Theoretical Computer Science 1080 (2026) 116059

Contents lists available at ScienceDirect

Theoretical Computer Science

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

A general quantum circuit for string matching: Unleashing quantum path parallelism ⋆

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

Section Editor: Lila Kari
Handling editor: Katie Harris

2000 MSC:
68W32
68Q12

Keywords:
Quantum computing
Text processing
Sequence analysis

ABSTRACT

Text searching is a fundamental problem in computer science, and it finds applications in several scientific fields. The *string matching problem* is the common framework for the class of text searching problems where the objective is to find all, possibly approximate, occurrences of a given pattern of length m within a larger text of length n. This paper introduces the *quantum path parallelism* approach, that is, a general strategy based on quantum computation, which is easily adapted to a variety of nonstandard text-searching problems. Our method translates a text-searching problem into an automata-based string-recognition problem, associating each possible approximation of the pattern with a different path accepted by the automaton. Under favorable conditions, our algorithm solves an approximate search problem in $O(\sqrt{nm}\log(n))$-time, yielding a quadratic speed-up over classical solutions. In other cases, such a speed-up is achieved only for short patterns, i.e. whenever $m = O(\log(n))$. We show the flexibility of our method, giving explicit adaptations to some specific approximate string matching problems.

1. Introduction

Quantum computing is a branch of computer science focused on developing computing systems that leverage the properties of quantum information. Classical computers process the information encoded into binary bits (each of which represents either a 0 or a 1), while the fundamental units of quantum information are *quantum bits* (qubits), which can exist in a *superposition* of multiple states. Two or more qubits can also be *entangled*, that is, correlated in a non-classical way. These properties allow quantum computers to perform certain tasks faster and more efficiently than classical computers.

This advantage is particularly evident in algorithms such as Shor's algorithm [1], which factors large numbers exponentially faster than classical methods, and Grover's algorithm [2], which speeds up unstructured searches quadratically. Recent demonstrations of quantum supremacy [3] - where quantum computers outperform classical ones in specific tasks - have fueled growing interest in quantum computing, driving its application across various areas of computer science.

⋆ An extended abstract of the present article appeared with title *Quantum Path Parallelism: A Circuit-Based Approach to Text Searching* as part of the Proceedings of the 2024 Annual Conference on Theory and Applications of Models of Computation (TAMC).

* Corresponding author.
E-mail addresses: 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.2026.116059
Received 30 March 2025; Received in revised form 2 April 2026; Accepted 20 May 2026
Available online 25 May 2026

Text processing and string problems, in particular, are among the areas that have recently gained the attention of quantum computing researchers. Text processing and string matching play a crucial role in computer science, forming the foundation of tasks such as search engines, data mining, natural language processing, DNA sequencing, social media analysis, compiler design, antivirus software development, among others. Over the years, innovations in tackling these problems have resulted in the development of foundational techniques like dynamic programming, hashing algorithms, automata-based algorithms and suffix trees. It is worth noting that in recent times, quantum solutions have emerged as a promising avenue for addressing these problems with the potential for significant speed-up. For example, efficient solutions have recently been given for exact string matching [4–7], approximate string matching [7], string comparison [8], edit distance [9], longest common substring [10,11], and longest palindrome substring [10].

The object of this paper is the *string matching* problem and its non-standard variants. Formally, given a pattern x of length m and a text y of length n, the string matching problem may consist in (1) verifying if x occurs in y or in (2) finding all occurrences of x in y and their corresponding positions. In the case of approximate matching, we are also interested in (3) identifying the pattern configurations for each of its occurrences in the text.

In the classical model of computation, the exact matching problem has an $\Omega(n)$-time complexity, while the non-standard case can have different solutions depending on the allowed approximations. The use of finite-state automata has often been at the basis of original and efficient solutions [12]. In this direction, the simulation of nondeterministic finite-state automata using bit-parallelism [13] represents a fundamental breakthrough in text searching. By exploiting bitwise operations in the word RAM model, this approach achieves a reduction in time complexity proportional to the machine word size.

The first quantum solution to exact string matching, due to Ramesh and Vinay [4], combines Grover search [2] with a parallel string matching technique. This algorithm is formalized in the quantum query model [14] and is designed to solve problem (1) making $\tilde{O}(\sqrt{n})$ queries. Therefore, problem (2) is solved in $\tilde{O}(\sqrt{\rho n})$ time,[1] where ρ is the number of occurrences of x in y, providing real quantum advantage when $\rho = o(n)$. For the average case, Montanaro [5] proved a super-polynomial separation between the quantum and the classical complexity of the problem. The first solution in the quantum circuit model [15] with a $\tilde{\mathcal{O}}(\sqrt{n})$-time complexity for (1) was presented by Niroula and Nam in 2021 [6], and recently extended [7] to swap matching with a $\tilde{\mathcal{O}}(\sqrt{\rho n 2^m})$ complexity for (3).

We generalize the basic model presented in [6] [2] and introduce the *quantum path parallelism* (QPP) approach, a general strategy based on quantum computation that can be easily adapted to a variety of nonstandard text searching problems. Our method translates the text searching problem to an automata-based string recognition problem, associating each possible approximation of the pattern with a different accepted path in the automaton. Defining an *association rule* that identifies each path of the automaton with a binary string allows for using quantum computation to test, in parallel, the presence of all possible approximations of the pattern for each text position. This results in a quantum algorithm that solves the text searching problem (3) in $O(\sqrt{\rho n 2^\omega}\log(n))$-time, where ω is a value strictly dependent on m and at least equal to the number of strings accepted by the automaton. Under favorable conditions, namely when $\omega = O(\log(m))$, the time complexity reduces to $O(\sqrt{nm}\log(n))$ for (1) and to $O(\sqrt{\rho nm}\log(n))$ for (3). In the case of short patterns, where one can assume $m = O(\log(n))$, the complexity reduces to the quasi-linear $O(n\log(n))$ for (1).

We observe that the efficiency of many existing quantum algorithms is measured in the *query complexity* model [4,5]. In such a scenario, the input is presented as a *black box* that can be accessed by an *oracle* that, given a function f, returns the image of the input (or other variables depending on it) via f. The query complexity of an algorithm is defined as the number of *queries* that it makes to the oracle(s).

While the query model is intriguing and abstract enough to be valuable for purely theoretical approaches, it holds limited significance when it comes to designing algorithms intended for practical implementation on real hardware, since it is often unclear how one could implement a specific phase oracle efficiently. On the other hand, there are different quantum models that easily and almost directly translate to concrete implementations on quantum computers. Perhaps, one of the most widespread among such models is the *quantum circuit model* [15], 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. In this context, a quantum circuit can be thought of as a Boolean circuit in which the basic gates represent elementary quantum operations, and the computational complexity of a quantum circuit is measured by its *depth*.

For the reasons discussed above, we designed our algorithms in the circuit model of computation, providing actual implementations of quantum circuits for solving text-searching problems.

Regarding quantum access to input strings, many solutions based on the quantum query model [4,10] assume the availability of a Quantum Random Access Memory (QRAM) [16], which allows any input string s to be accessed by a random oracle at unit cost. However, the most efficient QRAM designs [17] 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 require $O(\log^2(n))$ time. However, our algorithm does not rely on a random access oracle, and we assume that the input strings are available in quantum registers accessible by the circuit and which do not require any initialization.[3]

[1] Whenever the number of solutions is $\rho > 1$, and we would like to find all of them, $\Theta(\sqrt{\rho n})$ iterations of Grover's search are sufficient and necessary.

[2] In fact, the depth of our circuit is a logarithmic factor smaller than the one presented in [6], since we employ a constant-depth cyclic shifting operator

[3] Under this idealised assumption, all qubits can be initialised in parallel, resulting in $O(1)$ depth for input preparation. In practice, however, constructing and compiling state-preparation circuits from classical data typically requires $\Omega(n)$ time due to limitations of current compilation tools and hardware interfaces. These practical constraints do not affect our asymptotic depth estimates, whose purpose is to characterise the intrinsic complexity of the QPP framework independently of technological considerations.

Table 1
Some non-standard text searching problems, and their classical and QPP time complexity. Here, n and m denote the lengths of the input strings, α is the approximation bound, and ρ is the number of occurrences of x in y. Problem (1) refers to the decision problem of detecting the existence of an occurrence, while Problem (3) requires reporting all occurrences together with their corresponding configurations.

Problem	Classical Solution (1)	QPP Solution (1)	QPP Solution (3)
exact	$O(n)$	$O(\sqrt{n}\log(n))$	$O(\sqrt{\rho n}\log^2(n))$
cyclic rotations	$O(n)$	$O(\sqrt{nm}\log(n))$	$O(\sqrt{\rho nm}\log(n))$
swaps (unbounded)	$O(n\log m\log\sigma)$	$O(\sqrt{n2^m}\log(n))$	$O(\sqrt{\rho n2^m}\log(n))$
swaps (α-bounded)	$O(n\sqrt{\alpha\log(m)})$	$O(\sqrt{nm^\alpha}\log(n))$	$O(\sqrt{\rho nm^\alpha}\log(n))$
gaps (unbounded)	$O(nm)$	$O(m\sqrt{n2^m}\log^2(n))$	$O(\sqrt{\rho n2^m}\log^2(n))$
gaps (α-bounded)	$O(nm)$	$O(\alpha\sqrt{n2^m}\log^2(n))$	$O(\sqrt{\rho n2^m}\log^2(n))$

As a demonstration of the flexibility of our method, we show how it can be adapted for solving some approximate text searching problems obtaining a significant quantum advantage. Specifically, we solved the string matching problems allowing for cyclic rotations, swaps, and an approximate version of swaps called α-bounded .[4] Table 1 summarizes the time complexities obtained in this paper through the QPP approach in comparison with their respective classic solutions, which are generally designed to solve only problem (1). Problem (3) can be solved at the additional cost of $O(\rho m)$.

