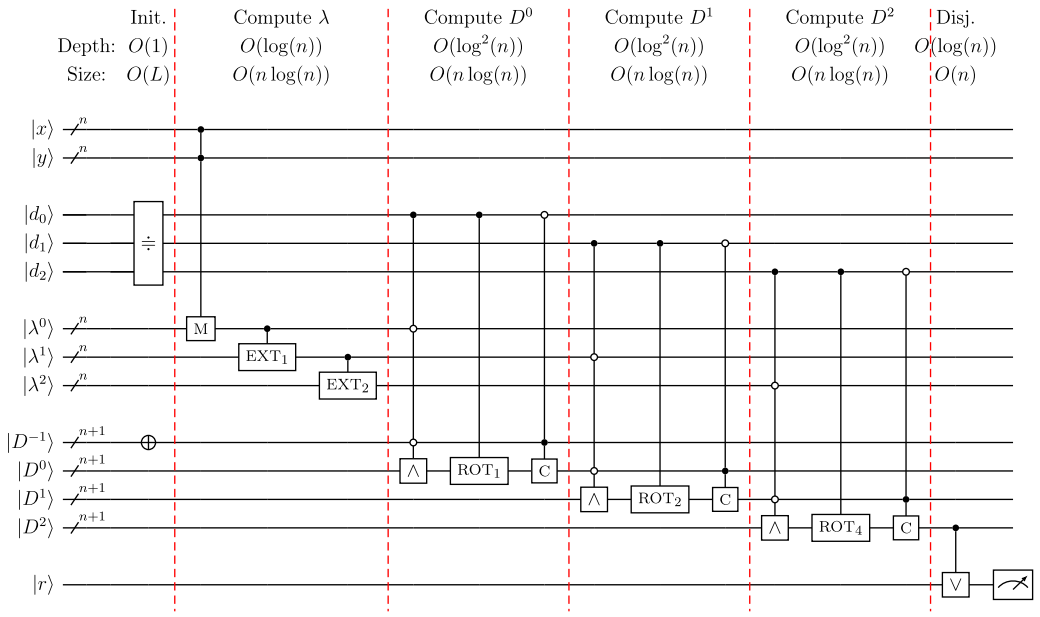

Fig. 7. Simplified quantum circuit for the FSM procedure. The circuit takes as input the registers encoding the text x, the pattern y, and the offset d, and proceeds in stages: *(i)* initialization of the auxiliary register D^{-1}; *(ii)* computation of the matching vectors λ^i via the operator M and their extension operators EXT_i; *(iii)* iterative construction of the vectors D^i using controlled conjunctions ($\wedge$), controlled rotations (ROT_i), and the CRC operator; and *(iv)* final disjunction ($\vee$) and measurement of the result qubit r. Each block is annotated with its asymptotic depth and size, showing that the overall circuit achieves depth $O(\log^2(n))$ and size $O(n \log n)$.

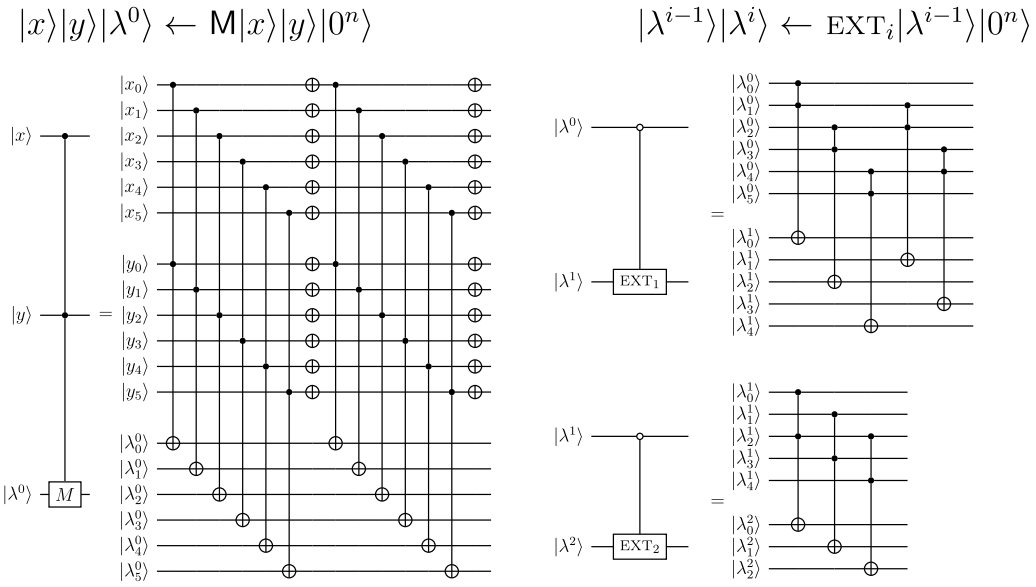

Fig. 8. (Left) Graphical representation of the M match operator acting on two input registers x and y, each composed of 8 qubits. The output register λ^0, of equal length, has its jth qubit set to 1 if and only if $x_j = y_j$. (Right) Graphical representation of two extension operators used to compute the vector λ^i from λ^{i-1}. The jth qubit of λ^i is set to 1 if and only if both qubits at positions j and $j + 2^{i-1}$ in λ^{i-1} are equal to 1. The compact notation for each operator is displayed alongside its corresponding circuit.

Therefore, the EXT_i operator can be implemented by means of a set of Toffoli gates. Fig. 8 (right) displays the quantum circuit representations of two extension operators used to compute the vector λ^i from λ^{i-1}. The jth qubit of λ^i is set to 1 if and only if both qubits at positions j and $j + 2^{i-1}$ in λ^{i-1} are equal to 1.

Lemma 3.4 (Complexity of computing the matching vectors λ^i). *Let $x, y \in \{0,1\}^n$ and define the bit-vectors $\lambda^0, \lambda^1, \ldots, \lambda^L$ by*

$$\lambda^0[j] = \mathbf{1}[x_j = y_j] \qquad \textit{and} \qquad \lambda^i[j] = \lambda^{i-1}[j] \wedge \lambda^{i-1}[j + 2^{i-1}] \ \textit{ for } 1 \le i \le L,$$

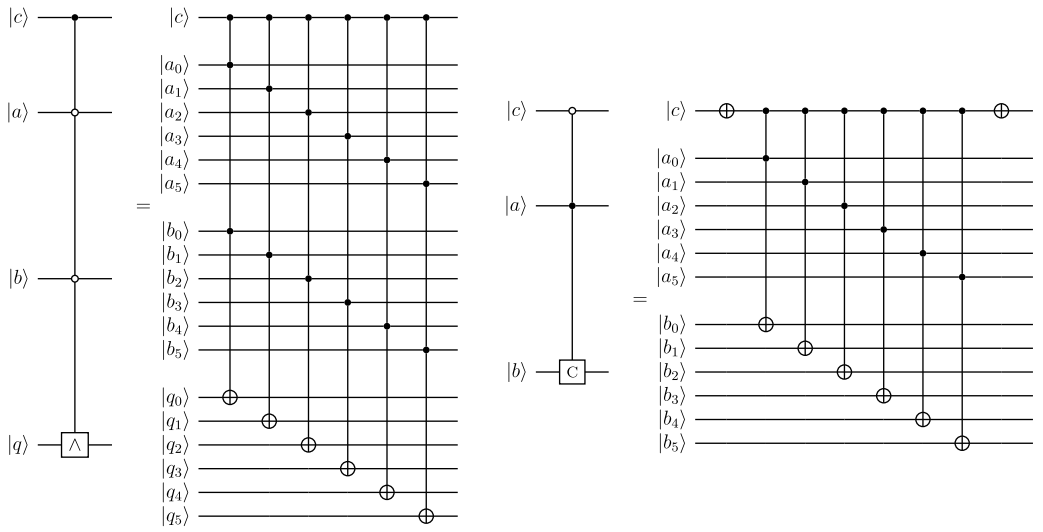

Fig. 9. (left) The graphical representation of the bitwise conjunction operator between two quantum registers $|a\rangle$ and $|b\rangle$ of 6 qubits; and (right) the graphical representation of the copy operator with reversal control over one input registers $|a\rangle$ of 6 qubits. The resulting qubit, $|b\rangle$, is a copy of $|a\rangle$ only if the control qubit $|c\rangle$ is set to 0. On the right of each of the operator is shown its compact graphical form.

where indices in the second term are valid for $0 \le j < n - 2^{i-1}$. *Then there exists a quantum circuit which, on input* $|x\rangle|y\rangle|0^{n(L+1)}\rangle$, *outputs* $|x\rangle|y\rangle|\lambda^0\rangle \cdots |\lambda^L\rangle$ *with*

- *circuit depth* $O(L)$,
- *circuit size* $\Theta(nL)$,

in the standard circuit model where two-qubit (or three-qubit) gates acting on disjoint sets of qubits can be executed in parallel. In particular, taking $L = \lfloor \log n \rfloor$ *yields depth* $O(\log n)$ *and size* $\Theta(n \log n)$, *while taking* $L = \lfloor \log d \rfloor$ *yields depth* $O(\log d)$ *and size* $\Theta(n \log d)$.

