A Quantum Circuit for the Cyclic String Matching Problem

Arianna Pavone[1] and Caterina Viola[2⋆]

[1] Dipartimento di Matematica e Informatica, Università degli Studi di Palermo
Via Archirafi 34, 90123, Palermo, Italy
ariannamaria.pavone@unipa.it
[2] Dipartimento di Matematica e Informatica, Università degli Studi di Catania
Viale A. Doria n.6, 95125, Catania, Italy
caterina.viola@unict.it

Abstract. The cyclic string matching problem aims to detect an occurrence of any cyclical shift of a given sequence of characters within a (usually larger) one. This variant of the (classical) string matching problem arises in several real-world problems, such as DNA analysis and spoken natural language recognition, in which it is necessary to deal with sequences that lack a precise beginning or end and are equivalent up to cyclical shifting. We present a quantum algorithm, in the form of a quantum circuit, for solving the cyclic string matching problem. This algorithm requires quadratically fewer time steps than the most efficient counterpart algorithm running on a classical machine. Additionally, we provide a practical implementation of the presented algorithm using the Qiskit toolkit[1].

Keywords: quantum computing, non-standard text searching, text processing

1 Introduction

In the field of computing, the challenge of pattern identification in textual data spans various disciplines, including natural language processing, information retrieval, and computational biology. Known as *string matching*, this task involves identifying every occurrence of a specified pattern x, of length m, within a text y, of length n. Both sequences are composed of characters drawn from an alphabet Σ, with a size of σ symbols. This field encompasses both *exact* and *approximate* matching.

In *exact string matching*, the goal is to find exact alignments of the pattern within the text. *Approximate string matching* allows for some discrepancies, using a distance function to measure how closely a substring matches the pattern. Various methods exist for approximate matching, including Hamming distance, Levenshtein distance, order-preserving string matching, and Cartesian tree pattern matching. These methods are widely used across different applications, such as genetic analysis, data compression, and natural language processing.

In this paper, we focus on a specific approximation problem known as the *cyclic string matching* problem [18]. Here, the pattern is a *cyclic string*, meaning it can appear in the text in its original form or any of its cyclic rotations. For example, the strings *accba*, *cbaac*, and *baacc* are all variations of the same cyclic string.

⋆ A. Pavone is supported by PNRR project ITSERR - Italian Strengthening of the ESFRI RI RESILIENCE.
C. Viola is 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.

[1] Qiskit is an open-source software development framework for working on utility-scale quantum computers, developed by IBM.

Arianna Pavone, Caterina Viola: A Quantum Circuit for the Cyclic String Matching Problem, pp. 50–70.
Proceedings of PSC 2024, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-07328-5 © Czech Technical University in Prague, Czech Republic

Cyclic strings are significant in both computational and mathematical contexts, particularly within combinatorics. They lack distinct initial or terminal positions, resembling a continuous loop. This property makes cyclic strings particularly interesting and useful in various scientific and practical applications.

Cyclic rotations have various applications, including, for instance, image and signal processing, where they can be employed to perform cyclic shifts on images or signals, such as image rolling [16] or time delay of a signal. Cyclic rotations can be also used to design efficient algorithms sorting data [20]. In addition, sequences admitting cyclic rotations are also relevant in various biological contexts, including viruses [28,9] and bacteria [26]. Thus, the analysis of organisms with a cyclic structure can benefit from algorithms designed for strings that allow for cyclic rotations [17].

The *Cyclic String Matching* problem has numerous applications, including: Bioinformatics, in identifying repeated motifs or conserved sequences within genomic data; Data Compression, in detecting and removing repetitive patterns to enhance compression; Pattern Recognition, in identifying and comparing cyclically repeating patterns; Natural Language Processing, in analyzing literary texts for cyclically repeating phrases or word sequences; Control and Monitoring Systems, in monitoring biological signals or recognizing fault patterns in machinery.

In classical computational models, solutions to the *Cyclic String Matching* problem can be achieved in linear time [18].

1.1 Quantum Computing and String Matching

Quantum computing represents an innovative frontier in computer science, leveraging the principles of quantum mechanics to develop advanced computing systems that differ significantly from classical models. Unlike classical computers, which process information using discrete binary bits confined to states 0 or 1, quantum computing utilizes *quantum bits* or *qubits*. Qubits can exist in a *superposition* of states, allowing them to represent multiple values simultaneously. Additionally, *entanglement* enables two or more qubits to perform correlated operations, enhancing computational power.

Over the past few decades, quantum computing has emerged as a transformative technology with the potential to revolutionize various fields, including cryptography and materials science. This nascent field has achieved significant milestones, highlighting its ability to solve problems once considered intractable for classical computers. One of the most notable achievements is the demonstration of quantum supremacy, where quantum computers have outperformed the world's most powerful supercomputers on specific tasks.

Quantum algorithms have made substantial progress, particularly in optimization and simulation tasks. Grover's algorithm [14] and Shor's algorithm [25] provided early examples of quantum advantage, offering quadratic speedup in database searching and exponential speedup in integer factorization, respectively. These theoretical advancements have paved the way for practical applications in secure communications and complex problem-solving.

Recent studies [7] have successfully applied quantum computation to tackle NP-complete problems, which are notoriously challenging for classical algorithms [27]. In natural language processing, classifiers that integrate quantum and classical computing techniques are expected to have a substantial impact, especially in areas such as classification [6].

As quantum hardware continues to evolve and algorithms become more sophisticated, the integration of quantum computing into broader scientific and industrial applications becomes increasingly feasible. This progression promises not only to expand our computational capabilities but also to redefine problem-solving paradigms across numerous disciplines.

Recent research has focused on applying quantum computing to string matching, exploring the potential for sub-linear time complexities. Notably, Ramesh and Vinay [23] combined Grover's search with parallel string matching to develop a quantum algorithm with $\tilde{\mathcal{O}}(\sqrt{n})$ complexity[2]. Montanaro [19] highlighted the substantial gap between quantum and classical complexities in this context. In 2021, Niroula and Nam [21] proposed a quantum circuit model solution with $\tilde{\mathcal{O}}(\sqrt{n})$ complexity, advancing the field significantly.

In the field of quantum computation, the cyclic rotation of the states of a given register of qubits has been effectively used in solutions for text processing [21,8,13], and specifically for exact and approximate string matching [11,12]. The recent algorithm by Niroula and Nam [21] makes clever use of cyclic rotations of the registers encoding the input strings to achieve a superposition of all their possible alignments and to perform a parallel comparison against the pattern.