Although in the worst case (i.e. when $\rho = n$) the quantum advantage is obtained only under suitable conditions, in the average case, when ρ tends to be a constant, the quantum advantage is obtained in many cases. Furthermore, many non-standard text searching techniques, that are highly efficient in practical implementations, and especially those implementing bit-parallelism, are based on filtering approaches, searching for a constant-length substring of the pattern (usually $m \leqslant 32$ is used). Under such conditions, ω is a constant and our approach requires $\tilde{\mathcal{O}}(\sqrt{n})$-time on average, yielding a real quantum advantage in all cases. However, such an analysis of average cases goes beyond the purpose of this paper.

The organization of the article is as follows. In Section 2, we introduce the basics of the quantum circuit model of computation together with some fundamental constructions that we need in our algorithm. Section 3 contains the general algorithm implementing the Quantum Path Parallelism, which is the core result of this article. Section 4 is devoted to the application of the aforementioned general algorithm to some specific variants of the string matching problems. Finally, in Section 5, we draw our conclusion and discuss future research directions arising from the present work.

2. Basics of quantum circuits

The fundamental unit in quantum computation is the *qubit* (or *quantum bit*). A qubit is a coherent superposition of the two orthonormal computational basis states, which are denoted by $|0\rangle$ and $|1\rangle$, using the conventional *braket* notation. Formally, a single qubit is an element from the *state space* $\mathcal{H}$, that is the two-dimensional Hilbert space on the complex numbers equipped with the inner product; therefore, the mathematical expression of a qubit $|\psi\rangle$ is a linear combination of the two basis states, i.e. $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where the values α and β, called *amplitudes*, are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$ and are related to the probability of measuring the qubit in the state $|0\rangle$ or $|1\rangle$, respectively. A *quantum measurement* is the only operation that can access the information contained in a qubit; however, this operation causes the qubit to collapse to one of the two basis states.

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 of them 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 qubits taken together 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 the tensor product of n state spaces, $\mathcal{H}^{\otimes n}$, and thus it is expressed as a linear combination of the 2^n states in $\{0,1\}^n$, that is $|\psi\rangle = \sum_{k=0}^{2^n-1} \alpha_k|k\rangle$, where the values α_k represent the probability of measuring the register in the state $|k\rangle$, and $\otimes$ denotes the tensor product. If k is an integer value that can be represented by a binary string of length n, we use the symbol $|k\rangle$ to denote the register of n qubits such that $|k\rangle = \bigotimes_{i=0}^{n-1} |k_i\rangle$, where $|k_i\rangle$ takes the value of the ith least significant binary digit of k. 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 $|q\rangle$. A *phase oracle* U_f for a function $f : \{0,1\}^n \to \{0,1\}$ takes as input a quantum register $|x\rangle$ of size n and leaves its value unchanged, applying to it a negative global phase only if x is a solution for the function, that is $f(x) = 1$. Formally, $U_f|x\rangle = (-1)^{f(x)}|x\rangle$.

2.1. The quantum circuit model

The model of computation we adopt in this paper is that of *quantum circuits*. A quantum circuit can be represented as a sequence of parallel wires (each corresponding to a qubit) passing through certain *gates* that perform quantum elementary operations on them. Indeed, a quantum circuit needs to be reversible and, therefore, for each gate, the number of input wires equals that of output wires.

[4] Our method can also be applied to solve the text searching problem with gaps.

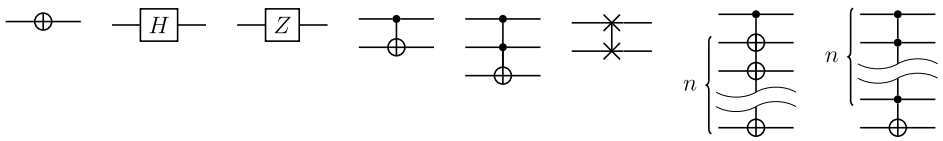

Fig. 1. The representation of the following basic gates (from left to right): Pauli-X, Hadamard, Pauli-Z, CNOT, Toffoli, Swap, Fanout and Multicontrolled NOT.

Furthermore, because of the No-Cloning Theorem [18], it is not possible to either copy or split (fan-out) the information carried by any wire. To use the information on a quantum state multiple times, it is necessary to include some *ancillæ*.[5]

There are two major measures of the computational complexity in general circuit models of computation. One of them is the total number of basic gates, namely the *size*; the other one is the *depth* of the direct acyclic graph that represents the circuit. The latter is usually the measure of election for the complexity of quantum circuits because, in such circuits, it is possible to execute two or more gates in parallel whenever they operate on disjoint sets of qubits. The depth of a quantum circuit coincides with the number of time steps performed before the output, that is, with its computational time.[6]

For the sake of transparency, we remark that our depth bounds are stated within the ideal quantum circuit model, which assumes unrestricted parallel execution of gates acting on disjoint qubits. Actual quantum hardware may impose additional constraints due to limited parallel control lines or restricted qubit connectivity. These practical considerations do not affect the asymptotic analysis developed here, whose purpose is to characterise the intrinsic depth of the QPP framework independently of a specific architecture.

The space complexity of a quantum circuit is the number of qubits that it involves, which can also be pictured as the number of wires.

There is a great variety of elementary quantum operators. We briefly list some of them, those that are used in this paper. To define such basic operations, we use the linearity of quantum maps. These elementary gates can be implemented in constant time by real quantum machines and, by definition, have depth $\Theta(1)$ in the quantum circuit model.

- The *Pauli*-X or *NOT* (X) gate is the quantum equivalent of the Boolean NOT gate. It operates on a single qubit, mapping $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$.
- The *Hadamard* gate (H) acts on a single qubit, mapping $|0\rangle$ and $|1\rangle$ to $|+\rangle$ and to $|-\rangle$.
- The *Pauli*-Z or *phase-flip gate* (Z) maps $|0\rangle$ to itself and $|1\rangle$ to $-|1\rangle$, by applying a negative phase to it. Based on the equivalence $Z = HXH$, the Z operator can be obtained from the previous two operators.
- The *controlled* NOT *gate* (CNOT) operates on a register of two qubits $|q_0, q_1\rangle$. If the control qubit $|q_0\rangle$ is $|1\rangle$, an X operator is applied on the qubit $|q_1\rangle$, otherwise both qubits stay unchanged. Formally, it maps $|q_0, q_1\rangle$ to $|q_0, q_0 \oplus q_1\rangle$, where $\oplus$ is the Boolean exclusive or, defined also as the sum $\mod 2$.
- The *Swap* gate (SWAP) is a two-qubit operator: it swaps the state of the two qubits $|q_0, q_1\rangle$ involved in the operation, mapping them to $|q_1, q_0\rangle$. Interestingly, the swap gate can be achieved by the application of three CNOT operators.
- The *Toffoli* gate (CCNOT) operates on 3 qubits: if the first two qubits are both equal to $|1\rangle$, an X operator is applied on the third qubit, otherwise all qubits are unchanged. Formally, $|q_0, q_1, q_2\rangle$ is mapped to $|q_0, q_1, q_0 q_1 \oplus q_2\rangle$.

Fig. 1 contains a graphical representation of the listed basic gates, together with the graphical representation of the general fanout and the general multicontrolled operators that we are going to define.

We introduce, now, three general constructions of non-elementary gates that we need for our algorithm, analyzing their depth.

Fanout Gates. Given a quantum register $|\psi\rangle = |q_0, q_1, \ldots, q_{n-1}\rangle$ of size n, a *fanout* operator simultaneously copies the control qubit $|q_0\rangle$ onto the $n-1$ target qubits $|q_i\rangle$, for $i \in \{1, \ldots, n-1\}$. Formally, the fanout operator maps $|q_0, q_1, q_2, \ldots, q_{n-1}\rangle$ to the register $|q_0, q_0 \oplus q_1, q_0 \oplus q_2, \ldots, q_0 \oplus q_{n-1}\rangle$. Although a fanout can be implemented using n CNOT gates arranged sequentially, thus achieving linear time, the no-cloning theorem prevents qubits from being fanned out in constant circuit depth [19]. However, assuming that the target qubits are all initialized to $|0\rangle$, it is easy to fan the information in the control qubit out in $\Theta(\log(n))$ depth, by a divide-and-conquer strategy employing controlled-not gates and $\log(n)$ ancilla qubits [20], all initialized to $|0\rangle$.

Multicontrolled Gates. A *multicontrolled operator* applies an elementary operation on the unique target if *all* the control qubits are set to $|1\rangle$. For example, the n-ary *multicontrolled CNOT* operator, which is a generalization of the Toffoli operator, maps a register $|q_0, q_1, q_2, \ldots, q_{n-1}\rangle$ into the register $|q_0, q_1, \ldots, q_{n-2}, (q_0 \cdot q_1 \cdots q_{n-2}) \oplus q_{n-1}\rangle$. Similarly, the n-ary *multicontrolled phase-flip* operator maps a register $|q_0, q_1, q_2, \ldots, q_{n-1}\rangle$ into the register $|q_0, q_1, \ldots, q_{n-2}, (-1)^{q_0 \cdot q_1 \cdots q_{n-2}} q_{n-1}\rangle$.

[5] Ancilla qubits or ancillæare additional qubits needed for the computation. Usually they are initialized to the classic 0 state, and after the information they store has been used they are usually cleaned-up, i.e. reset to 0.

[6] Throughout this work we adopt the standard convention of the quantum circuit model that *any number of gates acting on disjoint sets of qubits can be executed in parallel*. Consequently, the depth of a circuit reflects the number of sequential layers of gates, independently of how many gates appear in each layer. This assumption is used, for instance, in the analysis of the cyclic shifting gate, whose implementation consists of $O(n)$ disjoint SWAP operations that can be applied in parallel.