Proof. We first consider the initialization stage (M operator). The vector λ^0 implements the component-wise equality check between x and y. Using the identity $\mathbf{1}[x_j = y_j] = (x_j \wedge y_j) \vee (\neg x_j \wedge \neg y_j)$, one can compute $\lambda^0[j]$ with a constant number of two-/three-qubit gates per position j, e.g., two parallel layers of Toffoli gates with suitable X gates to handle negations. All n positions are independent, hence this stage has constant depth and gate count $\Theta(n)$.

Next, we consider the first part of ith iteration of the extension stage for a fixed i (EXT_i). For each valid j, the bit $\lambda^i[j]$ is the conjunction of two entries of λ^{i-1} with indices j and $j + 2^{i-1}$. We implement $\lambda^i[j]$ by a single Toffoli with controls $\lambda^{i-1}[j]$ and $\lambda^{i-1}[j + 2^{i-1}]$ and target the fresh qubit $\lambda^i[j]$. Consider the interaction graph $G_i = (V, E_i)$ where $V = \{0, 1, \ldots, n-1\}$ and

$$E_i = \{\, \{j, j + 2^{i-1}\} \;:\; 0 \le j < n - 2^{i-1} \,\}.$$

Each vertex has degree at most 2, so G_i is a disjoint union of paths and is therefore bipartite; in particular, its edge-chromatic number is 2. Hence the edge set E_i can be partitioned into two matchings $E_i^{(0)}$ and $E_i^{(1)}$ so that no two edges in the same matching share a vertex. Scheduling the Toffoli gates corresponding to $E_i^{(0)}$ in one parallel layer and those corresponding to $E_i^{(1)}$ in a second parallel layer computes all the targets $\lambda^i[j]$ in *constant* depth for this i. The number of Toffolis used at level i is $\Theta(n)$ (up to boundary effects), so the gate count per level is $\Theta(n)$.

Overall, the initialization has constant depth and $\Theta(n)$ gates, while each level $i = 1, \ldots, L$ has constant depth (two parallel Toffoli layers plus $O(1)$ layers of single-qubit gates, if any) and uses $\Theta(n)$ gates. Summing over L levels yields overall depth $O(L)$ and size $\Theta(nL)$.

Gate set considerations. If Toffoli is decomposed into a constant number of two-qubit gates and single-qubit gates, the above depth bounds incur only a constant-factor increase, leaving the asymptotic complexity bounds unchanged. □

3.3.6. *Computing the* $|D\rangle$ *quantum registers*

The the second part of the ith iteration of the extension stage computes the vector $|D^i\rangle$, for $0 \leqslant i \leqslant L$. According to the pseudocode outlined in Fig. 6 (lines 5-6), if $|d_i\rangle$ equals $|1\rangle$, then $|D^i\rangle$ is obtained by first performing a bitwise AND between $|D^{i-1}\rangle$ and $|\lambda^i\rangle$, and then applying a right shift by 2^i positions to the resulting register. Conversely, if $|d_i\rangle$ equals $|0\rangle$, $|D^i\rangle$ simply retains the value of $|D^{i-1}\rangle$.

In what follows, we detail the circuit-level realization of the primitives required by the update rule for the registers $|D^i\rangle$: (i) the *controlled bitwise conjunction* operator, (ii) the *rotation* operator (already described in Section 3.2), and (iii) the *copy operator with reversal control*.

The *controlled bitwise conjunction* operator applies a bitwise logical AND between two n-qubit registers, $|a\rangle$ and $|b\rangle$, storing the result in an n-qubit target register $|q\rangle$, but only when a single-qubit control register $|c\rangle$ is set to $|1\rangle$. Otherwise, all registers remain unchanged. Formally, for all $a, b \in \{0,1\}^n$, the mapping is given by

$$\wedge\,|c\rangle|a\rangle|b\rangle|0^n\rangle \;=\; |c\rangle|a\rangle|b\rangle \bigotimes_{i=0}^{n-1} |c \wedge a_i \wedge b_i\rangle.$$

Its implementation uses n triple-controlled gates (with $|c\rangle$, $|a_i\rangle$, and $|b_i\rangle$ as controls, and $|q_i\rangle$ as target) for $0 \leqslant i < n$. Since all gates share the same control qubit $|c\rangle$, they cannot be directly applied in parallel. To enable parallelism, the value of $|c\rangle$ is first copied to n ancilla qubits, allowing all n gates to be executed simultaneously and resulting in a circuit depth of $O(\log n)$. Fig. 9 depicts the graphical representation of the controlled bitwise conjunction operator over input registers of 6 qubits.

The *copy operator with reversal control* (or simply *CRC operator*) is designed to transfer the value of the qubits of a first register $|a\rangle$ into the corresponding qubits of a second register $|b\rangle$, assuming that $|a\rangle$ and $|b\rangle$ have the same length and that $|b\rangle$ is initially in the all-zero state $|0^n\rangle$. Unlike a general copying operation (which is not possible for arbitrary quantum states), this transfer is valid because the target register is prepared in a known computational basis state.

Formally, the CRC operator applies the following transformation to any input state $|c\rangle|a\rangle|0^n\rangle$, where $a \in \{0,1\}^n$ and $c \in \{0,1\}$:

$$\mathrm{CRC}\,|c\rangle|a\rangle|0^n\rangle = |c\rangle|a\rangle \bigotimes_{i=0}^{n-1} |\neg c \wedge a_i\rangle.$$

Its implementation relies on a sequence of n Toffoli gates, with the control qubit inverted by a preceding X gate so that the operation is triggered when $|c\rangle = |0\rangle$. Specifically, for each index $i \in \{0, \ldots, n-1\}$, a Toffoli gate uses $|c\rangle$ and $|a_i\rangle$ as control qubits and $|b_i\rangle$ as the target. Since all gates share the same control qubit $|c\rangle$, a direct parallel application is not possible. However, a standard fanout strategy can be employed: the value of $|c\rangle$ is first replicated into n ancillary qubits, which are then used to independently control the n Toffoli gates in parallel. By leveraging this technique, the CRC operator can be realized by a quantum circuit with depth $O(\log n)$. Fig. 9 shows the graphical representation of the CRC operator over an input registers $|a\rangle$ of 6 qubits.

Lemma 3.5 (Depth/size of the circuit computing the registers $|D^i\rangle$)**.** *Let $n \in \mathbb{N}$ and let $d \geq 1$ with binary length $L + 1 = \lfloor \log d \rfloor + 1$. Assume that for each iteration $i \in \{0, \ldots, L-1\}$ the update $|D^{i-1}\rangle \mapsto |D^i\rangle$ is implemented by the sequential composition of:*

1. *a* controlled bitwise conjunction that computes the bitwise AND of $|D^{i-1}\rangle$ and $|\lambda^i\rangle$ conditioned on $|d_i\rangle$,
2. a controlled rotation (right shift by 2^i positions) conditioned on $|d_i\rangle$,
3. a CRC operator that fixes the branch $d_i = 0$.