This idea was later generalized by Cantone *et al.* in [8] to get a quantum solution to the string matching problem allowing for swaps of adjacent characters, which is more time-efficient than the best known classical counterpart. In their paper, Niroula and Nam provide insight into the fact that a circuit performing a cyclic rotation of the states of a given register of qubits can be executed in time $\mathcal{O}(\log(n))$. The basic idea is that at each step of the algorithm that accomplishes the permutation, it is possible to place at least half of the qubits that still need to be moved to their final position. Since the number of qubits to be placed decreases by at least half at each iteration, $\mathcal{O}(\log(n))$ steps are needed to achieve the target permutation.

1.2 Our Contribution

We propose a novel quantum solution to the *Cyclic String Matching* problem, building on the algorithm by Niroula and Nam [21] to handle cyclically rotated patterns. Our approach utilizes the circuit-based quantum model, making it compatible with modern quantum computing platforms. We implement the algorithm using Qiskit, allowing for testing on both quantum simulators and actual quantum computers.

Our method offers significant computational efficiency, with a time-complexity of $\tilde{\mathcal{O}}(\sqrt{nm})$, where m and n are the lengths of the pattern and text, respectively. This represents a substantial speed-up over classical algorithms, particularly for patterns of constant length, promising enhanced performance in practical applications.

The paper is organized as follows. In Section 2 we introduce the basic notions and some essential notations for our discussion. In Section 3 we recall the construction by Pavone and Viola [22] of a quantum circuit for the cyclic shifting. In Section 4 we present the quantum algorithm for Cyclic String Matching and discuss its computational complexity. We draw our conclusions in Section 5.

In Appendix A, a simple implementation of the new algorithm using Qiskit is provided.

[2] The big-$\tilde{\mathcal{O}}$-notation works as the big-$\mathcal{O}$-notation except for hiding polylogarithmic factors

2 Preliminaries

2.1 Cyclic String Matchings

Given a string x, of length $n \geq 0$, we represent it as a finite array $x[0\,..\,n-1]$. The empty string is denoted by ε. We denote by $x[i]$ the $(i+1)$-st character of x, for $0 \leq i < n$, and by $x[i\,..\,j]$ the substring of x contained between the $(i+1)$-st and the $(j+1)$-st characters of x, for $0 \leq i \leq j < n$. For ease of notation, the $(i+1)$-th character of the string x will also be denoted by the symbol x_i so that $x = x_0 x_1 \ldots x_{n-1}$. Throughout our paper, we assume that all the strings we deal with have length equal to a power of 2. This can be assumed without loss of generality. Indeed, given a character string of length n, we can always add to the string a suitable number of copies of a special character outside of the alphabet so as to get a string of length 2^p for some $p \in \mathbb{N}$. A leftward *circular shift* (or rotation) of a string $x = x_0, \ldots, x_{m-1}$ by s positions is the string $R_s(x) = x_{(m-s) \mod m}, x_{(m-s+1) \mod m} \cdots, x_{(m-s+m-1) \mod m}$ for $0 \leq s < m$. There are m distinct cyclic rotations of x; more precisely, there is a cyclic rotation of x by i positions to the right for each $i \in \{0, \ldots, m-1\}$.

Let x and y be two strings of length m and n, respectively, with $m < n$. Cyclic String Matching is the problem of finding a substring of y that matches any leftward cyclic rotation of x; in other words, the objective of the problem is to find $s \in \{0, \ldots, m-1\}$ and a position $j \in \{0, \ldots, n-1\}$ such that $R_s(x)$ matches $y[j..j+m-1]$, that is $x[m-s+i] = y[j+i]$ for $0 \leq i < m$.

2.2 Quantum Computation

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 are called *amplitudes*, are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$, representing the probability of measuring the qubit in the state $|0\rangle$ or $|1\rangle$, respectively. A *quantum measurement* is the only operation giving access to the information on the state of 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 i-th least significant binary digit of k. We use $|q\rangle^{\oplus n}$ 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 $f\colon \{0,1\}^n \to \{0,1\}$ 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, $U_f |x\rangle = (-1)^{f(x)} |x\rangle$.

2.3 The Quantum Circuit Model

The model of computation we adopt in this paper is that of *quantum circuits*. Quantum circuits 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. Furthermore, because of the No-Cloning Theorem [29], 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*[3].

There are two major measures of the computational time complexity in general circuital 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. 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 set to $|1\rangle$, 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 $|q_0, q_1, q_0 q_1 \oplus q_2\rangle$.

[3] Ancilla qubits or ancillaæare 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.

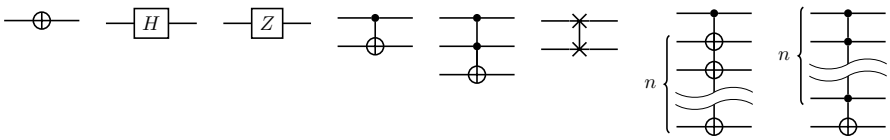

Figure 1. The representation of the following basic gates (from left to right): Pauli-X, Hadamard, Pauli-Z, CNOT, Toffoli, Swap, Fanout and Multicontrolled NOT.

Figure 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.

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

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 constant time fanout can be obtained by the product of n controlled-not gates, the no-cloning theorem makes it difficult to directly fan qubits out in constant depth [15]. 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 [10], all initialized to $|0\rangle$.

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 $|q_0, q_1, q_2, \ldots, q_{n-1}\rangle$ to $|q_0, q_1, \ldots, q_{n-2}, (q_0 \cdot q_1 \cdots q_{n-2}) \oplus q_{n-1}\rangle$. Similarly, the n-ary *multicontrolled phase-flip* operator maps $|q_0, q_1, q_2, \ldots, q_{n-1}\rangle$ to $|q_0, q_1, \ldots, q_{n-2}, (-1)^{q_0 \cdot q_1 \cdots q_{n-2}} q_{n-1}\rangle$.

It is possible to obtain a logarithmic-depth quantum circuit performing a multicontrolled operator [2], by exploiting parallelism and using $n/2$ ancilla qubits. We also mention a recent result [24] that enables the implementation of multi-controlled NOT gates in constant time in architectures with trapped ions and neutral atoms.

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 $0 \leq i < t$, 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 $\mathcal{O}(\log(t))$ time for the fanout operator.

2.4 Grover's Search Algorithm

Grover's search algorithm [14] is one of the most famous results proving theoretical quantum supremacy, together with Shor's algorithm [25]. Grover's algorithm searches a desired item within an unstructured dataset of n items in $\mathcal{O}(\sqrt{n})$ 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 [3].