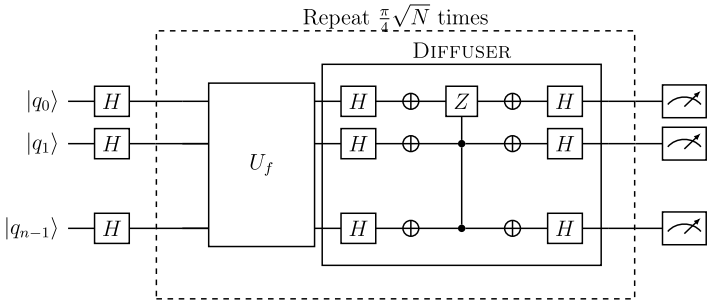

Fig. 2. The circuit implementing Grover's algorithm.

It is possible to obtain a logarithmic-depth quantum circuit performing a multicontrolled operator [21], by exploiting parallelism and using $n/2$ ancilla qubits. We also mention a recent result [22] that enables the implementation of multi-controlled NOT gates in constant time in architectures with trapped ions and neutral atoms.

Single-control gates. Assume the need to execute a certain number, say t, of parallel gates controlled by the same qubit $|c\rangle$. Since the control qubit would be involved in all the t operations, the resulting gates would no longer be executable in parallel. Let G be an operator that consists in the parallel execution of the mutually disjoint gates $\{G_0, G_1, \ldots, G_{t-1}\}$, each operating on disjoint sets of qubits from a register of global size n in time $T(n)$. Let $|q_0, q_1, \ldots, q_{n-1}\rangle$ be the quantum register to which one wants to apply the t gates in parallel. The state of $|c\rangle$ is transferred into t ancilla qubits $|a_i\rangle$, all initialized to $|0\rangle$, for $0 \leqslant i < t$, through a fan-out operator. In this way, t copies of the state $|c\rangle$ are obtained. The ancilla qubit $|a_i\rangle$ is then used as a control qubit for applying the gate G_i. Finally, the ancilla qubits are cleaned up by a new fan-out operator, controlled again by $|c\rangle$. Such a technique applies the t gates in parallel in $T(n)$ time, but requires $O(\log(t))$ time for the fanout operator.

2.2. Grover's search algorithm

Grover's search algorithm [2] is one of the most famous results proving theoretical quantum supremacy, together with Shor's algorithm [1]. Grover's algorithm searches a desired item within an unstructured dataset of n items in $O(\sqrt{n})$ time, 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 with less than n queries that solves the problem with certainty for an arbitrary dataset [23].

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 $|+\rangle^{\oplus n}$. 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. 2 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 $O\left(\sqrt{n}(T(n) + \log(n))\right)$,[7] where $T(n)$ is the depth of the phase oracle gate.

In the case where the function f has ρ solutions, with $\rho > 1$, $O(\sqrt{n})$ iterations are still enough to find a solution, but it can be shown that roughly $\frac{\pi}{4}\sqrt{\frac{n}{\rho}}$ iterations are required [24], and that this bound is optimal [25]. However, in general, the value ρ is not known a priori and can only be obtained through a complex procedure based on the Quantum Phase Estimation. Finally, if we have ρ solutions and we would like to find all of them, $\Theta(\sqrt{n\rho})$ iterations are sufficient and necessary [26].

[7] If we assume to be able to implement the multicontrolled Z gate in constant time [22], a Grover's search on a dataset of n items achieves $O(\sqrt{nT(n)})$ time.

Algorithm 1: Algorithm constructing a quantum circuit for the leftward rotation of a register of size n by s positions.

Input: n, s
Initialize n input wires (qubits) $|q_0\rangle, \dots, |q_{n-1}\rangle$;
for $i = 1$ *to* $\lceil \frac{n}{2} \rceil - 1$ **do**
    Swap $|q_i, q_{n-i}\rangle$
end
for $j = 0$ *to* $\lceil \frac{n}{2} \rceil - 1$ **do**
    Swap $|q_{n-\lceil \frac{s}{2} \rceil + j}, q_{n-\lfloor \frac{s}{2} \rfloor - j}\rangle$
end

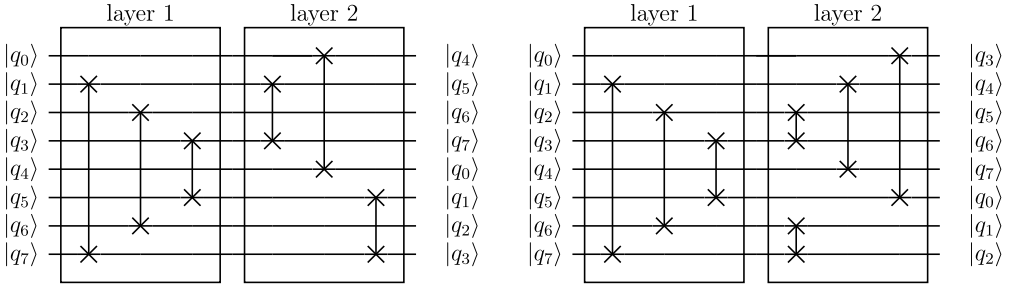

Fig. 3. The circuits taking in input a register of 8 qubits and operating a leftward rotation of 4 and 3 positions, respectively.

2.3. The controlled cyclic shifting gate

A *cyclic shifting* gate (or *rotation* gate) R_s applies a leftward cyclic shift of s positions. Formally, the operator R_s applies the following permutation

$$R_s(|q_0, q_1, \dots, q_{n-1}\rangle) = |q_{n-s}, q_{n-s+1}, \dots, q_{n-1}, q_0, \dots, q_{n-s-1}\rangle.$$

A cyclic shifting gate operating on a n-dimensional register can be implemented by a quantum circuit of logarithmic depth [27, 28], such an implementation is used in [6]. However, a cyclic shifting gate can always be obtained in constant depth [29,30], employing two layers of parallel Swap gates. Algorithm 1 contains the pseudocode of the algorithm [30] that on input (n, s) outputs the description of a quantum circuit circularly rotating a quantum register $|q\rangle$ of size n by s positions. Observe that, in this context, qubits are indexed modulo n. Fig. 3 shows the circuits rotating a quantum register of size 8 by 4 and 3 positions, respectively. Such a circuit has depth $O(1)$ and at each time-layer performs at most $\frac{n}{2}$ parallel swaps.

Next, we want to define a *controlled cyclic shifting gate* that rotates a quantum register by a number of positions that depends on an input value k. In this context, we assume to 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

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

Because the controlled cyclic shifting gate does not change the control register, sometimes 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 $|k\rangle$ register can be implemented using $\lceil \log(n) \rceil$ qubits. We have then $|j\rangle = \bigotimes_{i=0}^{\lceil \log(n) \rceil - 1} |k_i\rangle$, where $|j_i\rangle$ is initialized with the value of the ith least significant bit of the binary representation of the value k. It will then be enough to apply to register $|q\rangle$ the rotation operator R_{2^i} controlled by the qubit $|j_i\rangle$, for any value of i, such that $0 \leqslant i < \lceil \log(n) \rceil$.

Observe that the rotation operator R_{2^i} consists of a sequence of at most 2 time-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 2.1 that involves the use of $\frac{n}{2}$ 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, which we can actually clean and reuse for each of the two parallel blocks of gate R_{2^i}. Since we want to apply all the controlled R_{2^i} in parallel, overall we will need $\frac{n}{2} \log(n)$ ancillæ.[8] Therefore, as explained at the end of Section 2.1, the controlled cyclic shift by 2^i positions requires $O(\log(n))$ time-steps to fan the information on the control qubit out to the ancillæ plus $O(1)$ 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 $O(\log(n))$. Fig. 4 contains an illustration of such a circuit.

[8] Alternatively, we can clean and reuse the $\frac{n}{2}$ ancillæfor every R_{2^i}, trading time for space and achieving a circuit depth of $O(\log^2(n))$.

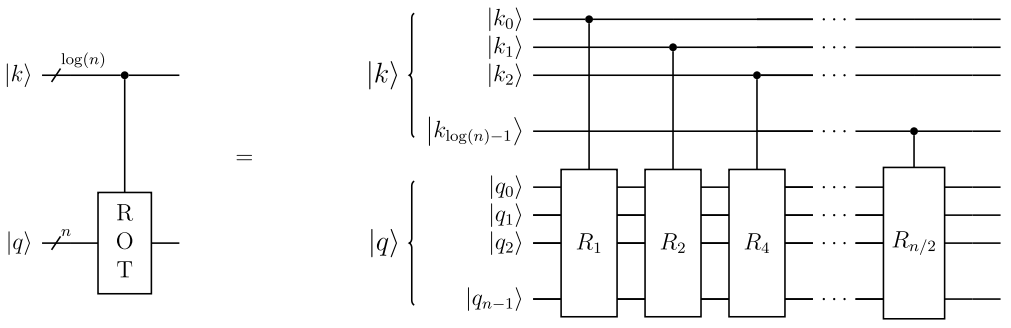

Fig. 4. A circular shift operator on a n-qubit register $|q\rangle$ controlled by a $\log(n)$-qubit register $|k\rangle$. The circuit makes use of $\frac{n}{2}\log(n)$ ancilla qubits, which are not shown in this graphical representation. On the left of the figure, we show the succinct graphical representation used below in this paper.

3. Quantum path parallelism

Nonstandard text searching concerns string matching problems where a substring of the text matches the pattern, based on general relationships between the corresponding symbols. Such a relationship can be manifold. To fit all the variants of the text searching problem in a single general framework, we assume the existence of a metric function for assessing how similar two strings are.

A *string metric* (or *string distance*) is a function $d : \Sigma^* \times \Sigma^* \to \mathbb{N}$ that measures the distance (or the *inverse similarity*) between two strings for approximate string matching. Given two strings $x, y \in \Sigma^*$, we generally assume that $d(x, y) = 0$ if they are equal, while $d(x, y) > 0$ in all other cases. Many formulations of the approximate string matching problem provide an upper bound, $k \geqslant 0$, so that x matches y if $d(x, y) \leqslant k$.