Under the standard parallel-disjoint-gates model, and using the fanout technique to replicate a one-qubit control into n per-qubit controls in $O(\log n)$ depth, each of the three primitives above admits an implementation with depth $O(\log n)$ and size $\Theta(n)$. Therefore, the whole computation of $|D^0\rangle, \ldots, |D^L\rangle$ has

$$\textit{depth}\;\; O(\log d \cdot \log n) \qquad \textit{and} \qquad \textit{size}\;\; \Theta(n \log d).$$

In particular, since $\log d \leq \log n$, the worst-case depth is $O(\log^2 n)$.

Proof. We analyze one iteration i and then sum over $i = 0, \ldots, L-1$.

(1) Controlled bitwise conjunction. This primitive applies, for each position j, a constant-size (e.g., Toffoli-based) gate controlled by $|d_i\rangle$ to compute the bitwise AND between the jth bits of $|D^{i-1}\rangle$ and $|\lambda^i\rangle$ into the jth target wire. To enable parallelism across all j, the single control $|d_i\rangle$ is fanned out to n ancillæ via a balanced tree in depth $O(\log n)$ and size $\Theta(n)$. All n per-position gates then run in $O(1)$ parallel layers on disjoint targets. Hence the controlled conjunction has depth $O(\log n)$ and size $\Theta(n)$.

(2) Controlled rotation by 2^i. By the construction recalled in Section 3.2, a cyclic shift by 2^i positions controlled by a one-qubit control can be implemented by (i) fanning out the control to $\Theta(n)$ ancillæ in depth $O(\log n)$, then (ii) applying the required parallel swap/permute layers in $O(1)$ depth. Thus the controlled rotation has depth $O(\log n)$ and size $\Theta(n)$.

(3) Copy operator with reversal control (CRC). As described in the text, CRC applies n parallel constant-size controlled gates, again requiring a fanout of the one-qubit control into n ancillæ. Therefore, CRC also runs in depth $O(\log n)$ with size $\Theta(n)$.

Per-iteration and total bounds. Within one iteration, the three primitives are composed *sequentially*, so the per-iteration depth is the sum of three $O(\log n)$ terms, i.e., $O(\log n)$, and the per-iteration size is $\Theta(n)$. There are $L = \lfloor \log d \rfloor$ iterations, which must be executed sequentially because $|D^i\rangle$ depends on $|D^{i-1}\rangle$. Summing over all iterations yields total depth $O(L \log n) = O(\log d \cdot \log n)$ and total size $\Theta(nL) = \Theta(n \log d)$. Finally, since $\log d \leq \log n$, the worst-case depth is $O(\log^2 n)$. □

3.3.7. Computing the final output

The last step of the circuit consists in computing the final output by applying a disjunction between the qubits in the register $|D^L\rangle$. This is done via the *register disjunction operator* that performs, for all $a \in \{0,1\}^n$, the following transformation.

$$\vee|a\rangle|0\rangle = |a\rangle\left|\bigvee_{i=0}^{n-1} a_i\right\rangle. \tag{2}$$

The implementation of the disjunction operator can be realized by a fanout operator on the $|a\rangle$ registers, and with target $|r\rangle$, squeezed by two layers of X operators on the $|a\rangle$ qubits (see Fig. 10). Thus, the depth of the resulting circuit is equal to $O(\log(n))$.

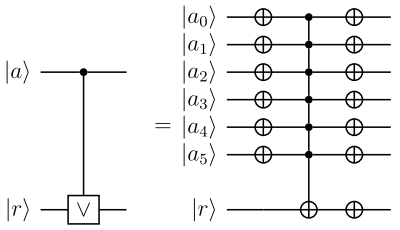

Fig. 10. The graphical representation of the register disjunction operator over input registers $|a\rangle$ of 6 qubits each. The resulting qubit, $|r\rangle$, is set to 1 if at least one of the qubits of $|a\rangle$ is set 1. On the right of each of the operators is shown its compact graphical form.

Lemma 3.6 (Complexity of the register disjunction)**.** *Let the symbol $\vee$ denote the register disjunction operator that, on input $|a\rangle|0\rangle$ with $a \in \{0,1\}^n$, defined as in Eq. (2). Then there exists a quantum circuit implementing $\vee$ in depth $O(\log n)$ and size $\Theta(n)$ in the standard parallel-disjoint-gates model. The same asymptotic bounds hold if Toffoli gates are decomposed into two-qubit gates up to a constant factor.*

Proof. One can compute $\bigvee_i a_i$ as $\neg \bigwedge_i \neg a_i$ by first applying X to all a_i, then a multi-controlled Toffoli onto the output qubit, and finally X on the output and on all a_i. Realizing the n-control Toffoli with a balanced ancilla construction gives depth $O(\log n)$ and size $\Theta(n)$; see standard decompositions of multi-controlled Toffoli gates. □

Theorem 3.7 (Overall complexity of the FSM circuit)**.** *Let $x, y \in \{0,1\}^n$ and let $d \geq 1$ with binary length $L = \lfloor \log d \rfloor$. The quantum circuit implementing the FSM procedure, which (i) initializes the control register by reversal, (ii) computes the matching vectors $\lambda^0, \ldots, \lambda^L$, (iii) performs L iterative updates to obtain $D^0, \ldots, D^L$, and (iv) computes the final disjunction $\bigvee_j D^L[j]$, admits an implementation with*

$$\textit{depth}\ \ O(\log d \cdot \log n) \qquad \textit{and} \qquad \textit{size}\ \ \Theta(n \log d),$$

under the standard parallel-disjoint-gates model (with Toffoli implementable at constant depth up to two-qubit decompositions). In particular, since $\log d \leq \log n$, the worst-case bounds are

$$\textit{depth}\ \ O(\log^2 n) \qquad \textit{and} \qquad \textit{size}\ \ \Theta(n \log n).$$

Proof. (Initialization) The reversal R_n has constant depth and size $\Theta(n)$ (Lemma 3.2). (Computation of λ^i) Computing $\lambda^0, \ldots, \lambda^L$ uses depth $O(L)$ and size $\Theta(nL)$ (Lemma 3.4), i.e., depth $O(\log d)$ and size $\Theta(n \log d)$. (Iterative updates for D^i) Each of the three primitives per iteration-controlled bitwise conjunction, controlled rotation by 2^i, and CRC-has depth $O(\log n)$ and size $\Theta(n)$ using standard fanout on the 1-qubit control. Over L sequential iterations this yields depth $O(L \log n) = O(\log d \cdot \log n)$ and size $\Theta(nL) = \Theta(n \log d)$ (Lemma 3.5). (Final disjunction) The register disjunction (OR-reduction) has depth $O(\log n)$ and size $\Theta(n)$ (Lemma 3.6). Summing the contributions, the depth is dominated by the D-phase, $O(\log d \cdot \log n)$; the size is $\Theta(n \log d)$. □

3.3.8. Initialization variants for different problem settings

The bounded matching operator described above provides a common quantum framework for solving the three fixed substring matching problems, i.e., FPM, FFM, and SFC. The distinguishing factor among these problems lies exclusively in the initialization of the register $|D\rangle^{-1}$, while the rest of the circuit remains unchanged.

- For the *Fixed Prefix Matching* (FPM) problem, $|D\rangle^{-1}$ is initialized so as to restrict the matching process to the prefix region, i.e., alignment at position $j = 0$. In this context $|D\rangle^{-1}$ is initialized to $|1\rangle|0\rangle^n$.
- For the *Fixed Factor Matching* (FFM) problem, $|D\rangle^{-1}$ is initialized to encode the prescribed starting position j, thereby constraining the computation to substrings beginning at that offset. In this context $|D\rangle^{-1}$ is initialized to $|0\rangle^j|1\rangle|0\rangle^{n-j}$.
- For the *Shared Fixed Substring Checking* (SFC) problem, the initialization of $|D\rangle^{-1}$ is set to allow all possible alignments j, enabling the circuit to test for the existence of a shared substring at any valid position. In this context $|D\rangle^{-1}$ is initialized to $|1\rangle^{n+1}$.

Hence, all the three variants of the problem reduce to the same quantum computation, differing only by the way in which the initial state of $|D\rangle^{-1}$ is prepared. This highlights the versatility of the bounded matching operator as a general-purpose primitive for substring comparison tasks.