Given a function $f\colon \{0,1\}^{\log(n)} \to \{0,1\}$ 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. Figure 2 represents the circuital translation of Grover's algorithm, in which details on the circuit implementing the Diffuser used by the algorithm are provided.

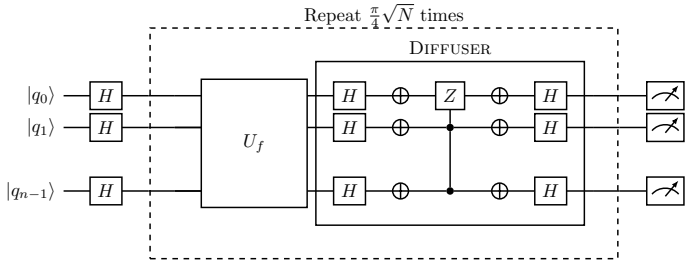

Figure 2. The circuit implementing Grover's algorithm.

The Diffuser requires a multicontrolled Z gate of size n; therefore, its depth is logarithmic in n. Thus, Grover's algorithm has depth $\mathcal{O}\left(\sqrt{n}(T(n)+\log(n))\right)$,[4] 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 [5], and that this bound is optimal [4]. 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 [1].

[4] If we assume to be able to implement the multicontrolled Z gate in constant time [24], a Grover's search on a dataset of n items achieves $\mathcal{O}\left(\sqrt{n}T(n)\right)$ time.

```
CIRCULAR-SHIFT(|q⟩, k):
1. for i ← 1 to log(n) − log(k) do
2.    for j ← 0 to n/(2^i k) − 1 do (in parallel)
3.       for h ← jk2^i to (j2^i + 1)k − 1 do (in parallel)
8.          Swap(|q_h⟩, |q_{h−2^{i−1}k}⟩)
```

Figure 3. The pseudocode of the CIRCULAR-SHIFT algorithm for circularly rotating a quantum register $|q\rangle$ of size n by k positions.

3 A controlled circular shift operator

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

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

A circular shift gate can always be decomposed into (a sequence of) parallel Swap gates. Figure 3 contains the pseudocode of a quantum circuit circularly rotating a quantum register $|q\rangle$ of size n by s positions, for the case that s is a power of 2, proposed by [22,21][5]. Such a circuit has depth $\mathcal{O}(\log(n))$ and at each time-layer performs at most $\frac{n}{2}$ parallel swaps.

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

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

Because the controlled cyclic shift gate does not change the control register, sometimes we abuse the notation writing ROT_j for $\text{ROT}(|j\rangle, \cdot)$.

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 $\log(n)$ qubits. We have then $|j\rangle = \bigotimes_{i=0}^{\log(n)-1} |k_i\rangle$, where $|j_i\rangle$ is initialized with the value of the i-th least significant bit of the binary representation of the value k. It will then be enough, for any value of i, such that $0 \leq i < \log(n)$, 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 $\log(n)$ 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 the last paragraph of Section 2.3 that involves the use of $\log(n)\frac{n}{2}$ ancilla qubits for the application of all parallel operators controlled by the same qubit, that is, for each $0 \leq i \leq \log(n)$, the cyclic shift controlled by 2^i positions we need $\frac{n}{2}$ ancilla qubits. The controlled cyclic shift by 2^i positions requires $\mathcal{O}(\log(n))$ time-steps to fan the information on the control qubit out to the ancillæ plus $\mathcal{O}(\log(n))$ to perform the cyclic rotation. Therefore, the overall controlled circular shift operator ROT_j over a register of size

[5] Observe that the algorithm in [22] operates a rightward cyclic shift, while we work with leftward cyclic shifts. However, translating a circuit for the rightward cyclic shift to a leftward one by the same number of positions, and vice versa, is a very easy task

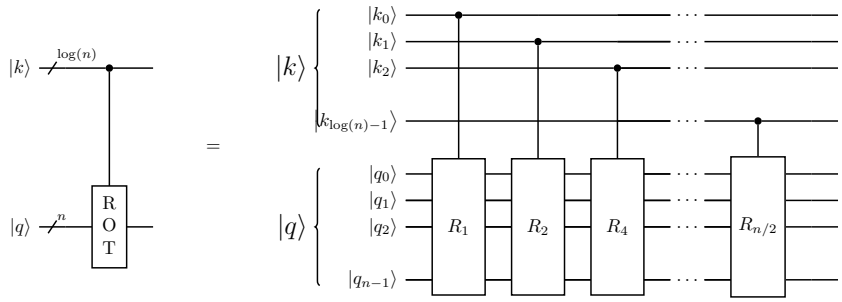

Figure 4. A circular shift operator on a n-qubit register $|q\rangle$ controlled by a $\log(n)$-qubit register $|k\rangle$. The circuit makes use of $\log(n)$ ancilla qubits which is not shown in this graphical representation. On the left of the figure, we show the succinct graphical representation used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

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 $\mathcal{O}(\log^2(n))$. Figure 4 contains an illustration of such a circuit.

4 The Algorithm

Throughout the entire Section 4, let a pattern x, of length m, and a text y, of length n.

The problem of finding an occurrence of a cyclic rotation of the pattern x in y can be restated as the problem of finding an element in the subset $Cyc(x, y) \subseteq \{1, \ldots, m-1\} \times \{1, \ldots, n-1\}$ such that

$$Cyc(x, y) = \{(i, j) \,:\, 0 \leq i < m,\, 0 \leq j < n \text{ , and } y[j..j+m-1] = R_i(x)\}\,. \tag{1}$$

We point out that in such a definition of the computational problem, the objective is to find not only the position of the text at which a cyclic rotation of the pattern occurs but also the amplitude of the rotation. However, there is a slightly alternative version of the problem in which the wanted output of a solving algorithm is only the position of the text at which a cyclic rotation of the text occurs. The latter version of the cyclic string-matching problem can be restated as the problem of finding an element in the subset $Cyc'(x, y) \subseteq \{1, \ldots, n-1\}$ such that

$$Cyc'(x, y) = \{j \colon 0 \leq j < n \text{ and } \exists 0 \leq i < m \text{ such that } y[j..j+m-1] = R_i(x)\}\,. \tag{2}$$

Our algorithm solves the problem of finding an element in the set defined in Equation 1 and, as we show, it can be easily adapted to solving the problem of finding an element in the set defined in Equation 2.