Given a string $x \in \Sigma^m$, a distance function d and a bound k, the *string matching automaton* associated with x, d and k, is a deterministic acyclic finite state automaton (DFA), denoted by the symbol $\mathcal{A}_{(x,d,k)}$, that recognizes the language $\mathcal{L} = \{y \in \Sigma^* \ : \ d(x, y) \leqslant k\}$. Formally, the string matching DFA $\mathcal{A}_{(x,d,k)}$ is defined by the tuple $\mathcal{A} \equiv \{Q, \Sigma, I, F, \delta\}$, where Q is the set of states of the automaton, Σ is the set of input symbols, $I \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $\delta : Q \times \Sigma \to Q$ is the transition function. We will also make use of the generalized transition function $\delta^* : \Sigma^* \to Q$, which is recursively defined by $\delta^*(x.c) = \delta(\delta^*(x), c)$, for any $x \in \Sigma^*$ and $c \in \Sigma$, and $\delta^*(\epsilon) = I$, for the base case. Along this paper, when the values of x, d, and k are clearly defined by context, we use the symbol $\mathcal{A}$ in place of $\mathcal{A}_{(x,d,k)}$. By definition, if the pattern occurrence is detected at position j of the text then we have that $d(y[j..j + r - 1], x) \leqslant k$, where the value r, which is the length of the *window*, i.e. the portion of the text recognized as an occurrence of the pattern, can vary within a limited range of values dependent on the function d and the bound k. For example, in exact string matching, as well as in swap string matching, we have $r = m$ for all possible occurrences. Differently, in the approximate string matching problem allowing for the presence of at most α gaps we have $m - \alpha \leqslant r \leqslant m$. We use the symbol $\mathcal{R}$ to indicate the set of all allowable lengths for any occurrence of the pattern in the text.

Since a quantum register of dimension $\log(n)$, like any binary sequence of the same dimension, can take on all values between 0 and $n - 1$, for simplicity we will assume that the text y has length $n = 2^p$, for some $p > 0$. We also assume that the text ends with a special character \$ not belonging to the alphabet Σ. These two assumptions can be made without 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 symbol \$. Thus, the resulting string has at most twice the length of the original string.

We also note that any occurrence of length r can begin at any position j of the text, for $0 \leqslant j \leqslant n - r$. However, in this paper we also relax this condition by admitting values of j between 0 and $n - 1$ and assuming that a substring of the text can be obtained by shifting it cyclically. This assumption can also be made without loss of generality since the last character of the text is always the special character \$ that does not occur within the pattern.

Based on the assumptions above, the text searching problem, defined as an approach based on the use of the string matching DFA, can be defined as follows.

Definition 3.1 (Text-Searching Based on a String Matching DFA)**.** Let y be a text of length n and let x be a pattern of length m. Moreover, let $d : \Sigma^* \times \Sigma^* \to N$ be a distance function, let $k \geqslant 0$ be an approximation bound and let $\mathcal{R}$ be the set of all possible lengths of any occurrence of x in y. Finally, let $\mathcal{A} \equiv \{Q, \Sigma, I, F, \delta\}$ be the string matching DFA constructed on x. The text searching problem consists in finding all positions at which a substring of y with length $r \in \mathcal{R}$ accepted by $\mathcal{A}$ begins. More formally, we want to find the elements of the set $\Gamma(x, y)^{(\mathcal{R})} = \bigcup_{r \in \mathcal{R}} \Gamma(x, y)^{(r)}$, where

$$\Gamma(x, y)^{(r)} = \{0 \leqslant j < n \ : \ \delta^*(y[j..j + r - 1]) \in F\}. \tag{1}$$

3.1. The ideas behind the general algorithm

A DFA can be seen, to some extent, as a generalization to acyclic graphs. In this context, graph nodes correspond to automaton states, and an edge labelled by c between two nodes, q_i and q_j, exists if $\delta(q_i, c) = q_j$. It is therefore possible to extend, in a natural way, the concept of a path to a DFA. The following definition underlies the idea on which the path parallelism approach is based.

Definition 3.2 (Complete Automaton Path)**.** Let $\mathcal{A} \equiv \{Q, \Sigma, I, F, \delta\}$ be a string matching DFA , defined for a string x, and let d and k be a distance function and a bound , respectively. We say that a path in the automaton is complete if it starts at the initial state I and ends at a final state. More formally, a complete automaton path of length r is a sequence of states $\langle q_0, q_1, \dots, q_{r-1}\rangle$, where $q_j \in Q$ for $0 \leqslant j < r$, $q_0 = I$, $q_{r-1} \in F$ and $\delta(q_j, c) = q_{j+1}$ for a symbol $c \in \Sigma$ and $0 \leqslant j < r-1$.

The symbol $\text{PATH}(\mathcal{A})$ denotes the set of all possible complete paths in $\mathcal{A}$. Any complete automaton path $p \in \text{PATH}(\mathcal{A})$ naturally induces a sequence of symbols associated with it, namely the *path labeling* of p. It roughly consists of the sequence of symbols that label the transitions in p. More formally, we get the following definition.

Definition 3.3 (Path Labeling)**.** Let $\mathcal{A} \equiv \{Q, \Sigma, I, F, \delta\}$ be a string matching DFA, the path labeling of a path $p = \langle q_0, q_1, \dots, q_{r-1}\rangle$ of length r, with $p \in \text{PATH}(\mathcal{A})$, is the sequence $x_p = \langle c_0, c_1, \dots, c_{r-2}\rangle$, where $c_i \in \Sigma$ and $\delta(q_i, c_i) = q_{i+1}$, for $0 \leqslant j < r-1$.

Although the QPP approach can be adapted to handle most variants of nonstandard text searching, this paper focuses on the cases where a path labeling is a permutation of the sequence of characters in x, or at any rate in a subset of such sequence. We denote the path labeling of a path p by the symbol x_p to emphasize the fact that x_p is a sequence of symbols obtained from x by applying the modifications induced by p.

By Definitions 3.2 and 3.3, and because we are interested in the configuration of the occurrences, Eq. (1) can be rewritten as

$$\Gamma(x, y)^{(r)} = \{(j, p) \ : \ 0 \leqslant j < n \ \text{ and } p \in \text{PATH}(\mathcal{A}) \text{ and } y[j..j + r - 1] = x_p\}. \tag{2}$$

Each complete path p of $\mathcal{A}$ is associated with a *path code*, that is, a binary sequence π that uniquely identifies it within the set $\text{PATH}(\mathcal{A})$.

Definition 3.4 (Path Codes Association Rule)**.** Let $\mathcal{A}$ be a string matching DFA, and ω a constant value. The path codes association rule defines a two-way association between the elements of the set $\text{PATH}(\mathcal{A})$ and a subset $\Pi_{\mathcal{A}} \subseteq \{0,1\}^{\omega}$ such that each complete path of $\mathcal{A}$ is uniquely identified by a code of length ω.

We say that a binary sequence π of length ω is a *valid path code* if it is associated with a complete path in $\mathcal{A}$, i.e. if $\pi \in \Pi_{\mathcal{A}}$, where $\Pi_{\mathcal{A}}$ (or simply Π, if $\mathcal{A}$ is clear by context) denotes the set of all valid path codes of the automaton. If we denote by π_p the path code associated with p, we have $\Pi_{\mathcal{A}} = \{\pi_p \ : \ p \in \text{PATH}(\mathcal{A})\}$. Observe that $\Pi_{\mathcal{A}} \subseteq \{0,1\}^{\omega}$, and since all path codes share the same length ω, we need $\omega \geqslant \log_2(|\text{PATH}(\mathcal{A})|)$.

Since there is a one-to-one correspondence between the sets $\Pi_{\mathcal{A}}$ and $\text{PATH}(A)$, in what follows we will interchangeably use the notations x_p and x_π to denote the same path labeling, which is induced by p, whenever π is the path code of p.

The implementation of the QPP approach requires the following definitions of a transformation function and a test function.

Definition 3.5 (Transformation Function)**.** Let $\mathcal{A}$ be a string matching DFA defined for a string x of length m, a distance function d and a bound k, and let Π the set of all valid path codes defined by a given association rule. The transformation function, for any occurrence of length $r \in \mathcal{R}$, is a function $t : \Sigma^m \times \Pi \to \Sigma^r$, such that $t(x, \pi) = x_\pi$ for every $\pi \in \Pi$.

Roughly speaking, such a function maps the sequence of characters in x into the sequence of characters in x_π, applying the transformations induced by the path p associated with π.

Definition 3.6 (Test Function)**.** Let $\mathcal{A}$ be a string matching DFA defined for a string x, a distance function d and a bound k, and let Π be the set of all valid path codes defined by a given association rule. The test function is a map $\upsilon : \{0,1\}^{\omega} \to \{0,1\}$, such that, for any $\pi \in \{0,1\}^{\omega}$, $\upsilon(\pi) = 1$ if, and only if, $\pi \in \Pi_{\mathcal{A}}$.

In other words the test function checks whether a path code $\pi \in \{0,1\}^{\omega}$ is valid. The definitions introduced allows us to rewrite Eq. (2) as follows.

$$\Gamma(x, y)^{(r)} = \{(j, \pi) \ : \ 0 \leqslant j < n, \ \pi \in \{0,1\}^{\omega}, \ y[j..j + r - 1] = x_\pi \text{ and } \upsilon(\pi) = 1\}. \tag{3}$$

3.2. The general quantum algorithm

In this section, we describe how Eq. (3) translates into a quantum algorithm. Fig. 5 provides a simplified diagram of the circuit that implements the QPP algorithm.

The circuit consists of the initialization, the preprocessing and matching phases. The latters, with the subsequent corresponding uncomputing phases, implement the phase oracle of the function γ.

The register $|j\rangle$, of size $\log(n)$, is used to hold the value of any position j where a substring of the text may begin. The register starts with the value $|0\rangle^{\log(n)}$ and is then initialized to $|+\rangle^{\log(n)}$ through the application of $\log(n)$ Hadamard's gates. At the end of the initialization phase the register $|j\rangle$ contains the superposition of all possible values in $\{0, \dots, n-1\}$. Formally, $|j\rangle = \frac{1}{\sqrt{n}} \sum_{i \in \{0,1\}^{\log(n)}} |i\rangle$.