3.4. 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 that implements the iterative test is illustrated in Fig. 11. It consists of three phases, each of which is 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 input to the final check. In the current section, we describe the details of the role, structure, and depth-complexity for each of such phases. Finally, we discuss the overall depth-complexity of the iterative test.

The *search phase* makes use of Grover's search algorithm [5] for finding (with high probability) a solution (if any) to a black box function, making just $O(\sqrt{n})$ queries to the function. We refer to Section 3.1 for a detailed discussion of a quantum circuit implementing Grover's search and its complexity.

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)

Fig. 11. The structure of the quantum test used in the algorithm in Fig. 3. The test consists of 3 phases, each of which is implemented as a quantum procedure.

Specifically, the input black box to the algorithm is accessed by a phase oracle P_ψ implementing the function $\psi_{(x,y)} : \{0,\dots,n-1\} \times \{0,\dots,n-1\} \longrightarrow \{0,1\}$, 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_ψ 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

$$\psi_{(x,y)}(j,d) = \begin{cases} 1 & \text{if } \exists\, i \in \{0,\dots,n-1\} : \vec{x}^j[i\,.\,i+d-1] = y[i\,.\,i+d-1] \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

When the values of x and y are clear from the context, for simplicity we will use the symbol ψ instead of $\psi_{(x,y)}$. Using this convention, the phase oracle P_ψ operates so as to achieve the transformation $P_\psi|j\rangle|d\rangle = (-1)^{\psi(j,d)}|j\rangle|d\rangle$, for all $j, d \in \{0,1\}^{\log(n)}$ [15]. In Section 4.1 we show how the phase oracle P_ψ can be effectively implemented by means of a circuit having depth $O(\log^3(n))$.

After $O(\sqrt{n})$ 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_φ implementing the function $\varphi_{(x,y)} : \{0,\dots,n-1\} \times \{0,\dots,n-1\} \times \{0,\dots,n-1\} \longrightarrow \{0,1\}$, where, for all strings x and y, and for all $0 \leqslant i, j, d < n$, we have:

$$\varphi_{(x,y)}(i,j,d) = \begin{cases} 1 & \text{if } \vec{x}^j[i\,.\,i+d-1] = y[i\,.\,i+d-1] \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Therefore, the oracle P_φ 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 give an implementation of the phase oracle P_φ in $O(\log^3(n))$ time [16].

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

Ultimately, the quantum test ends by checking whether the two substrings of length d starting 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_φ implementing the function φ defined in (4).

The whole structure of the quantum test is depicted in Fig. 11. The first two phases require both $O(\sqrt{n})$ queries to the oracles P_ψ and P_φ, respectively, while the last check requires a single query to the boolean oracle U_φ. Therefore, the quantum iterative test requires $O(\sqrt{n})$ queries.

We point out that, when a solution exists, both the search and verification phases may fail with a probability $O(1/n)$, 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 $O(\sqrt{n}\log(n))$, since the execution of the quantum iterative test requires $O(\sqrt{n})$ queries and the binary search for the length of the LCS requires $O(\log(n))$ iterations.

4. A circuit-model based implementation

In this section, we provide an actual implementation of the iterative test shown in Fig. 11 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 Fig. 14. 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. 14 are implemented

[15] We remark that in the first Grover search the variable $|j\rangle$ represents the sole search input, while the parameter $|d\rangle$ is kept fixed throughout the process and only appears as a constant input to the oracle function ψ.

[16] In the second Grover search the variable $|i\rangle$ is the actual search input, while both $|j\rangle$ and $|d\rangle$ are fixed parameters provided to the oracle function φ.

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.).

Operator	Symbol	Size	Depth B.S.	Depth G.C.
Controlled Circular Shift	ROT	$O(n\log(n))$	$O(\log(n))$	$O(\log(n))$
Shared Fixed Substring Check	SFC	$O(n\log(n))$	$O(\log^3(n))$	$O(\log^4(n))$
Fixed Prefix Matching	FPM	$O(n\log(n))$	$O(\log^3(n))$	$O(\log^4(n))$

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

Fig. 12. The evolution of the 6 registers ($|j\rangle|x\rangle|y\rangle|d\rangle|r\rangle$ and $|out\rangle$) involved in the computation of the boolean oracle U_ψ, whose circuit is depicted in Fig. 14.

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

Fig. 13. The evolution of the 7 registers ($|i\rangle|j\rangle|x\rangle|y\rangle|d\rangle|r\rangle|out\rangle$) involved in the computation of the boolean oracle U_φ, as shown in Fig. 14.

as boolean oracles rather than as phase oracles. For this reason, such oracles and the corresponding gates are denoted as U_ψ and U_φ rather than P_ψ and P_φ, respectively. However, this choice is not crucial and it does not lead to differences in the computational complexity since - as we have already observed - initializing the output register of a boolean oracle to the value $|-\rangle$ allows it to behave like a phase oracle.

Below, we provide a brief overview of how these operators are structured and refer the reader interested in the details of the constructions to the references cited above.

Let us 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 $O(\log(n))$ time. The output of the computation, for all circuits, is stored in the $|out\rangle$ register consisting of a single qubit.

4.1. Implementing the circuits for the three phases

The circuit for the search phase is depicted on the top of Fig. 14. It makes use of the additional register $|j\rangle$, 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 consists of applying Grover's search algorithm on the phase oracle, U_ψ, of the $\psi_{(x,y)}$ function, as defined in (3). The boolean oracle U_ψ 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_ψ is stored on the $|out\rangle$ register, which is initialized to $|-\rangle$ to make U_ψ to behave as a phase oracle.

In Fig. 12, we show the evolution of the 6 registers involved in the computation of the boolean oracle U_ψ, namely $|j\rangle|x\rangle|y\rangle|d\rangle|r\rangle$ and $|out\rangle$ (see also Fig. 14).