We denote by $\chi_{Cyc(x,y)}$ the *characteristic function* of $Cyc(x, y)$, that is, the function $\chi_{Cyc(x,y)} \colon \{1, \ldots, m-1\} \times \{1, \ldots, n-1\} \to \{0, 1\}$ defined by

$$\chi_{Cyc(x,y)}(i, j) = \begin{cases} 1 & \text{whenever } (i, j) \in Cyc(x, y), \\ 0 & \text{otherwise.} \end{cases}$$

Analogously, the characteristic function of $Cyc'(x, y)$ is $\chi_{Cyc'(x,y)} \colon \{1, \ldots, n-1\} \to \{0, 1\}$ defined by

$$\chi'_{Cyc(x,y)}(j) = \begin{cases} 1 & \text{whenever } j \in Cyc'(x, y), \\ 0 & \text{otherwise.} \end{cases}$$

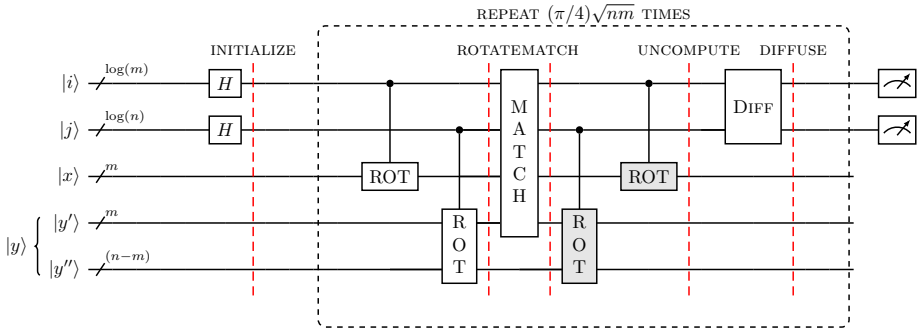

Figure 5. A simplified diagram describing the quantum circuit solving Cyclic String Matching.

The algorithm is an application of Grover's search algorithm (ref. to Section 2.4) for a solution to the function $\chi_{Cyc(x,y)}$, in the form of a quantum circuit. It consists of two steps that are iterated $\lfloor(\pi/4)\sqrt{nm}\rfloor$ times: the phase oracle for the function $\chi_{Cyc(x,y)}$; and the application of Grover's diffuser. The phase oracle of the function $\chi_{Cyc(x,y)}$ is implemented by the rotation and the match steps, and their respective uncomputing steps. In Figure 5, we give a simplified diagram illustrating the circuit, neglecting the ancilla registers needed in the computation.

In the following sections, we describe the computation giving the details of the main steps, alongside their correctness, and analyze the overall time complexity. Throughout the next sections, we will refer to the representation of the circuit shown in Figure 5.

4.1 The Initialization Step

Let us describe the quantum registers involved in the computation and their initialization.

- The register $|i\rangle$ has size $\log(m)$ and it retains the information on the amplitude of the circular rotation of the pattern. More precisely, the kth qubit of the register $|i\rangle$ is either 1, 0, or a coherent superposition of these, depending on the kth digit of the binary representation of the amplitude of the rotation, which is a value between 0 and $m-1$. Starting from $|0\rangle^{\otimes \log(m)}$, the register $|i\rangle$ is initialized to $|+\rangle^{\otimes \log(m)}$ through the application of $\log(m)$ Hadamard's gates so that it contains the superposition of all possible values in $\{0,...,m-1\}$. Formally, $|i\rangle = \frac{1}{\sqrt{m}} \sum_{s\in\{0,1\}^{\log(m)}} |s\rangle$.
- The register $|j\rangle$ has size $\log(n)$ and retains the information on the position $j \in \{0,\ldots,n-1\}$ of the substring to be examined within the text, that is, specifically, the (binary representation of the) value of the position j of the text where the first character of such a substring is found. Starting from $|0\rangle^{\otimes \log(n)}$, the register $|j\rangle$ is initialized to $|+\rangle^{\otimes \log(n)}$ through the application of $\log(n)$ Hadamard's gates so to contain the superposition of all possible values in $\{0,...,n-1\}$. Formally, $|j\rangle = \frac{1}{\sqrt{n}} \sum_{t\in\{0,1\}^{\log(n)}} |t\rangle$.
- The register $|x\rangle$ has size m and and its constituent qubits are initialized so that $|x_h\rangle$ contains the h-th bit of the pattern, for $0 \leq h < m$.
- Similarly, the register $|y\rangle$ has size n and its constituent qubits are initialized so that $|y_k\rangle$ contains the k-th bit of the text, for $0 \leq k < n$. For visualization purposes

the $|y\rangle$ register is divided into two registers, $|y'\rangle$ and $|y''\rangle$, of size m and $n-m$, respectively.
– Finally, we have three ancilla registers[6], $|a\rangle$, $|b\rangle$, and $|c\rangle$, that – for simplicity – are not depicted in Figure 5. The register $|a\rangle$ is an ancilla register that is needed to use the same control register for different operators (see Section 2.3). In particular, $|a\rangle$ has length $\log(m)+\log(n)$; the first $\log m$ qubits of $|a\rangle$ are needed for the implementation of the controlled cyclic shift operator ROT (see Section 3) to be applied to the pattern x, while the last $\log(n)$ qubits of $|a\rangle$ are needed for the implementation of the text y. All the qubits from $|a\rangle$ are initialized to $|0\rangle$. The register $|b\rangle$ consists of $\lceil\frac{m}{2}\rceil+1$ qubits and is needed to implement the multicontrolled phase-flip gate needed in the implementation of the phase oracle for $\chi(x,y)$, and precisely within the matching step (see Section 4.4) and is initialized to $|0\rangle^{\otimes\lceil\frac{m}{2}\rceil+1}$. Similarly, the register $|c\rangle$ consists of $\lceil\frac{\log(m)+\log(n)}{2}\rceil+1$ qubits and is needed to implement the multicontrolled phase-flip gate needed in the implementation of the Grover's Diffuser (see Section 2.4) and is initialized to $|0\rangle^{\otimes\lceil\frac{m}{2}\rceil+1}$.

Overall, the circuit needs $\mathcal{O}(\log(mn)+m+n)$ qubits, i.e., it has $\mathcal{O}(n)$-space complexity.

Observe that each of the operators applied during this step takes constant time, and since they can be applied in parallel, the initialization phase takes $\mathcal{O}(1)$ time.