The register $|\pi\rangle$ has size ω and retains the path codes of the DFA constructed for the pattern x. Starting from $|0\rangle^{\omega}$, the register $|\pi\rangle$ is initialized to $|+\rangle^{\omega}$ and, at the end of the initialization phase, it contains the superposition of all possible values in $\{0, \dots, 2^{\omega} - 1\}$. Formally, $|\pi\rangle = \frac{1}{\sqrt{2^{\omega}}} \sum_{i \in \{0,1\}^{\omega}} |i\rangle$. Registers $|a\rangle$, $|b\rangle$, and $|c\rangle$ are single qubits and store the output of specific computations. While $|a\rangle$ and $|c\rangle$ are initialized to $|0\rangle$, $|b\rangle$ is initialized to $|-\rangle$. The register $|x\rangle$ has size $m \log(\sigma)$ and can be viewed as a sequence of m registers x_i (for $0 \leqslant i < m$) of size $\log(\sigma)$, where $\sigma = |\Sigma|$. Each $|x_i\rangle$ is initialized with the $\log(\sigma)$ bits of the binary representation of the ith symbol of the pattern.

Similarly, the register $|y\rangle$ has size $n \log(\sigma)$ and can be viewed as a sequence of n registers y_i of size $\log(\sigma)$, such that $|y_i\rangle$ is initialized with the $\log(\sigma)$ bits of the binary representation of the ith character of the text, for $0 \leqslant i < n$. For visualization purposes, in Fig. 5, the

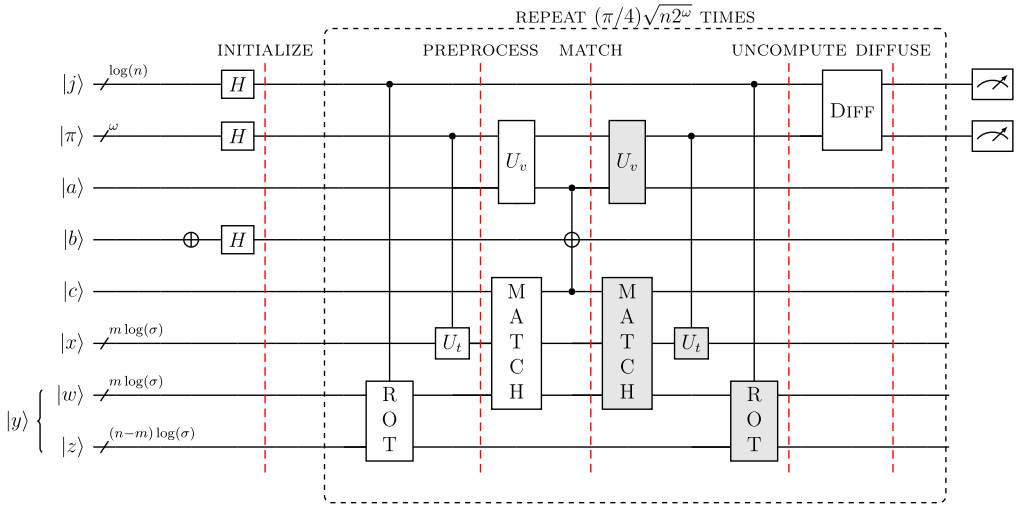

Fig. 5. A simplified diagram describing the circuit of the general algorithm based on the QPP approach for non-standard text searching.

$|y\rangle$ register is divided into two registers, $|w\rangle$ and $|z\rangle$, of size $m\log(\sigma)$ and $(n-m)\log(\sigma)$, respectively. Each of the operators applied during this phase takes constant time, and since they can be applied in parallel, the initialization phase takes $O(1)$ time.

The preprocessing phase prepares the registers $|x\rangle$ and $|y\rangle$ for the upcoming matching phase. The first step is to apply a cyclic shifting operator (see Section 2.3) controlled by the register $|j\rangle$ to the register $|y\rangle$. Such an operator is identified as ROT in Fig. 5) . Since $|j\rangle$ is initialized to the superposition of all possible values between 0 and $n-1$, at the end of the application of the ROT gate, the register $|y\rangle$ contains the superposition of all possible circular rotations of the text. Since the register $|w\rangle$ contains the first r symbols of $|y\rangle$, at the end of preprocessing it will contain the superposition of all circular substrings of the text with length r. As we explained in Section 2.3, such an operator can be implemented in depth $O(\log(n))$ using only $\frac{\log(n)}{2}$ ancillæ.

During the preprocessing phase, we also apply the operator U_t, which is the boolean oracle implementing the transformation function $t : \Sigma^m \times \Pi \to \Sigma^r$. Note that U_t is controlled by the register $|\pi\rangle$, representing the path code associated with a specific path of the automaton. For all intents and purposes, the path code $|\pi\rangle$ governs the transformations to be applied to the sequence of characters in $|x\rangle$, depending on the type of approximation problem being considered. Formally, the boolean oracle U_t applies the transformation $U_t|\pi, x\rangle = |\pi, x_\pi\rangle$. Thus, after the application of U_t the register $|x\rangle$ will contain the superposition of the strings x_π, for $\pi \in \{0,1\}^\omega$.

After the preprocessing phase, the registers $|y\rangle$, $|w\rangle$, and $|x\rangle$ are as follows

$$|y\rangle = \frac{1}{\sqrt{n}}\sum_{j=0}^{n-1}\left[\bigotimes_{i=0}^{n-1}|y_{(j+i)}\rangle\right], \quad |w\rangle = \frac{1}{\sqrt{n}}\sum_{j=0}^{n-1}\left[\bigotimes_{i=0}^{r-1}|y_{(j+i)}\rangle\right], \quad |x\rangle = \frac{1}{\sqrt{2^{\omega}}}\sum_{\pi\in\{0,1\}^{\omega}}|x_\pi\rangle,$$

where the sums $(j+i)$ are computed cyclically, that is, modulo n.[9] If $D_t(\omega, m)$ is the depth of the circuit U_t, then the time complexity of the preprocessing phase is $O\big(\max(\log(n), D_t(\omega, m))\big)$.

Assuming w_j refers to the string w shifted by j positions, the matching phase consists of the joint application of two operators allowing to check if the string w_j (for $0 \leqslant j < n$) matches the string x_π, for some $\pi \in \Pi$. The first operator applied in this phase is the boolean oracle U_v that implements the test function v. Such a boolean oracle checks whether a given string $\pi \in \{0,1\}^\omega$ is a valid path code. Formally, for any $\pi \in \{0,1\}^\omega$, we have $U_v|\pi, 0\rangle = |\pi, 1\rangle$ if $\pi \in \Pi$, and $U_v|\pi, 0\rangle = |\pi, 0\rangle$ otherwise. We denote by $D_v(\omega)$ the depth of the circuit implementing the boolean oracle U_v.

The second operator (denoted by MATCH in Fig. 5) compares the two strings w_j and x_π in parallel, for all values $0 \leqslant j < n$ and $0 \leqslant \pi < 2^\omega$. Although the two strings are compared characterwise, the comparisons between the characters can be performed in parallel. More precisely, for $0 \leqslant i < r$, the characters $(w_j)_i$ and $(x_\pi)_i$ are compared in $O(\log(\sigma))$ time by means of $\log(\sigma)$ parallel CNOT gates and a multiple CNOT gate with $\log(\sigma)$ controls. Specifically, once the r characters of the two strings have been compared, the final comparison of the two strings can be performed in $O(\log(r))$ time by means of a multiple CNOT gate with r controls. Thus, since $r \in O(m)$, the overall time complexity of the MATCH operator is $O(\log(\sigma) + \log(m))$. After the matching phase we have $\text{MATCH}|0, x_\pi, w_j\rangle = |1, x_\pi, w_j\rangle$ if $x_\pi = w_j$, and $\text{MATCH}|0, x_\pi, w_j\rangle = |0, x_\pi, w_j\rangle$ otherwise. The overall time complexity of the matching phase is $O\big(\max(\log(\sigma) + \log(m), D_v(\omega))\big)$.

The outputs of U_v and MATCH are stored into the qubits $|a\rangle$ and $|c\rangle$, respectively, and are finally combined by a CCNOT gate whose output is stored into the register $|b\rangle$. Since the preprocessing and matching phases must operate jointly within the phase oracle of the γ function, the output register $|b\rangle$ is initialized to $|-\rangle$, so that at the end of the phase we have

[9] The equation above should be interpreted over the full joint state, since the registers $|y\rangle$, $|w\rangle$, and $|x\rangle$ are entangled with $|j\rangle$ and $|\pi\rangle$, thus forbiding a tensor-product decomposition.

$$|j\rangle \otimes |\pi\rangle \otimes |b\rangle = \left[\frac{1}{\sqrt{n2^\omega}} \sum_{\substack{0 \leqslant \pi < 2^\omega \\ 0 \leqslant j < n}} (-1)^{\gamma(x_\pi, w_j)} (|j\rangle \otimes |\pi\rangle) \right] \otimes |-\rangle.$$

The operators applied in the preprocessing and matching phases implement the boolean oracle of the function γ. The fact that the register $|b\rangle$, holding the result of the computation at the end of the process, is initialized to $|-\rangle$ causes the computation to result in a phase oracle of the function γ. The computation ends with an *uncompute* process obtained by applying the inverses of the operators used in the previous two phases.

The search for a solution of the function γ is implemented through an application of Grover's algorithm. Multiple iterations of the entire computation process, followed by the application of the Grover's Diffuser to registers $|k\rangle$ and $|\pi\rangle$, may then be required.

Theorem 3.7 (Correctness of QPP)**.** *Let $S = \{(j, \pi) : \gamma(j, \pi) = 1\}$ be the set of all valid matches. One iteration of the QPP oracle implements a phase oracle U_γ satisfying*

$$U_\gamma |j, \pi\rangle = (-1)^{\gamma(j,\pi)} |j, \pi\rangle .$$

Moreover, after $O\left(\sqrt{(n2^\omega)/|S|}\right)$ Grover iterations, the final measurement of the register $|j\rangle|\pi\rangle$ returns an element of S with probability at least $2/3$.