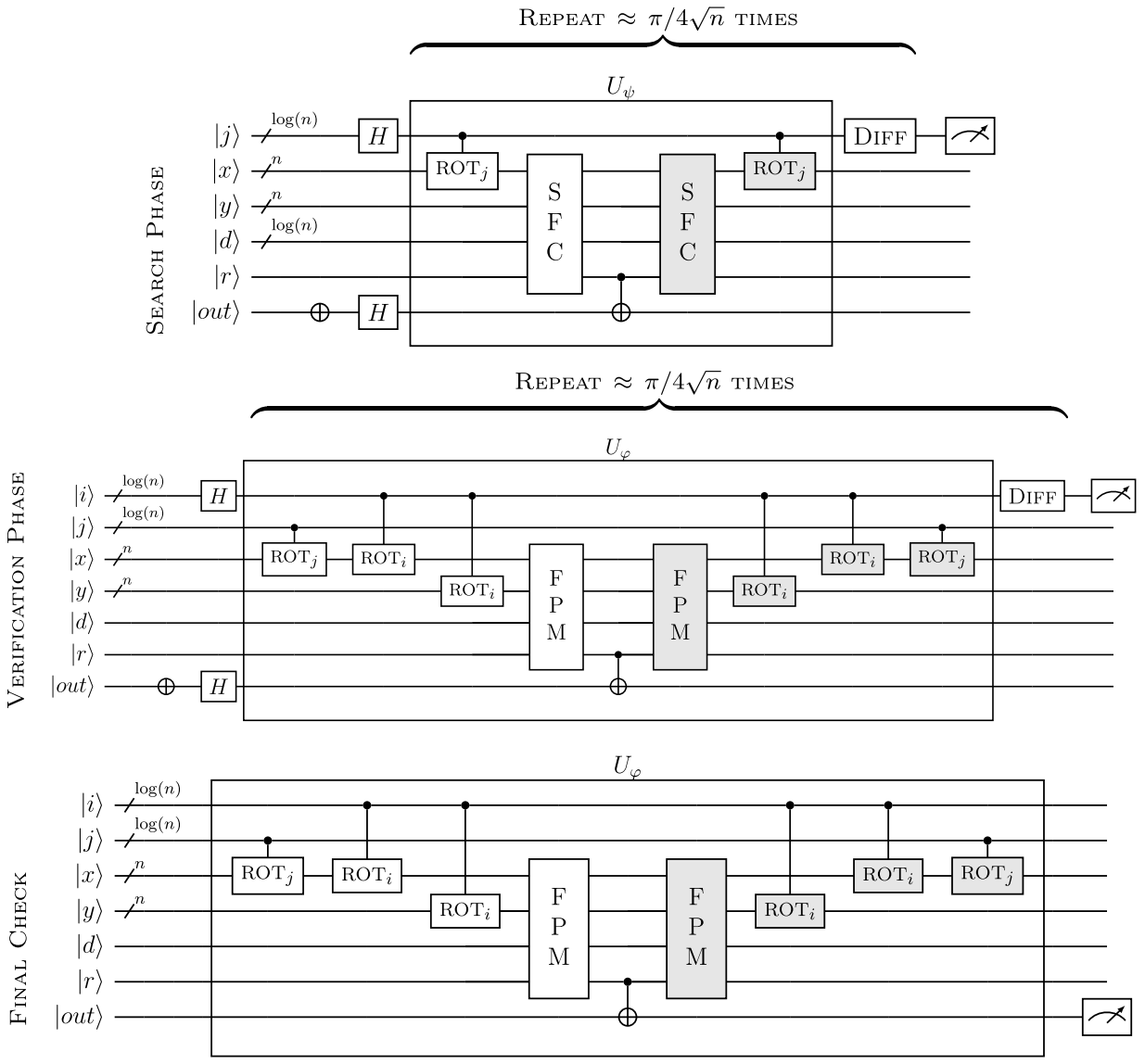

Fig. 14. The quantum circuits implementing the three phases of the iterative test of the QUANTUM-LCS algorithm. For simplicity, the circuits do not contain all the ancillæ qubits. The operators in gray color represent the inverse operators.(For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

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

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 Fig. 14, 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 Eq. (4). The quantum phase oracle, U_φ, 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_φ gets stored in the $|out\rangle$ register, a single qubit that is initially set to $|-\rangle$ in order to make U_φ to behave as a phase oracle within Grover's search algorithm.

Fig. 13 illustrates the evolution of the 7 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_φ, as depicted in Fig. 14.

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 $O(\log(n))$, as well as their inverse. The FPM operator is executed in depth $O(\log^3(n))$, while the Grover's diffuser on the register $|i\rangle$ has depth $O(\log(\log(n)))$ [17]. Since Grover's search requires $O(\sqrt{n})$ iteration, we can conclude that the depth of the circuit implementing the verification phase is equal to $O(\sqrt{n}\log^3(n))$.

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_φ. Thus, the resulting circuit implementing the final check has a depth equal to $O(\log^3(n))$.

The following theorem formalizes the overall complexity of our QUANTUM-LCS algorithm in terms of circuit depth and size.

Theorem 4.1. *Let x and y be two strings of length n over an alphabet of size σ. The QUANTUM-LCS algorithm can be implemented in the quantum circuit model using $\tilde{O}(n)$ elementary gates (i.e., circuit size $\tilde{O}(n)$) and with circuit depth*

$$\begin{cases} O(\sqrt{n}\log^4 n), & \text{if } \sigma = 2 \text{ (binary case)}; \\ O(\sqrt{n}\log^5 n), & \text{otherwise}. \end{cases}$$

Proof. The QUANTUM-LCS algorithm proceeds in three phases. In the first phase, we construct an initial superposition over all potential starting positions for substrings of a given length d. This requires $O(\sqrt{n}\log^3 n)$ time steps due to the use of the fixed-point quantum search procedure.

In the second phase, we perform a quantum check to determine whether the selected substrings of x and y match. This comparison, implemented via a quantum parallel matching subroutine, also requires $O(\sqrt{n}\log^3 n)$ time steps.

The third phase involves the classical post-processing of quantum measurement outcomes and selection of the longest successfully matched substrings. This step has depth complexity $O(\log^3 n)$.

We obtain an overall circuit depth of $O(\sqrt{n}\log^3 n)$ for the core quantum operations, which leads to a total depth of $O(\sqrt{n}\log^4 n)$ in the case of binary alphabets. For larger alphabets, the encoding and comparison procedures introduce an additional logarithmic overhead, resulting in a depth of $O(\sqrt{n}\log^5 n)$. The total circuit size remains $\tilde{O}(n)$ in both cases. □

Our analysis establishes a concrete depth and size bound for the quantum LCS problem, highlighting the potential of circuit-based models in designing efficient quantum algorithms for fundamental string problems (Fig. 15).[18]

5. Quantum longest palindromic substring

Our solution to LCS can be easily adapted to solve the LPS problem, which, given a string x of length n, is the problem of deciding the existence of a palindromic substring of length d, starting at position i. In the present section, we describe and discuss the details of such a quantum-classical hybrid algorithm whose quantum circuit formalization has the same depth complexity as the circuit presented for the LCS problem.

The main loop, based on classical computation, consists of an iterative binary search for the length of the LPS, while the quantum part consists of the iterative test. Consistent to the fact that the LPS problem is computationally simpler than the LCS one, the iterative quantum test consists of only two phases, namely the search phase and the final check.

[17] Recall that Grover's diffuser acts as the reflection $I - 2|s\rangle\langle s|$ over the uniform superposition on n elements. When applied to a register of $\log(n)$ qubits, its implementation requires multi-controlled phase inversions, which reduce to the construction of an M-CNOT gate with $\log(n)$ controls. Since such an operator can be realized in $O(\log(\log(n)))$ depth using parallelized decompositions of multi-controlled gates [32,37], the overall depth of the diffuser is bounded by this factor.

[18] It is worth noting that achieving a similar trade-off between circuit depth and size for the LCS problem in the classical Boolean model remains an open and challenging task. While linear-time sequential algorithms are well known for LCS, the design of Boolean circuits that simultaneously achieve sublinear depth and subquadratic size is, to the best of our knowledge, still elusive. This suggests that the depth bounds we obtain in the quantum circuit model could point toward a potential separation in the computational power of quantum and classical circuits for this class of problems.