We can define the *global* register

$$|q\rangle := |i\rangle \otimes |j\rangle \otimes |x\rangle \otimes |y\rangle\,,$$

whose constituent qubits are all those needed in the circuit (except for the ancillæ). Referring to $|q\rangle$, we can algebraically prove the correctness of the quantum algorithm described by the circuit in Figure 5. After the initialization we have

$$|\psi\rangle := \textsc{Initialize}\,|q\rangle = \frac{1}{\sqrt{m}}\sum_{s\in\{0,1\}^{\log(m)}}|s\rangle \otimes \frac{1}{\sqrt{n}}\sum_{t\in\{0,1\}^{\log(n)}}|t\rangle\otimes|x\rangle\otimes|y\rangle\,.$$

4.2 The Grover's Iterations

Since we are searching for solutions to $\chi_{cyc(x,y)}$ in a space of dimension $m\cdot n$, and since we do not know the number of solutions, we repeat the rotation step, the matching step, and the Grover's Diffuser $\lfloor\frac{\pi}{4}\sqrt{mn}\rfloor$ times. The phase oracle for the function $\chi_{cyc(x,y)}$ consists of the rotation step, the matching step, and the uncomputing step. After the $\lfloor\frac{\pi}{4}\sqrt{mn}\rfloor$ Grover's iterations we measure the first $m+n$ qubits of the register $|q\rangle$.

Already in Section 2.4, we presented the circuit implementing the Grover's Diffuser and discussed how a suitable number of Grover's iterations (i.e. phase oracle followed by Diffuser) maximizes - and makes very close to 1 - the probability of measuring a solution to the target function, which in our case is $\chi_{cyc(x,y)}$. Therefore, in order to prove the correctness of our procedure it is enough to prove that indeed the rotation step, the matching step, and the uncomputing step implement the oracle function for $\chi_{cyc(x,y)}$, that is the combined effect of a single application of each of these step transforms $|\psi\rangle$ by changing nothing but the phase of the state $|s\rangle\,|t\rangle$ corresponding to a pair (s,t) that is a solution $\chi_{cyc(x,y)}$, that is a pair (s,t) such that the cyclic rotation of x by s positions matches the substring $[y_j,\ldots,y_{j+m-1}]$ of the text.

[6] Ancilla registers are quantum registers made of ancilla qubits.

4.3 The Rotation Step

The Rotation Step consists of two controlled cyclic shift gates, see Section 3: the first of which performs a leftward cyclic rotation of the register $|x\rangle$ controlled by $|i\rangle$, while the second one performs a leftward cyclic rotation of the register $|y\rangle$ controlled by $|j\rangle$. Since the two sets containing the qubits manipulated by $\textsc{rot}(|i\rangle, |x\rangle)$ and $\textsc{rot}(|j\rangle, |y\rangle)$, respectively, are disjoint, we can execute the two gates in parallel. The depth of the circuit performing ROT on $|i\rangle \otimes |x\rangle$ is $\mathcal{O}(\log^2(m))$, and the depth of the circuit performing ROT on $|j\rangle \otimes |y\rangle$ is $\mathcal{O}(\log^2(n))$; therefore, overall, the rotation steps has time complexity $\mathcal{O}(\max(\log^2(m), \log^2(n))) = \mathcal{O}(\log^2(n))$. The joint action of $\textsc{rot}(|i\rangle, |x\rangle)$ and $\textsc{rot}(|j\rangle, |y\rangle)$ on the register $|\psi\rangle$ is described by

$$|\psi'\rangle = \textsc{Rotate}\,|\psi\rangle = \frac{1}{\sqrt{m}} \sum_{s\in\{0,1\}}^{\log(m)} |s\rangle \otimes \frac{1}{\sqrt{n}} \sum_{t\in\{0,1\}}^{\log(n)} |t\rangle \otimes \frac{1}{\sqrt{m}} \sum_{i=0}^{m} |R_s(x)\rangle \otimes \frac{1}{\sqrt{n}} \sum_{i=0}^{n} |R_t(y)\rangle\,.$$

4.4 The Matching Step

The Matching Step consists of a single gate (denoted by MATCH in Figure 5) that makes a *qubitwise* comparison between the two registers $|x\rangle$ and $|y'\rangle$, which at the moment before the application of MATCH are actually equal to $\frac{1}{\sqrt{m}} \sum_{i=0}^{m} |R_s(x)\rangle$ and $\frac{1}{\sqrt{n}} \sum_{i=0}^{n} |R_t(y)\rangle$; in the case that $\frac{1}{\sqrt{m}} \sum_{i=0}^{m} |R_s(x)\rangle$ and $\frac{1}{\sqrt{n}} \sum_{i=0}^{n} |R_t(y)\rangle$ are qubitwise equal, MATCH flip the phase of the first qubit of $|i\rangle$, which results in a phase-flip of the register $|i\rangle \otimes |j\rangle$. The quantum circuit implementing the MATCH gate is illustrated in Figure 6.

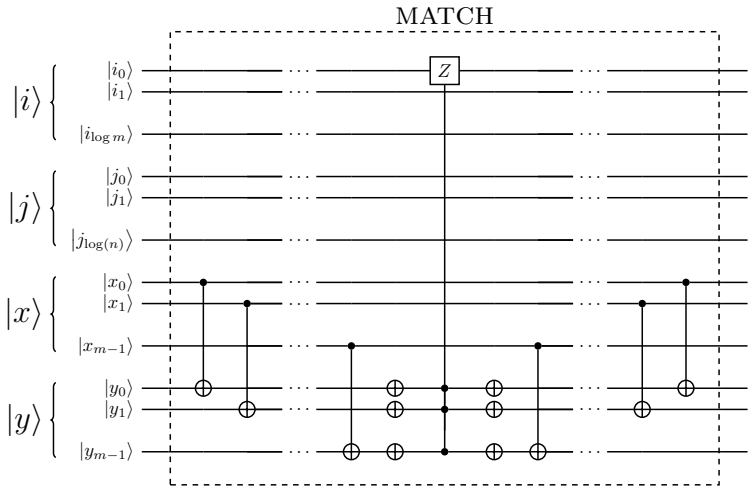

Figure 6. A circuit detecting an exact match between two strings of length m.