Proof. The proof proceeds by analysing each stage of the circuit—preparation, preprocessing, evaluation, and uncomputation—showing that together they implement the phase oracle required by Grover's algorithm.

(1) Preparation. The initial state is

$$|\Psi_0\rangle = \left(\tfrac{1}{\sqrt{n}} \sum_{j=0}^{n-1} |j\rangle \right) \left(\tfrac{1}{\sqrt{2^\omega}} \sum_{\pi \in \{0,1\}^\omega} |\pi\rangle \right) |x\rangle|y\rangle|0\rangle_a|-\rangle_b|0\rangle_c.$$

Thus the algorithm starts in the uniform superposition of all (j, π).

(2) Action of the preprocessing stage. Controlled by $|j\rangle$, the ROT gate rewrites the text register into the superposition of all windows $y[j..j + r - 1]$. Controlled by $|\pi\rangle$, the oracle U_t transforms the pattern register so that the content of $|x\rangle$ becomes $t(x, \pi)$. No amplitude is lost or duplicated and the transformation is unitary.

(3) Logical evaluation of the predicate γ. The test oracle U_v writes $v(\pi)$ into $|a\rangle$. The matching oracle MATCH compares the transformed pattern $t(x, \pi)$ with the window of the text starting at position j and writes the result into $|c\rangle$. Hence, immediately before the controlled-phase operation on $|b\rangle$, the joint content of $|a\rangle|c\rangle$ equals

$$|a, c\rangle = \begin{cases} |1, 1\rangle & \text{iff } \gamma(j, \pi) = 1, \\ |*\rangle & \text{otherwise.} \end{cases}$$

(4) Phase-kickback. Since $|b\rangle = |-\rangle$, a Toffoli gate controlled by $|a\rangle|c\rangle$ flips the phase of $|b\rangle$ if and only if $\gamma(j, \pi) = 1$. Thus the global operation on the $|j, \pi\rangle$ register is exactly the phase oracle U_γ.

(5) Uncomputation. All preprocessing and matching operations are reversed. Therefore all ancilla qubits return to their initial state and no residual entanglement remains with the registers $|j\rangle|\pi\rangle$. The state after the oracle call is therefore

$$|\Psi_1\rangle = \frac{1}{\sqrt{n2^\omega}} \sum_{j,\pi} (-1)^{\gamma(j,\pi)} |j, \pi\rangle|x\rangle|y\rangle|0\rangle_a|-\rangle_b|0\rangle_c.$$

(6) Grover convergence. The diffuser on the $|j\rangle|\pi\rangle$ register acts as a reflection about the average amplitude. By the standard two-dimensional analysis of Grover's algorithm, after $O(\sqrt{(n2^\omega)/|S|})$ applications of the QPP oracle followed by the diffuser, the state is rotated to within constant angle of the subspace spanned by $\{|j, \pi\rangle : (j, \pi) \in S\}$, yielding success probability at least $2/3$. □

The proof applies verbatim to any instantiation of the QPP framework, including all automata-based variants discussed in Section 4. The following result on the complexity of the QPP algorithm holds.

Theorem 3.8. *The QPP algorithm, for searching a pattern of length m on a text of length n, both strings over an alphabet Σ of size σ, and with a path code size equal to ω, requires $O(\sqrt{n2^\omega}(\max(\log(n), D_t(w, m)) + \max(\log(m) + \log(\sigma), D_v(w)) + \log(\log(n) + \omega)))$-time.*

With reference to Theorem 3.8, the assumption that can be made most reliably is that the dominant term within the second multiplicative factor is $\log(n)$. This reduces, in most cases, the complexity to $O(\sqrt{n2^\omega}\log(n))$. In some favorable cases, we are allowed to use a set of path codes of size $\omega = O(\log(m))$, reducing the complexity to the sub-linear $O(\sqrt{nm}\log(n))$ time. Alternatively, under the assumption that $\omega = O(\log(n))$ (i.e. for short patterns) the running time of the algorithm reduces to the quasi-linear $O(n\log(n))$ time.

For completeness, we also report asymptotic gate counts for the main components of the QPP oracle. The controlled cyclic shifting gate ROT consists of $O(n)$ SWAP operations arranged in constant depth, yielding a total gate count $O(n)$. The transformation oracle

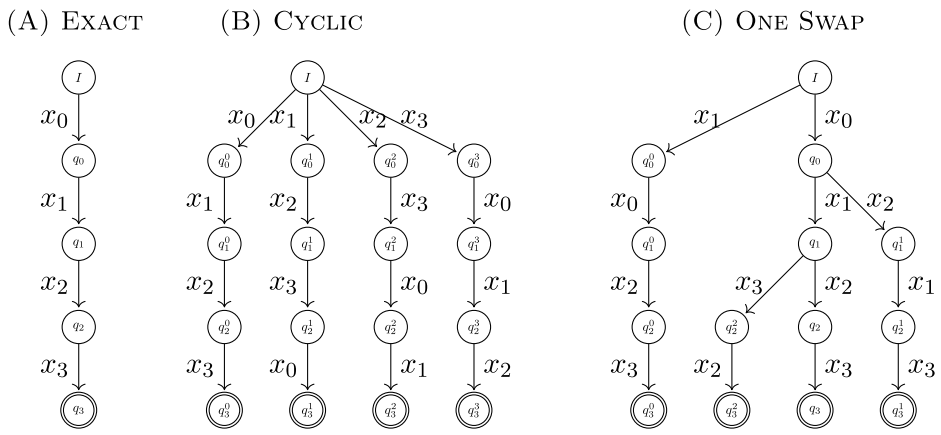

Fig. 6. The representation of the string matching DFAs for solving 4 text searching problems, built on the pattern $x_0x_1x_2x_3$. From left to right we show the DFAs for: exact string matching, string matching with rotations, and string matching with swaps.

U_t and the test oracle U_υ have gate counts $o(m^3)$ and $O(\omega)$, respectively, in all instantiations considered in Section 4. The matching operator MATCH compares m symbols in parallel and therefore contributes $O(m)$ gates. Summing the contributions, one iteration of the QPP oracle uses $O(n + m^3 + \omega)$ gates in total, while maintaining circuit depth $O(\max(\log n, D_t, D_\upsilon))$. Thus, depth provides a lower bound on execution time on hardware with full parallelism, whereas the gate count provides a corresponding upper bound on hardware with limited parallel control.

4. Solving some nonstandard text searching problems

In this section, we adapt the general algorithm presented in Section 3 to solve three nonstandard string matching problems, and specifically the string matching with cyclic rotations, with (bounded) swaps and with (bounded) gaps.

4.1. The string matching problem allowing for cyclic rotations

In the *cyclic rotation string matching* problem [31,32] (or string matching allowing for rotations) the pattern is a cyclic string, meaning that it can occur within the text in its original form or after having undergone a cyclic rotation. Cyclic strings have been the subject of extensive research in a variety of computer science and mathematical fields, with an emphasis on combinatorics. These strings possess a fundamental property: they do not have distinct initial or terminal positions, but the two ends of the string are connected, like a necklace of letters. Furthermore, a cyclic string can be viewed as a conventional linear string where the first and the last letters are wrapped and joined together. In this context, the three strings *accba*, *cbaac*, and *baacc* represent the same cyclic string, and it has three occurrences in the text *baccbaacbbc*. In the standard model of computation the problem can be solved in $O(n)$ time [31].

A cyclic string of length m has exactly m different configurations, thus the automaton for detecting all cyclic rotations of a pattern x can be obtained by triggering, at the initial state of the automaton, m distinct parallel paths such that the ith path is labeled with the sequence of characters obtained from the cyclic rotation of the pattern of s positions to the left, for $0 \leqslant s < m$. As a consequence, the initial portions of two paths, as well as their final portions, can be joined if the corresponding labeling share a prefix or, respectively, a suffix. See Fig. 6 (B) for the cyclic automaton built on a pattern with 4 distinct characters.

In this context, a path code can be defined as a binary sequence of dimension $\omega = \log(m)$ containing the binary representation of the amplitude s of the cyclic rotation applied to the pattern for $0 \leqslant s < m$. The path labeling associated with a path code is that obtained from the cyclic rotation of the pattern of s positions, and the corresponding transformation function t, takes a binary sequence $\pi \in \{0,1\}^{\log(m)}$ and maps a string x of length m to its cyclic rotation of s positions to the left, where $s = \sum_{i=0}^{\log(m)} \pi_i 2^i$. The boolean oracle U_t of the transformation function t can be implemented by the cyclic rotation operator introduced in Section 2.3, while the test function υ can be dropped since any path code π consisting of $\log(m)$ bits represents a valid rotation amount between $0 \leqslant s < m$, that is, it can be implemented as the constant function which returns 1 for any $\pi \in \{0,1\}^{\log(m)}$.

Since the term $2^\omega = m$, the resulting algorithm for solving the cyclic string matching problem achieves $O(\sqrt{nm}(\log(n) + \log(\sigma) + \log(m)))$-time complexity, which reduces to $O(\sqrt{nm}\log(n))$, offering an almost quadratic speed-up to the classical $O(n)$ solution.

4.2. The string matching problem allowing for swaps

In the *string matching problem allowing for swaps* [33] (or *swap string matching*) the objective is to find all occurrences of a pattern within a text by allowing two adjacent characters of the pattern to be swapped. It is assumed that each character can be involved in a single swap.

In the classical model of computation, we can find all ρ matching positions in $O(n\sqrt[3]{m}\log(m)\log(\sigma))$ time [33]. But we require $O(\rho m)$ additional time to retrieve all matching configurations.