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

QUANTUM TEST:
1.  i ← get a random k such that x⃗^k[0 .. d − 1] is palindromic     (SEARCH PHASE)
2.  check if x⃗^i[0 .. d − 1] is palindromic                        (FINAL CHECK)
```

Fig. 15. (Top) The pseudocode of the algorithm for computing the LPS of a string x of length n. The quantum part of the algorithm reduces to the iterative test of line 4. (Bottom) The structure of the quantum test used in the algorithm. The test consists of 2 phases, each of which is implemented as a quantum procedure.

The quantum procedures that implement the two phases use a single phase oracle U_ρ, defined on the function $\rho : \{0, \ldots, n-1\} \times \{0, \ldots, n-1\} \to \{0, 1\}$, where

$$\rho_x(i, d) = \begin{cases} 1 & \text{if } x[i \,.\, i + d - 1] \text{ is palindromic} \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, the oracle P_φ operates so as to achieve the following phase transformation, for all $i, d \in \{0, 1\}^{\log(n)}$:

$$P_\rho |i\rangle |d\rangle = (-1)^{\rho(i,d)} |i\rangle |d\rangle.$$

Specifically, the search phase is responsible for identifying a position i (if any) such that $x[i \,.\, i + d - 1]$ is palindromic. This phase is based on a Grover search that iterates $O(\sqrt{n})$ times the oracle U_ρ. Once the index i has been identified in the search phase, the final check verifies whether $x[i \,.\, i + d - 1]$ is palindromic, through a single execution of U_ρ.

The iterative test thus requires $O(\sqrt{n})$ queries to the oracle. Since the binary search requires $O(\log(n))$ iterations, the algorithm for computing the LPS requires $O(\sqrt{n}\log(n))$ oracle queries.

5.1. Quantum oracle for substring palindromicity

Determining whether a substring of a given string is a palindrome is a fundamental problem in computer science, with applications in text processing, bioinformatics, and pattern recognition. In the quantum setting, this problem can be addressed efficiently by making use of operations that can be implemented with logarithmic-depth quantum circuits. Specifically, we use the ROT quantum operator to perform cyclic rotations of a quantum register and the PFM quantum operator to compare prefixes of two strings. By combining these operations, we design a quantum circuit verifying the palindromicity of a substring through cyclic shifts and prefix matching techniques.

Formally, given a string x of length n, the goal is to determine whether the substring of length d, starting at position i, is palindromic. The proposed quantum approach follows these steps:

1. Apply a cyclic left shift of x by i positions.
2. Copy the result into a quantum register of length n, reversing the order of characters so that the register contains x (rotated by i positions) in reverse order.
3. Apply a cyclic right shift of the second register by d positions.
4. Verify whether the first d qubits of the two registers match.

The following technical lemma states that our procedure for verifying the palindromicity of a substring of x of length d works correctly.

Lemma 5.1. *Given a string x of length n, we let x' be the cyclic left shift of x by i positions, y be the reverse of x', and y' be the cyclic right shift of y by d positions. Then, the substring of x of length d starting at position i is palindromic if and only if x' and y' share a common prefix of length d.*

Proof. To prove this statement, we decompose the string x of length n into three factors such that $x = u \cdot w \cdot z$, where u is a prefix of length i, w is a substring of length d , and z is a suffix of length $n - d - i$. The goal is to determine whether w is palindromic. The sequence of steps is illustrated in Fig. 16.

After the first step, we cyclically shift x left by i positions, obtaining $x' = w \cdot z \cdot u$. In the following step x' is copied in reversed order into a new string y of length n so that $y = \bar{x'} = \bar{u} \cdot \bar{z} \cdot \bar{w}$.[19]

[19] We recall that, for any string x, the notation $\bar{x}$ denotes the *reversal* of x, that is, the string obtained by writing the characters of x in reverse order.

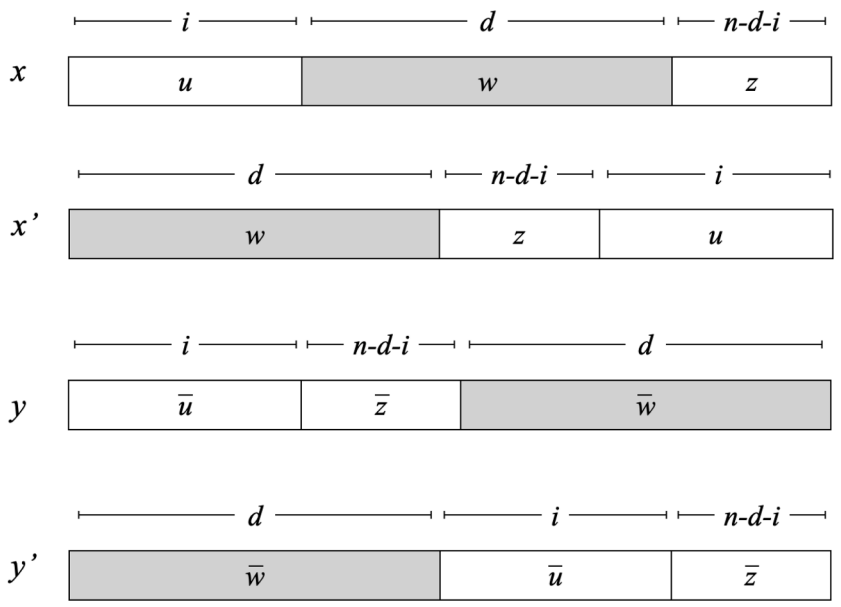

Fig. 16. Illustration of the substring palindromicity test. The circuit rotates x, copies it in reversed order, and realigns the registers so that checking the first d qubits proves whether the substring w is a palindrome.

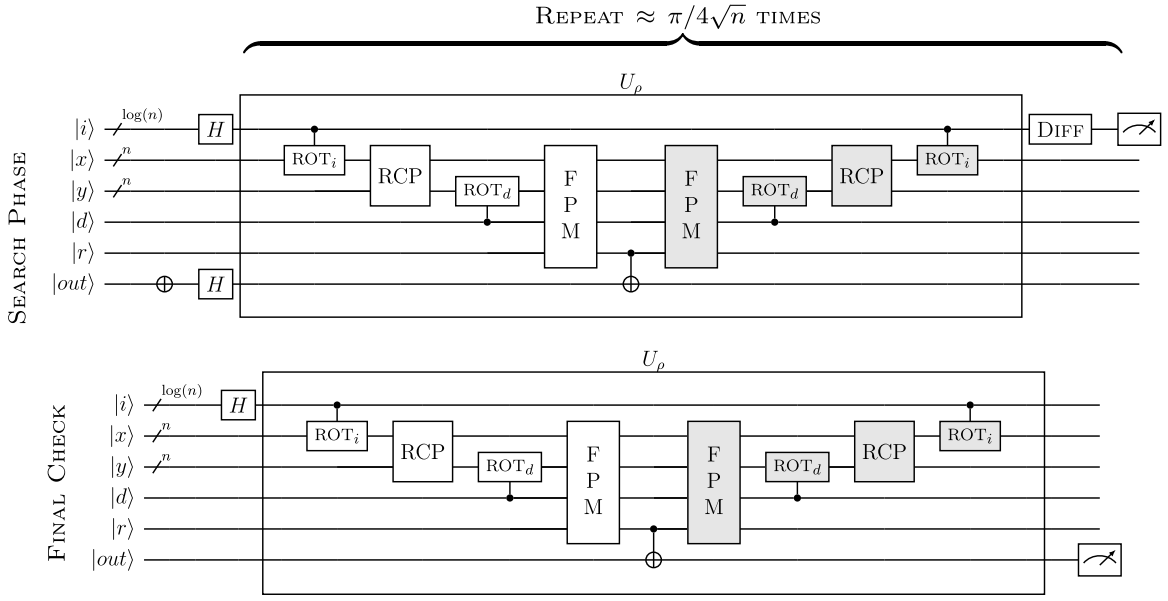

Fig. 17. The quantum circuits implementing the two phases of the iterative test of the Quantum-LPS algorithm. For simplicity, the circuits do not contain all the ancillæ qubits. The operators in gray color represent the inverse operators.

The final step consists of cyclically shifting y to the right by d positions, resulting in the string $y' = \bar{w} \cdot \bar{u} \cdot \bar{z}$.
It follows that the first d characters of y' correspond to the reverse of the prefix of x' of length d, thus proving the lemma. □

For example, consider the string $x = 11101010110$ with $n = 11$, and let $i = 3$, $d = 5$, where we underlined the substring we want to test for palindrome. We perform the following steps:

- Step 1: Left shift by 3 positions, obtaining: $x' = \underline{01010}110111$.
- Step 2: Copy to another register in reverse order: $y = 11101\underline{01010}$.
- Step 3: Right shift y by 5 positions: $y' = \underline{01010}111011$.
- Step 4: Compare the first $d = 5$ bits of x and y: 01010 vs. 01010, confirming a palindrome.

5.2. Implementing the circuits for the two phases

Based on the procedure outlined above, the quantum circuit designed for palindromic substring verification consists of two main phases: the *Search Phase* and the *Final Check*. Both phases rely on the implementation of the boolean oracle U_ρ. However, their usage differs significantly: while the Final Check operates in a deterministic manner and requires only a single execution of U_ρ, the Search Phase leverages U_ρ within Grover's algorithm for unstructured quantum search. Consequently, in this phase, U_ρ is employed to simulate the corresponding phase oracle P_ρ, enabling an efficient search process.

The circuit implementing U_ρ comprises several quantum registers that facilitate its execution. A primary register $|x\rangle$, of size n, is used to store the input string x, while an auxiliary register $|y\rangle$, of the same length, is allocated to maintain a reversed copy of x. Additionally, a register $|j\rangle$, of size $\log(n)$, is introduced to hold the rotation values of x. This register is initialized in the uniform superposition state $|+\rangle^{\log(n)}$, allowing the system to consider all possible cyclic shifts of x simultaneously. Alongside these, a register $|d\rangle$, also of size $\log(n)$, encodes the substring length d, ensuring the circuit's flexibility in processing substrings of different lengths. The circuit also includes an auxiliary register $|r\rangle$, initialized to $|0\rangle$, which temporarily stores intermediate computation results, while the final outcome is recorded in the output register $|out\rangle$, initially set to $|-\rangle$ to enforce the correct behavior of the phase oracle.

The oracle's execution follows a structured sequence of operations, each crucial to the verification process. First, a cyclic left rotation is applied to the register $|x\rangle$, controlled by the register $|j\rangle$. This transformation efficiently reorders the characters of x according to the selected shift, and its implementation can be achieved with circuit depth $O(\log(n))$. Following this, a reversal copy of $|x\rangle$ is stored into the auxiliary register $|y\rangle$, a step that ensures the availability of the reversed substring for comparison. This reversal process is executed in constant depth using a sequence of n C-NOT gates, where each gate takes the jth qubit of $|x\rangle$ as control and the $(n-j-1)$th qubit of $|y\rangle$ as target. In Fig. 17, this operation is denoted as the *Reversal Copy* (RCP) operator.