The comparisons between qubits can be performed in parallel. More precisely, for $0 \leq \ell < m$, the states of the qubits $|i_\ell\rangle$ and $|j_\ell\rangle$ are compared in $\mathcal{O}(\log(m))$-time by means of m parallel CNOT gates and a multiple controlled phase-flip gate with m controls. Specifically, the parallel application of the m parallel CNOT takes $\mathcal{O}(1)$ time, while the multiple controlled phase-flip gate needs $\mathcal{O}(\log(m))$ time (and $\frac{m}{2}$ ancillæ). Thus, the overall depth of the MATCH gate is $\mathcal{O}(\log(m))$. After the match phase we have

$$
\begin{aligned}
|\psi''\rangle =& \textsc{Matching}\,|\psi'\rangle \\
=& \frac{1}{\sqrt{mn}} \sum_{s\in\{0,1\}}^{\log(m)} \sum_{t\in\{0,1\}}^{\log(n)} (-1)^{\chi_{Cyc(x,y)}(s,t)} |s\rangle \otimes |t\rangle \otimes \frac{1}{\sqrt{m}} \sum_{i=0}^{m} |R_s(x)\rangle \otimes \frac{1}{\sqrt{n}} \sum_{i=0}^{n} |R_t(y)\rangle\,.
\end{aligned}
$$

4.5 The Uncomputing Step

The uncomputing step rotate the register $|x\rangle$ and $|y\rangle$ back to their initial configuration and it consists of a *rightward* cyclic rotation of $|x\rangle$ controlled by $|i\rangle$ and *rightward* cyclic rotation of $|y\rangle$ controlled by $|j\rangle$. Clearly, this step has time-complexity equal to that of the rotating step (see Section 4.3), that is $\mathcal{O}(\log^2(n))$. After the uncomputing step we have

$$
\begin{aligned}
|\psi'''\rangle =& \textsc{Uncomputing}\,|\psi''\rangle \\
=& \frac{1}{\sqrt{mn}} \sum_{s\in\{0,1\}}^{\log(m)} \sum_{t\in\{0,1\}}^{\log(n)} (-1)^{\chi_{Cyc(x,y)}(s,t)} |s\rangle \otimes |t\rangle \otimes \frac{1}{\sqrt{m}} \sum_{i=0}^{m} |R_{-s}R_s(x)\rangle \otimes \frac{1}{\sqrt{n}} \sum_{i=0}^{n} |R_{-t}R_t(y)\rangle \\
=& \frac{1}{\sqrt{mn}} \sum_{s\in\{0,1\}}^{\log(m)} \sum_{t\in\{0,1\}}^{\log(n)} (-1)^{\chi_{Cyc(x,y)}(s,t)} |s\rangle \otimes |t\rangle \otimes \frac{1}{\sqrt{m}} \sum_{i=0}^{m} |x\rangle \otimes \frac{1}{\sqrt{n}} \sum_{i=0}^{n} |y\rangle\,.
\end{aligned}
$$

Therefore, after the uncomputing step our register is unchanged except for the flipped phase of the states $|s\rangle\,|t\rangle$ corresponding to a pair (s,t) that is a solution $\chi_{Cyc(x,y)}$, that is we have proved that the rotating, the matching, and the uncomputing step implements the phase oracle for $\chi_{Cyc(x,y)}$, which is what we wanted to prove for correctness (see 4.2).

4.6 Time Complexity

The overall circuit implementing the phase oracle for $\chi_{Cyc(x,y)}$ has depth $\mathcal{O}(\log^2(n) + \log(m) = \mathcal{O}(\log^2(n))$. Since Grover's Diffuser has depth $\mathcal{O}(\log(m))$, and because we need $\lfloor \frac{n}{\pi}\sqrt{mn} \rfloor$ Grover's iterations, the circuit for cyclic string matching presented has depth equal to $\mathcal{O}\left(\sqrt{mn}(\log^2(n) + \log(m))\right) = \mathcal{O}\left(\sqrt{nm}(\log^2(n))\right)$. Using the big-$\tilde{\mathcal{O}}$-notation, which works as the big-$\mathcal{O}$-notation except for hiding polylogarithmic factors, we can write that the depth of the presented circuit presented equals $\tilde{\mathcal{O}}\left(\sqrt{nm}(\log^2(n))\right)$. Therefore, we get an almost quadratic speed-up against the most time-efficient algorithm for the problem considered.

In the appendix, we provide a practical implementation of the algorithm using Qiskit, an open-source toolkit developed by IBM and based on the Python language. The code shown in this section is available to browse and run within a public Google Colab Tutorial.[7]

5 Conclusions

In this paper, we have investigated the quantum cyclic string matching problem, presenting a novel approach that leverages quantum computing principles to enhance

[7] `https://colab.research.google.com/drive/-1bbRjsYl7UCVT6P4gNwJulARfd64L1RCT?usp=sharing`

computational efficiency. We began by discussing the foundational concepts of string matching, both exact and approximate, and highlighted the significance of cyclic strings in various applications, from bioinformatics to data compression and pattern recognition.

Our primary contribution is a quantum algorithm for cyclic string matching, building on the work of Niroula and Nam [21]. This algorithm utilizes the circuit-based quantum model and incorporates cyclic rotations to achieve a superposition of all possible alignments of the pattern within the text. Implemented using Qiskit, our method offers significant computational efficiency, with a complexity of $\tilde{\mathcal{O}}(\sqrt{nm})$, providing a substantial speed-up over classical algorithms.

As quantum hardware and algorithms continue to evolve, the integration of quantum computing into broader scientific and industrial applications appears increasingly promising. Our work demonstrates the potential of quantum computing to redefine problem-solving paradigms in string matching, paving the way for future innovations and applications in this and related fields.