In the case that the number of allowed swaps is unbounded, we can define an association rule that associates each swap configuration of the pattern with a binary code of length $\omega = m - 1$, similarly as in [7]. Therefore, it would be easy to get an algorithm with $O(\sqrt{n2^m}\log(n))$-time complexity for problem (1), obtaining a quantum advantage for $m = O(\log(n))$. In the case of problem (3) we achieve a $O(\sqrt{\rho n2^m}\log(n))$-time complexity, with a quantum advantage for $\rho = o(n)$ and $m = O(\log(n))$.

We note that a more efficient classical algorithm for string matching with swaps has been proposed in [34], achieving $O(n\log m\log\sigma)$-time complexity for the decision problem.

Therefore, when compared to this improved bound, the quantum advantage of our approach for problem (1) holds only under specific parameter regimes, such as when the pattern length is sufficiently large or when the number of admissible configurations grows with the input size.

In particular, when $m = O(\log n)$, the classical algorithm may outperform the quantum approach, while for larger values of m the QPP framework may still provide an advantage.

Next, we consider the α-bounded variant of this problem, where an upper bound $\alpha > 0$ is given to the number of swaps involved in a match. We first analyze in more detail the simplest case, in which $\alpha = 1$, and discuss later how to generalize it to the cases with $\alpha > 1$.

4.3. The string matching problem with bounded swaps

This variant of the problem can be solved in a classical model of computation in $O(n\sqrt{\alpha\log(m)})$-time [35].

The string matching DFA that recognizes all swapped versions of a pattern x of length m, with at most a single swap, is obtained by using a main sequence of m states $\langle q_0, q_1, \ldots, q_{m-1}\rangle$, for which we have $\delta(I, x_0) = q_0$ and $\delta(q_i, x_{i+1}) = q_{i+1}$, for $0 \leqslant i < m-1$. Such a main sequence represents the pattern configuration in which no swap is present. Additionally, at each state q_i of the automaton for $0 \leqslant i < m-2$, a new linear path of length $m-1-i$ is triggered whose sequence of arcs is labeled by the sequence $\langle x_{i+2}, x_{i+1}, x_{i+3}, \ldots, x_{m-1}\rangle$. More precisely, the path triggered at the state q_i represents the pattern configuration in which a swap is needed at position $i+1$. See Fig. 6 (C) for an automaton built on a pattern with 4 characters.

Let $\mathcal{A}$ be the DFA for the α-bounded swap matching problem, with $\alpha = 1$, constructed over a pattern x of length m. It is easy to observe that the number of complete paths of $\mathcal{A}$ is equal to m. Thus, the set Π of all path codes for the swap string matching problem is defined so that any $p \in \text{PATH}(\mathcal{A})$ is associated to a binary sequence $\pi \in \{0,1\}^{\log(m)}$, which identifies (in binary notation) the position s of the unique swap (if any) in the pattern. Formally, given the value $s = \sum_{i=0}^{\log(m)} \pi_i 2^i$, the characters $x[s]$ and $x[s+1]$ need to be swapped in order to match the corresponding substring of the text. Since the value $m-1$ does not identify a valid location for a swap, we agree that this value is used for the case where no swap is present in the text substring.

The corresponding transformation function t maps a pair (π, x), where π is a binary string of length $\log(m)$ and x is a string of length m, to the string x_π of length $r = m$ such that $x_\pi[i] = x[i]$ if $i \notin \{s, s+1\}$, $x_\pi[s] = x[s+1]$ and $x_\pi[s+1] = x[s]$. For example, if $x = agcatcga$ is a pattern of length 8 and $\pi = 011$ is a path code of length 3, then the corresponding path labeling is the string $x_\pi = agctacga$ where a swap occurs at position 3.

The boolean oracle U_t implementing the transformation function t acts on the pattern register $|x\rangle$ by swapping the qubits $|x_s\rangle$ and $|x_{s+1}\rangle$ whenever the path code $|\pi\rangle$ contains the value s. Such a transformation can be obtained at the cost of $\log(m)$ applications of the rotation operator (see Section 2.3), controlled by the $|\pi\rangle$ register, and operating on an additional register of m ancillæ. The latter is then used to control the swap operations, executable in constant time. It is possible to prove that the total depth of the resulting boolean oracle U_t is equal to $O(\log(m))$.

Since any path code π consisting of $\log(m)$ bits represents a valid swap position between 0 and $m-1$, the test function υ can be dropped once more; that is, it can be implemented as the constant function mapping any $\pi \in \{0,1\}^{\log(m)}$ onto 1.

The resulting quantum algorithm, for the case $\alpha = 1$, achieves an overall $O(\sqrt{nm}(\log(n) + \log(m)))$ complexity, which reduces to $O(\sqrt{nm}\log(n))$, achieving an almost quadratic speed-up compared with the classical solution.

In the more general case where $\alpha > 0$ the problem can be solved by considering π as the concatenation of α binary sequences, $\langle \pi_0, \pi_1, \ldots, \pi_{\alpha-1}\rangle$, all of size $\log(m)$, and where π_i identifies the position of the ith swap, with $0 \leqslant i < \alpha$. A path codes is then a binary sequence $\pi \in \{0,1\}^{\alpha\log(m)}$.

It is agreed that two sequences π_i and π_j, with $i \neq j$, must represent different positions in the pattern, with the exception of position $m-1$, which identifies the case where the correspondent swap operation does not apply. This allows for a number of swaps between 0 and α. Specifically, the case where the occurrence of the text involves no swap operation is obtained by posing $\pi_i = m-1$, for $0 \leqslant i < \alpha$.

The boolean oracle U_t implementing the transformation function t in the general case can be obtained by applying the transformation described in the case $\alpha = 1$ for each binary sequence π_i. Since these transformations all act on the same register $|x\rangle$, they must be applied sequentially. Thus, the resulting circuit has a depth equal to $O(\alpha\log(m))$.

The test function υ in the general case must check for the presence of two pairs of values i and j, such that $i \neq j$ and $\pi_i = \pi_j$. This verification can be done for each pair in time $\log(\log(m))$. Since it must be done for each pair of values, the full check will take $O(m^2 \log(\log(m)))$ time.

The resulting quantum algorithm achieves an $O(\sqrt{nm^\alpha}(\log(n) + \log(m) + m^2 \log\log(m)))$ overall time-complexity, which reduces to $O(\sqrt{nm^\alpha}\log(n))$ time, under the assumption that $\alpha \log(m) = o(m)$.

Assuming that the pattern length m is sufficiently small, the running time $\sqrt{n\,m^\alpha}\log n$ becomes asymptotically smaller than $n\sqrt{\alpha \log m}$ whenever $\alpha \log m \ll \log n$, that is, for any regime in which $m^\alpha = o(n)$.

4.4. String matching problem allowing for a bounded number of gaps

The approximate string matching problem with α-bounded gaps [36] arises in many questions concerning musical information retrieval and bioinformatics. Specifically, if α is an integer value, with $\alpha < m$, we say that a string y of length $m - \alpha$ realizes a match with α gaps against a string x of length m if we can insert α gap symbols in y to make it matching x. In this context, we assume that a gap symbol matches any character of the alphabet. We also assume that a gap cannot be inserted at either the first or last position of the string y. In a classical model of computation, problem (1) with α-bounded gaps can be solved in $O(nm)$-time [36] by dynamic programming and $O(nm/w)$-time [36] by bit-parallelism, while we need $O(\rho m)$ additional time for problem (3).

Note that the set $\mathcal{R}$ of the possible lengths of an occurrence of x in y corresponds to the set $\{m - \alpha, ., m - 1, m\}$. Consequently, the algorithm is repeated $\alpha + 1$ times, once for each possible value of $r \in \mathcal{R}$.

The DFA for the approximate string matching problem with *exactly* α gaps recognizes the language $\mathcal{L}(\mathcal{A})$ of all strings of length $r = m - \alpha$ that are subsequences of x, beginning with x_0 and ending with x_{m-1}. Each string $w \in \mathcal{L}$ can be associated with a *gap map*, that is, a binary sequence of length m where the ith bit is set to zero if and only if it is associated with a gap. Since the first and the last bits of a gap map are always set to 1, we can exclude them and make use of gap maps of length $m - 2$. Thus, the association rule creates a two-way mapping between the set of complete paths of the automaton and the set of the gap maps of length $m - 2$. The set Π is therefore the subset of $\{0, 1\}^{m-2}$ containing all binary sequences with $m - 2 - \alpha$ bit set to 0.

Let $\pi \in \Pi$ and let x' be the string of length $r = m - \alpha$ obtained from x removing the characters of position i, such that $\pi[i] = 1$. Similarly, let x^{G} be the string of length α obtained from x by removing the characters of position i such that $\pi[i] = 0$. The transformation function t maps the string x to a new string $x_\pi = x'x^{\mathrm{G}}$, where the symbols associated to any gap position are moved to the right end of the sequence, preserving the original ordering of the symbols. Formally, for $0 \leqslant i < m$, we have $x_\pi[i] = x'[i]$ if $i \leqslant m - \alpha$ and $x_\pi[i] = x^\epsilon[i - m + \alpha]$ otherwise. We can obtain a boolean oracle U_t for the transformation function running in $O(m^2)$ time.

In addition, since we allow the presence of exactly α gaps in the window of the text, the test function reduces to checking if the qubits in the $|\pi\rangle$ register sum up to α. We can construct a boolean oracle U_υ capable of performing this control and running in $O(m^2)$ time. Hence, the overall time required by the algorithm for problem (1) of approximate string matching with exactly α gaps is equal to $O(\sqrt{n2^m}(\log^2(n) + \log(\sigma) + m^2))$.

Since the original problem needs to check for all occurrences with *at most* α gaps, the above procedure must be repeated $\alpha + 1$ times, considering all values of the number of gaps from 0 to α. Under the assumption that $m = O(\log(n))$, this yields an overall running time of $O(\alpha\sqrt{n2^m}\log^2 n)$ for problem (1), and $O(\sqrt{\rho n 2^m}\log^2(n))$ for problem (3), where ρ denotes the number of occurrences of x in y allowing for at most α swaps.[10]

When compared with the classical $O(nm)$ algorithm, the quantum approach provides an asymptotic advantage only for sufficiently small pattern lengths. In particular, for problem (1), assuming α constant, the quantum running time is asymptotically smaller than $O(nm)$ whenever $m \leq \log_2 n - \Theta(\log\log n)$. Similarly, for problem (3), the quantum algorithm outperforms the classical one whenever $m \leq \log_2\left(\frac{n}{\rho}\right) - \Theta(\log\log n)$, that is, when the pattern length is logarithmic in the text size and the number of valid occurrences is sublinear.