Once the reversal is completed, the next step involves applying a cyclic right rotation to the register $|y\rangle$, controlled by $|d\rangle$. This operation effectively shifts the reversed sequence, aligning it with the portion of x that needs to be checked for palindromicity. The verification itself is performed by the FPM (Prefix Matching) operator, which determines whether the first d characters of $|x\rangle$ and $|y\rangle$ match. If the two registers share a common prefix of length d, it implies that the examined substring exhibits palindromic symmetry. The result of this verification is then copied into the output register $|out\rangle$, signaling the successful identification of a palindromic substring.

To maintain quantum coherence and prevent information loss, the final step of the oracle consists of reversing all previous operations. By applying the inverse transformations in reverse order, the circuit restores the original quantum state of all registers, ensuring that no residual information affects subsequent computations. This guarantees that the oracle functions correctly within Grover's algorithm while preserving the integrity of the quantum system.

The complexity of the QUANTUM-LPS algorithm can be summarized as follows.

Theorem 3. *Let x be a string of length n over an alphabet of size σ. The QUANTUM-LPS algorithm can be implemented in the quantum circuit model with overall circuit size $\tilde{O}(n)$ and depth*

$$\begin{cases} O(\sqrt{n}\log^4 n), & \text{if } \sigma = 2; \\ O(\sqrt{n}\log^5 n), & \text{otherwise.} \end{cases}$$

Proof. The QUANTUM-LPS algorithm operates in two main phases. The first phase constructs a quantum superposition over the space of candidate palindromic substrings of a given length, centered at each possible midpoint of the input string. This search is performed using a fixed-point amplitude amplification procedure and requires $O(\sqrt{n}\log^3 n)$ time steps.

The second phase performs a quantum palindrome verification procedure by checking the symmetry of each candidate substring. This matching process can be implemented with depth $O(\log^3 n)$.

Summing up, the two phases require $O(\sqrt{n}\log^3 n)$ and $O(\log^3 n)$ time steps, respectively. Consequently, the overall depth of the quantum iterative test is $O(\sqrt{n}\log^3 n)$. This leads to an efficient implementation of the QUANTUM-LPS algorithm. Specifically, for binary strings, the algorithm achieves a circuit depth of $O(\sqrt{n}\log^4 n)$, while for general alphabets the depth increases by an additional logarithmic factor, reaching $O(\sqrt{n}\log^5 n)$. In both cases, the overall circuit size remains $\tilde{O}(n)$. □

Therefore, both the LCS and the LPS problems can be solved efficiently in the quantum circuit model, with the same asymptotic depth complexity. This aligns with the fact that the two problems have the same complexity in the classical Turing model of computation, confirming their similarity.

6. Conclusions

We have provided a concrete implementation of the first quantum algorithm for the LCS problem within the circuit model, contributing to the foundational study of depth-efficient quantum string processing. 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 circuit-level complexity of $O(\sqrt{n}\log^4(n))$, highlighting the trade-offs between time and space efficiency in quantum computation.

We note that achieving a similar depth/size trade-off in the classical Boolean circuit model remains a major open challenge. While the LCS problem has a well-known $\Theta(n)$ lower bound in the sequential setting, this does not imply a corresponding depth lower bound

for Boolean circuits, where the input is available in parallel. Nonetheless, the inherently global nature of LCS - requiring pairwise substring comparisons and propagation of match lengths - suggests that designing circuits with both sublinear depth and subquadratic size may be difficult. We are not aware of any classical construction achieving depth $\widetilde{O}(\sqrt{n})$ and size $\widetilde{O}(n)$ for LCS. Proving that such a construction is impossible would represent a breakthrough in circuit complexity and offer a formal quantum advantage in this setting. We highlight this as an important and intriguing direction for future research.

On the classical side, we stress that translating work-optimal, polylog-time PRAM/word-RAM algorithms for suffix-tree-based pipelines into uniform bounded-fan-in Boolean circuits with quasi-linear size and polylogarithmic depth is nontrivial and, to the best of our knowledge, not covered by existing references under standard alphabet assumptions. Establishing such circuits would be an interesting classical contribution in its own right. Our quantum result provides an explicit circuit construction with proven depth $\tilde{O}(\sqrt{n})$ and well-accounted size, in a directly comparable circuit model.

We are aware that the circuit-based model adopted in this work, like the oracle-based model, relies on assumptions that are not currently realizable on existing quantum hardware. Nevertheless, we believe that such idealizations are legitimate within theoretical computer science, where the primary goal is to understand computational limits and algorithmic structure. In this spirit, our work offers a depth-efficient solution within a fully specified model, which may serve as a foundation for future developments as quantum technologies mature.

While both the circuit-based and oracle-based models involve idealized assumptions, the long-term trajectory of quantum hardware development may favor the former. In particular, scaling up the number of available qubits appears to be a more feasible engineering goal than maintaining deep quantum coherence or large-scale entanglement across many qubits, as required by query-based access models such as QRAM. This trend reinforces the relevance of circuit-level algorithmic analysis under explicit resource assumptions.

Thus, our results not only advance the state of quantum algorithms for LCS and LPS but also underscore the broader implications for circuit-based quantum computing, where resource optimization is essential. Future work will continue to explore these trade-offs and further refine quantum algorithms for practical deployment on quantum hardware.

CRediT authorship contribution statement

Domenico Cantone: Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization; **Simone Faro:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization; **Arianna Pavone:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization; **Caterina Viola:** Writing – review & editing, Writing – original draft, Validation, Supervision, Investigation, Formal analysis, Conceptualization.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] F.L. Gall, S. Seddighin, Quantum meets fine-grained complexity: sublinear time quantum algorithms for string problems, Algorithmica 85 (5) (2023) 1251–1286. https://doi.org/10.1007/s00453-022-01066-z
[2] S. Akmal, C. Jin, Near-optimal quantum algorithms for string problems, Algorithmica 85 (8) (2023) 2260–2317. https://doi.org/10.1007/S00453-022-01092-X
[3] D. Cantone, S. Faro, A. Pavone, C. Viola, Quantum circuit based longest common substring, in: M.P. Ugo de'Liguoro, L. Roversi (Eds.), Proceedings of the 25th Italian Conference on Theoretical Computer Science (ICTCS), 3811 of *CEUR Workshop Proceedings*, IC-EATCS, CEUR, 2024, pp. 1–15. https://ceur-ws.org/Vol-3811/paper090.pdf.
[4] P.W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Comp. 26 (5) (1997) 1484–1509. https://doi.org/10.1137/s0097539795293172