References

1. A. Ambainis: *Quantum search algorithms.* SIGACT News, 35(2) 2004, pp. 22–35.
2. S. Balauca and A. Arusoaie: *Efficient constructions for simulating multi controlled quantum gates*, in Computational Science - ICCS 2022, vol. 13353 of LNCS, Springer, 2022, pp. 179–194.
3. R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf: *Quantum lower bounds by polynomials.* J. ACM, 48(4) 2001, pp. 778–797.
4. M. Boyer, G. Brassard, P. Høyer, and A. Tapp: *Tight bounds on quantum searching.* Fortschritte der Physik, 46(4-5) 1998, pp. 493–505.
5. G. Brassard, P. Høyer, M. Mosca, and A. Tapp: *Quantum amplitude amplification and estimation*, in Quantum Computation and Information, S. G. Lo Monaco and H. E. Brandt, eds., vol. 305 of Contemporary Mathematics, American Mathematical Society, 2002, pp. 53–74.
6. G. Buonaiuto, R. Guarasci, A. Minutolo, G. D. Pietro, and M. Esposito: *Quantum transfer learning for acceptability judgements.* Quantum Mach. Intell., 6(1) 2024, p. 13.
7. E. T. Campbell, A. Khurana, and A. Montanaro: *Applying quantum algorithms to constraint satisfaction problems.* Quantum, 2018.
8. D. Cantone, S. Faro, and A. Pavone: *Quantum string matching unfolded and extended*, in Reversible Computation - Proceedings, vol. 13960 of LNCS, Springer, 2023, pp. 117–133.
9. R. Dulbecco and M. Vogt: *Evidence for a ring structure of polyoma virus dna.* Proceedings of the National Academy of Sciences, 50(2) 1963, pp. 236–243.
10. M. Fang, S. Fenner, F. Green, S. Homer, and Y. Zhang: *Quantum lower bounds for fanout.* Quantum Info. Comput., 6(1) jan 2006, pp. 46–57.
11. S. Faro and T. Lecroq: *The exact online string matching problem: A review of the most recent results.* ACM Comput. Surv., 45(2) 2013, pp. 13:1–13:42.
12. S. Faro, F. P. Marino, and A. Pavone: *Efficient online string matching based on characters distance text sampling.* Algorithmica, 82(11) 2020, pp. 3390–3412.
13. S. Faro, A. Pavone, and 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, X. Chen and B. Li, eds., vol. 14637 of Lecture Notes in Computer Science, Springer, 2024, pp. 247–259.
14. 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, 1996, pp. 212–219.
15. P. Høyer and R. Spalek: *Quantum fan-out is powerful.* Theory C., 1 2005, pp. 81–103.
16. Y. Lao and O. Ait-Aider: *Rolling shutter homography and its applications.* IEEE Trans. Pattern Anal. Mach. Intell., 43(8) 2021, pp. 2780–2793.
17. F. Lisacek: *Algorithms on strings, trees and sequences: Dan gusfield.* Comput. Chem., 24(1) 2000, pp. 135–137.

18. M. Lothaire: *Applied Combinatorics on Words*, Cambridge University Press, 2005.
19. A. Montanaro: *Quantum pattern matching fast on average.* Algorithmica, 77(1) 2017, pp. 16–39.
20. D. R. Musser: *Introspective sorting and selection algorithms.* Softw. Pract. Exp., 27(8) 1997, pp. 983–993.
21. P. Niroula and Y. Nam: *A quantum algorithm for string matching.* npj Quantum Information, 7 02 2021, p. 37.
22. A. Pavone and C. Viola: *The quantum cyclic rotation gate*, in Proceedings of the 24th Italian Conference on Theoretical Computer Science (ITCTS 2023), vol. Vol-3587, Italy, September 13-15 2023, University of Palermo, pp. 206–218.
23. H. Ramesh and V. Vinay: *String matching in $\mathcal{O}(n+m)$ quantum time.* Journal of Discrete Algorithms, 1(1) 2003, pp. 103–110.
24. S. E. Rasmussen, K. Groenland, R. Gerritsma, K. Schoutens, and N. T. Zinner: *Single-step implementation of high-fidelity n-bit Toffoli gates.* Phys. Rev. A, 101 Feb 2020, p. 022308.
25. P. W. Shor: *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer.* SIAM J. Comp., 26(5) 1997, pp. 1484–1509.
26. M. Thanbichler, S. Wang, and L. Shapiro: *The bacterial nucleoid: A highly organized and dynamic structure.* Journal of cellular biochemistry, 96 10 2005, pp. 506–21.
27. C. Viola and S. Živný: *The Combined Basic LP and Affine IP Relaxation for Promise VCSPs on Infinite Domains.* ACM Trans. Algorithms, 17(3) 2021, pp. 21:1–21:23.
28. R. Weil and J. Vinograd: *The cyclic helix and cyclic coil forms of polyoma viral dna.* Proceedings of the National Academy of Sciences, 50(4) 1963, pp. 730–738.
29. W. K. Wootters, W. K. Wootters, and W. H. Zurek: *A single quantum cannot be cloned.* Nature, 299 1982, pp. 802–803.

A A Qiskit Implementation

Initially, we introduce the fundamental structure of the algorithm and the implementation of the involved operators, as outlined in the reference provided. Subsequently, we delve into more specific algorithmic aspects that enable us to tackle the problem under broader conditions. The code discussed in this study is readily accessible online for consultation and execution via a Colab document.

```
def rot(n, s):
  qc = QuantumCircuit(n)
  for i in range(1, log2(n)-log2(s)+2):
    for j in range(0, n/(s*(2**i))):
      for q in range(j*s*2**i,s*(j*2**i+1)):
        qc.swap(n-1-(q+s), n-1-(q+2**(i-1)*s+s))
  return qc.to_gate(label='ROT'+str(s))

def rot_gate(n,s):
  return rot(n,s).to_gate(label='Rot'+str(s))
```

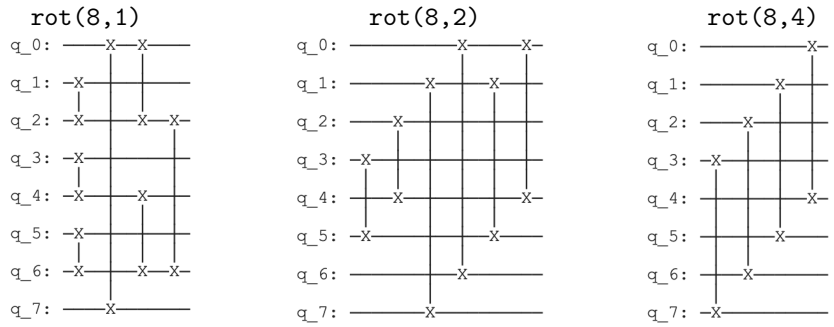

Figure 7. On top: the Qiskit procedure `rot(n,s)` for implementing the left-rotation quantum operator which cyclically rotate a register with size `n` of `s` positions to the right. On bottom: three leftward rotation operators generated by the procedure for a quantum register of 8 qubits and shift amounts equal to 1, 2 and 4 positions, respectively.

```
def parametrized_rot(n):
  logn = int(np.log2(n))
  kr = QuantumRegister(logn,'k')
  yr = QuantumRegister(n,'y')
  qc = QuantumCircuit(kr,yr)
  for i in range(logn):
    qc = qc.compose(rot_gate(n,2**i).control(1), [kr[i]]+yr[:])
  return qc
```