4.5. String matching problem allowing for an unbounded number of gaps

We conclude by briefly discussing the regime in which the number of allowed gaps is unbounded, namely $\alpha = O(m)$. In this setting, problem (1) leads to a quantum running time of $O(m\sqrt{n2^m}\log^2 n)$, while problem (3) admits a complexity $O(\sqrt{\rho n 2^m}\log^2(n))$.

When compared with the classical $O(nm)$ approach, these bounds show that any potential quantum advantage is severely constrained. For problem (1), assuming $\alpha = \Theta(m)$, the quantum algorithm is asymptotically faster than the classical one only if

$$2^m = O\left(\frac{n}{\log^4 n}\right),$$

which implies $m = O(\log n)$. Similarly, for problem (3), a quantum speedup is achievable only when

$$\rho\, 2^m = O\left(\frac{nm^2}{\log^4 n}\right),$$

[10] The omission of the α factor in this expression is justified only under restricted parameter regimes. In particular, when $\alpha = O(1)$, the contribution of α does not affect the asymptotic behavior of the complexity. More generally, the full expression including the α factor must be retained. Analogous considerations apply to the omission of the m factor in subsequent expressions, which is valid only in regimes where $m = O(\log n)$.

again forcing the pattern length to remain logarithmic in the text size and the number of valid occurrences to be sufficiently small.

Therefore, in the unbounded-gap regime, the exponential dependence on m inherent in the quantum approach prevents any asymptotic advantage for moderate or large patterns. This highlights that the proposed quantum techniques are meaningful primarily in the bounded-gap setting, where α is small compared to m.

5. Conclusion

We have introduced a new general method exploiting the features of quantum computation for solving nonstandard text searching problems. Our basic idea takes inspiration from a recent paper by Niroula and Nam [6], whose result is extended, generalized, and adapted here. Our algorithm can be adapted to solve most approximate string matching problems in time $O(\sqrt{n2^{\omega}}\log(n))$, where the value ω depends on the number of complete paths in the automaton built on the pattern and on the proposed association rule. For some of these problems, the running time naturally reduces to $O(\sqrt{nm}\log(n))$ (i.e. $\omega(m) = O(\log(m))$), while in other less favorable cases, it is possible to achieve the quasi-linear time $O(n\log(n))$ only in the case of short patterns, that is, assuming $m = O(\log(n))$.

We have presented our general method in a form that is well suited to solve approximate string matching problems in which any occurrence has the structure of a permutation of the pattern or a subsequence of it. However, the method can be easily extended to handle different types of approximation, such as mismatches or character classes, just to name a few examples. For instance, going into a little more detail, the case of *string matching with errors* could be easily addressed by introducing an additional quantum register, named $|e\rangle$, of size m, representing the error map within the pattern. Formally, it is assumed that, for $0 \leqslant i < m$, the qubit $|e_i\rangle$ takes the value $|1\rangle$ if a mismatch is required at the position i of the pattern and is $|0\rangle$ otherwise. In this context, the path codes are associated with the error maps rather than the pattern permutations, and the transformation function takes $|e\rangle$ rather than $|x\rangle$ as an argument. The match operator can be modified to transpose the mismatch information in the $|e\rangle$ register.

Our future research will go in the direction of a more complete generalization of the path parallelism technique and of finding efficient solutions to specific approximation problems.

CRediT authorship contribution statement

Simone Faro: Writing – review & editing, Writing – original draft, Investigation; **Arianna Pavone:** Writing – review & editing, Writing – original draft, Investigation; **Caterina Viola:** Writing – review & editing, Writing – original draft, Investigation.

Declaration of competing interest

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.

Supplementary material

Supplementary material associated with this article can be found in the online version at 10.1016/j.tcs.2026.116059.

References

[1] P.W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Comput. 26 (5) (1997) 1484–1509. https://doi.org/10.1137/s0097539795293172
[2] L.K. Grover, A fast quantum mechanical algorithm for database search, in: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, STOC '96, ACM, New York, NY, USA, 1996, pp. 212–219. https://doi.org/10.1145/237814.237866
[3] F. Arute, K.A., Quantum supremacy using a programmable superconducting processor, Nature 574 (2019) 505–510. https://api.semanticscholar.org/CorpusID:204836822.
[4] H. Ramesh, V. Vinay, String matching in $\tilde{O}(\sqrt{n}+\sqrt{m})$ quantum time, J. Discrete Algorithms 1 (1) (2003) 103–110. https://doi.org/10.1016/S1570-8667(03)00010-8
[5] A. Montanaro, Quantum pattern matching fast on average, Algorithmica 77 (1) (2017) 16–39. https://doi.org/10.1007/s00453-015-0060-4
[6] P. Niroula, Y. Nam, A quantum algorithm for string matching, npj Quantum Inf. 7 37 (2021). https://doi.org/10.1038/s41534-021-00369-3
[7] 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, *Lecture Notes in Computer Science*, Springer, Vol. 13960, 2023, pp. 117–133. https://doi.org/10.1007/978-3-031-38100-3_9
[8] 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.
[9] M. Boroujeni, S. Ehsani, M. Ghodsi, M. Hajiaghayi, S. Seddighin, Approximating edit distance in truly subquadratic time: quantum and MapReduce, J. ACM 68 (3) (2021). https://doi.org/10.1145/3456807
[10] 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
[11] 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), *CEUR Workshop Proceedings*, IC-EATCS, CEUR, Vol. 3811, 2024, pp. 1–15. https://ceur-ws.org/Vol-3811/paper090.pdf.
[12] M. Crochemore, W. Rytter, Text Algorithms, Oxford Univ., 1994. http://www-igm.univ-mlv.fr/%7Emac/REC/B1.html.
[13] R.A. Baeza-Yates, G.H. Gonnet, A new approach to text searching, Commun. ACM 35 (10) (1992) 74–82. https://doi.org/10.1145/135239.135243
[14] 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
[15] 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

[16] K. Phalak, A. Chatterjee, S. Ghosh, Quantum random access memory for dummies, 2023, . 2305.01178
[17] V. Giovannetti, S. Lloyd, L. Maccone, Quantum random access memory, Phys. Rev. Lett. 100 (16) (2008). https://doi.org/10.1103/physrevlett.100.160501
[18] 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.
[19] P. Høyer, R. Spalek, Quantum fan-out is powerful, Theory C. 1 (2005) 81–103.
[20] M. Fang, S. Fenner, F. Green, S. Homer, Y. Zhang, Quantum lower bounds for fanout, Quantum Info. Comput. 6 (1) (2006) 46–57.
[21] S. Balauca, A. Arusoaie, Efficient constructions for simulating multi controlled quantum gates, in: Computational Science - ICCS 2022, *LNCS*, Springer, Vol. 13353, 2022, pp. 179–194. https://doi.org/10.1007/978-3-031-08760-8_16
[22] 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
[23] 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
[24] 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, *Contemporary Mathematics*, American Mathematical Society, Vol. 305, 2002, pp. 53–74. https://doi.org/10.1090/conm/305/05215
[25] 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%28199806%2946%3A4/5%3C493%3A%3AAID-PROP493%3E3.0.CO%3B2-P.
[26] A. Ambainis, Quantum search algorithms, SIGACT News 35 (2) (2004) 22–35. https://doi.org/10.1145/992287.992296
[27] A. Pavone, C. Viola, The quantum cyclic rotation gate, in: G. Castiglione, M. Sciortino (Eds.), Proceedings of the 24th Italian Conference on Theoretical Computer Science (ITCTS 2023), Vol-3587 of *Italian Chapter of the European Association for Theoretical Computer Science*, IC-EATCS, University of Palermo, Italy, 2023, pp. 206–218.
[28] A. Pavone, C. Viola, The quantum cyclic rotation gate, SN Comput. Sci. 5 (7) (2024). https://doi.org/10.1007/s42979-024-03141-4
[29] C. Moore, M. Nilsson, Parallel quantum computation and quantum codes, SIAM J. Comput. 31 (3) (2001) 799–815. https://doi.org/10.1137/S0097539799355053
[30] 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), *CEUR Workshop Proceedings*, IC-EATCS, CEUR, Vol. 3811, 2024, pp. 29–41. https://ceur-ws.org/Vol-3811/paper270.pdf.
[31] M. Lothaire, Applied Combinatorics on Words, Cambridge University Press, 2005.
[32] K. Fredriksson, S. Grabowski, Average-optimal string matching, J. Discrete Algorithms 7 (4) (2009) 579–594. https://doi.org/10.1016/j.jda.2008.09.001
[33] A. Amir, Y. Aumann, G.M. Landau, M. Lewenstein, N. Lewenstein, Pattern matching with swaps, J. Algorithms 37 (2) (2000) 247–266. https://doi.org/10.1006/jagm.2000.1120
[34] A. Amir, R. Cole, R. Hariharan, M. Lewenstein, E. Porat, Overlap matching, Inf. Comput. 181 (1) (2003) 57–74. https://doi.org/10.1016/S0890-5401(02)00035-4
[35] O. Lipsky, B. Porat, E. Porat, B. Riva Shalom, A. Tzur, String matching with up to k swaps and mismatches, Inf. Comput. 208 (9) (2010) 1020–1030. https://doi.org/10.1016/j.ic.2010.04.001
[36] M. Crochemore, C.S. Iliopoulos, C. Makris, W. Rytter, A.K. Tsakalidis, T. Tsichlas, Approximate string matching with gaps, Nord. J. Comput. 9 (1) (2002) 54–65.