[5] 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. https://doi.org/10.1145/237814.237866
[6] F. Arute, K. Arya et. al, Quantum supremacy using a programmable superconducting processor, Nature 574 (2019) 505–510. https://api.semanticscholar.org/CorpusID:204836822.
[7] H. Ramesh, V. Vinay, String matching in $\tilde{O}(\sqrt{n}+\sqrt{m})$ quantum time, J. Discrete Algoritms 1 (1) (2003) 103–110. https://doi.org/10.1016/S1570-8667(03)00010-8
[8] P. Niroula, Y. Nam, A quantum algorithm for string matching, npj Quantum Inf. 7 (Article number 37) (2021). https://doi.org/10.1038/s41534-021-00369-3
[9] 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, 13960 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 117–133. https://doi.org/10.1007/978-3-031-38100-3_9
[10] S. Faro, A. Pavone, C. Viola, Quantum path parallelism: a circuit-Based approach to text searching, in: Theory and Applications of Models of Computation: 18th Annual Conference, TAMC 2024, Hong Kong, China, May 13-15, 2024, Proceedings, Springer-Verlag, Berlin, Heidelberg, 2024, p. 247-259. https://doi.org/10.1007/978-981-97-2340-9_21
[11] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, The MIT Press, 2nd edition, 2001.
[12] 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, 5, 1994, pp. 35–44. https://doi.org/10.1109/HICSS.1994.323593
[13] A. Montanaro, Quantum pattern matching fast on average, Algorithmica 77 (1) (2017) 16–39. https://doi.org/10.1007/s00453-015-0060-4
[14] A. Ambainis, Quantum walk algorithm for element distinctness, SIAM J. Comput. 37 (1) (2007) 210–239. https://doi.org/10.1137/S0097539705447311
[15] 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, 305 of *Contemporary Mathematics*, American Mathematical Society, 2002, pp. 53–74. https://doi.org/10.1090/conm/305/05215
[16] F. Magniez, A. Nayak, J. Roland, M. Santha, Search via quantum walk, SIAM J. Comput. 40 (1) (2011) 142–164. https://doi.org/10.1137/090745854

[17] 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. https://doi.org/10.1145/3313276.3316368

[18] 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, 2676 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 55–69. https://doi.org/10.1007/3-540-44888-8_5

[19] 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. https://doi.org/10.1145/3564246.3585153

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

[21] E. Bernstein, U.V. Vazirani, Quantum complexity theory, SIAM J. Comput. 26 (5) (1997) 1411–1473. https://doi.org/10.1137/S0097539796300921

[22] 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. https://doi.org/10.1109/SFCS.1993.366852

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

[24] V. Giovannetti, S. Lloyd, L. Maccone, Quantum random access memory, Phys. Rev. Lett. 100 (16) (2008). https://doi.org/10.1103/physrevlett.100.160501

[25] V. Giovannetti, S. Lloyd, L. Maccone, Architectures for a quantum random access memory, Phys. Rev. A 78 (5) (2008). https://doi.org/10.1103/physreva.78.052310

[26] D.K. Park, F. Petruccione, J.-K.K. Rhee, Circuit-based quantum random access memory for classical data, Sci. Rep. 9 (1) (2019). https://doi.org/10.1038/s41598-019-40439-3

[27] D. Cantone, S. Faro, A. Pavone, C. Viola, Quantum circuits for fixed matching substring problems, in: K. Arai (Ed.), Intelligent Computing, Springer Nature Switzerland, Cham, 2024, pp. 667–686.

[28] N.D. Mermin, Quantum Computer Science: An Introduction, Cambridge University Press, USA, USA, 2007.

[29] D. Deutsch, Quantum computational networks, Proc. R. Soc. London A Math. Phys. Sci. 425 (1989) 73–90. https://api.semanticscholar.org/CorpusID:123073680.

[30] W.K. Wootters, W.K. Wootters, W.H. Zurek, A single quantum cannot be cloned, Nature 299 (1982) 802–803. https://api.semanticscholar.org/CorpusID:4339227.

[31] C.M. Dawson, M.A. Nielsen, The Solovay-Kitaev algorithm, Quantum Info. Comput. 6 (1) (2006) 81-95.

[32] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin, H. Weinfurter, Elementary gates for quantum computation, Phys. Rev. A 52 (5) (1995) 3457–3467. https://doi.org/10.1103/physreva.52.3457

[33] M.A. Nielsen, I.L. Chuang, Quantum Computation and Quantum Information (10th Anniversary edition), Cambridge University Press, 2016. https://www.cambridge.org/de/academic/subjects/physics/quantum-physics-quantum-information-and-quantum-computation/quantum-computation-and-quantum-information-10th-anniversary-edition?format=HB.

[34] P. Høyer, R. Spalek, Quantum fan-out is powerful, Theory C. 1 (2005) 81–103.

[35] M. Fang, S. Fenner, F. Green, S. Homer, Y. Zhang, Quantum lower bounds for fanout, Quantum Info. Comput. 6 (1) (2006) 46–57.

[36] T. Toffoli, Reversible computing, in: J.W. de Bakker, J. van Leeuwen (Eds.), Automata, Languages and Programming, 85 of *LNCS*, Springer, 1980, pp. 632–644. https://doi.org/10.1007/3-540-10003-2_104

[37] Y. He, M.-X. Luo, E. Zhang, H.-K. Wang, X.-F. Wang, Decompositions of n-qubit Toffoli gates with linear circuit complexity, Int. J. Theor. Phys. 56 (2017) 2350–2361. https://doi.org/10.1007/s10773-017-3389-4

[38] S. Balauca, A. Arusoaie, Efficient constructions for simulating multi controlled quantum gates, in: Computational Science - ICCS 2022, 13353 of *LNCS*, Springer, 2022, pp. 179–194. https://doi.org/10.1007/978-3-031-08760-8_16

[39] S.E. Rasmussen, K. Groenland, R. Gerritsma, K. Schoutens, N.T. Zinner, Single-step implementation of high-fidelity n-bit Toffoli gates, Phys. Rev. A 101 (2020) 022308. https://doi.org/10.1103/PhysRevA.101.022308

[40] A. Broadbent, E. Kashefi, Parallelizing quantum circuits, Theor. Comput. Sci. 410 (26) (2009) 2489–2510. https://doi.org/10.1016/J.TCS.2008.12.046

[41] A. Borodin, On relating time and space to size and depth, SIAM J. Comput. 6 (4) (1977) 733–744. https://doi.org/10.1137/0206054

[42] A. Borodin, S. Cook, N. Pippenger, Parallel computation for well-endowed rings and space-bounded probabilistic machines, Inf. Contr. 58 (1) (1983) 113–136. https://doi.org/10.1016/S0019-9958(83)80060-6

[43] J. Watrous, Quantum Computational Complexity, Springer New York, New York, NY, 2009, pp. 7174–7201. https://doi.org/10.1007/978-0-387-30440-3_428

[44] C. Moore, M. Nilsson, Parallel quantum computation and quantum codes, SIAM J. Comput. 31 (3) (2001) 799–815. https://doi.org/10.1137/S0097539799355053

[45] L.K. Grover, Trade-offs in the quantum search algorithm, Phys. Rev. A 66 (2002) 052314. https://doi.org/10.1103/PhysRevA.66.052314

[46] R. Beals, H. Buhrman, R. Cleve, M. Mosca, R. de Wolf, Quantum lower bounds by polynomials, J. ACM 48 (4) (2001) 778–797. https://doi.org/10.1145/502090.502097

[47] M. Boyer, G. Brassard, P. Høyer, A. Tapp, Tight bounds on quantum searching, Fortschr. Phys. 46 (4-5) (1998) 493–505. https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291521-3978\protect\protect\leavevmode@ifvmode\kern+.1667em\relax%28199806\protect\protect\leavevmode@ifvmode\kern+.1667em\relax%2946\protect\protect\leavevmode@ifvmode\kern+.1667em\relax%3A4/5\protect\protect\leavevmode@ifvmode\kern+.1667em\relax%3C493\protect\protect\leavevmode@ifvmode\kern+.1667em\relax%3A%3AAID-PROP493\protect\protect\leavevmode@ifvmode\kern+.1667em\relax%3E3.0.CO%3B2-P.

[48] A. Ambainis, Quantum search algorithms, SIGACT News 35 (2) (2004) 22–35. https://doi.org/10.1145/992287.992296

[49] T.J. Yoder, G.H. Low, I.L. Chuang, Fixed-point quantum search with an optimal number of queries, Phys. Rev. Lett. 113 (2014) 210501. https://doi.org/10.1103/PhysRevLett.113.210501

[50] S. Faro, A. Pavone, C. Viola, Families of constant-depth quantum circuits for rotations and permutations, in: M.P. Ugo de'Liguoro, L. Roversi (Eds.), Proceedings of the 25th Italian Conference on Theoretical Computer Science (ICTCS), 3811 of *CEUR Workshop Proceedings*, IC-EATCS, CEUR, 2024, pp. 29–41. https://ceur-ws.org/Vol-3811/paper270.pdf.