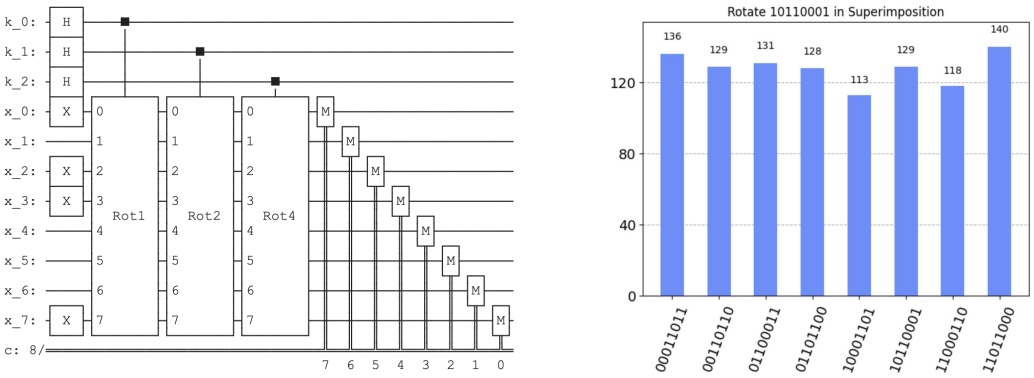

Figure 8. An implementation of the circuit for obtaining a superposition of all the cyclic shifts of an 8-qubit register obtained via the controlled cyclic shift operator.

```
def match(n):
  xr = QuantumRegister(n,'x')
  yr = QuantumRegister(n,'y')
  out = QuantumRegister(1,'out')
  qc = QuantumCircuit(xr,yr,out)
  for i in range(n):
    qc.cx(xr[i],yr[i])
    qc.x(yr[i])
  qc.mcx(yr[:],out)
  for i in range(n):
    qc.x(yr[i])
    qc.cx(xr[i],yr[i])
  return qc

def match_gate(n):
  return match(n).to_gate(label='Match'+str(n))
```

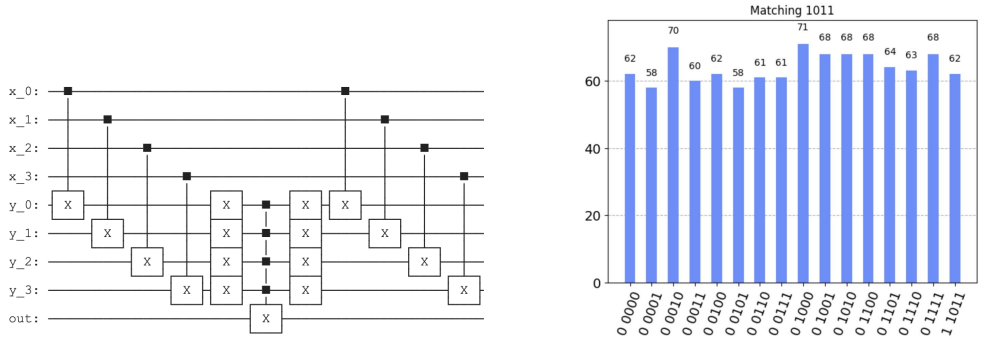

Figure 9. An implementation of the quantum gate checking for a match of two strings, each encoded by a 4-qubit register.

```
def csmo(m,n):
  logm = int(np.log2(m))
  logn = int(np.log2(n))
  ir = QuantumRegister(logm,'i')
  jr = QuantumRegister(logn,'j')
  xr = QuantumRegister(m,'x')
  yr = QuantumRegister(n,'y')
  out = QuantumRegister(1,'out')
  qc = QuantumCircuit(ir,jr,out,xr,yr)
  qc = qc.compose(parametrized_rot(m), ir[:]+xr[:])
  qc = qc.compose(parametrized_rot(n), jr[:]+yr[:])
  qc = qc.compose(match_gate(m), xr[:]+yr[:m]+out[:])
  qc = qc.compose(parametrized_rot(n).inverse(), jr[:]+yr[:])
  qc = qc.compose(parametrized_rot(m).inverse(), ir[:]+xr[:])
  return qc
```

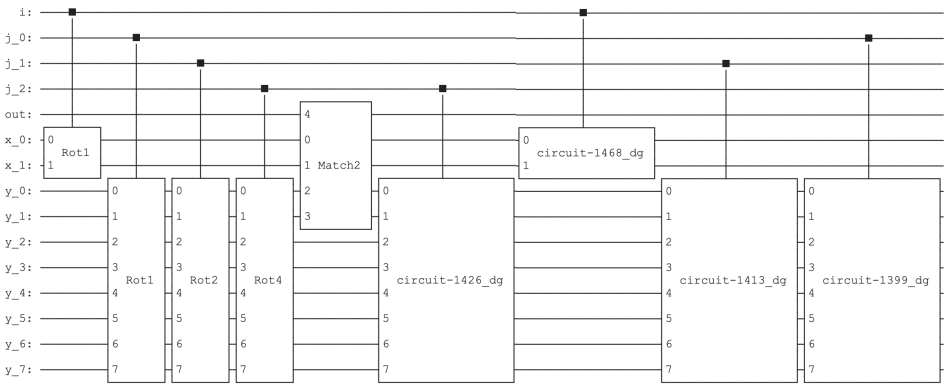

Figure 10. Circuital implementation of Grover's search algorithm for identifying a cyclic occurrence of a pattern `x` of length 2 within a text `y` of length 8.

```
def GroverCSM(x,y,t):
  m = len(x)
  n = len(y)
  logm = int(np.log2(m))
  logn = int(np.log2(n))
  ir = QuantumRegister(logm,'i')
  jr = QuantumRegister(logn,'j')
  xr = QuantumRegister(m,'x')
  yr = QuantumRegister(n,'y')
  out = QuantumRegister(1,'out')
  icr = ClassicalRegister(logm,'ic')
  jcr = ClassicalRegister(logn,'jc')
  qc = QuantumCircuit(ir,jr,out,xr,yr,icr,jcr)
  qc = qc.compose(register_initialize(x), xr[:])
  qc = qc.compose(register_initialize(y), yr[:])
  qc.h(ir)
  qc.h(jr)
  qc.x(out)
  qc.h(out)
  iterations = int((np.pi/4)*(np.sqrt(n*m/t)))
  for i in range(iterations):
    qc = qc.compose(csmo_gate(m,n), ir[:]+jr[:]+out[:]+xr[:]+yr[:])
    qc = qc.compose(diff(logm+logn), ir[:]+jr[:])
  for i in range(logm):
    qc.measure(ir[i], icr[i])
  for i in range(logn):
    qc.measure(jr[i], jcr[i])
  return qc
```

Figure 11. Qiskit implementation of Grover's search algorithm for identifying a cyclic occurrence of a pattern `x` of length `m` within a text `y` of length `n`. The implementation assumes that both strings are binary sequences and that the number `t` of occurrences of `x` in `y` is known.

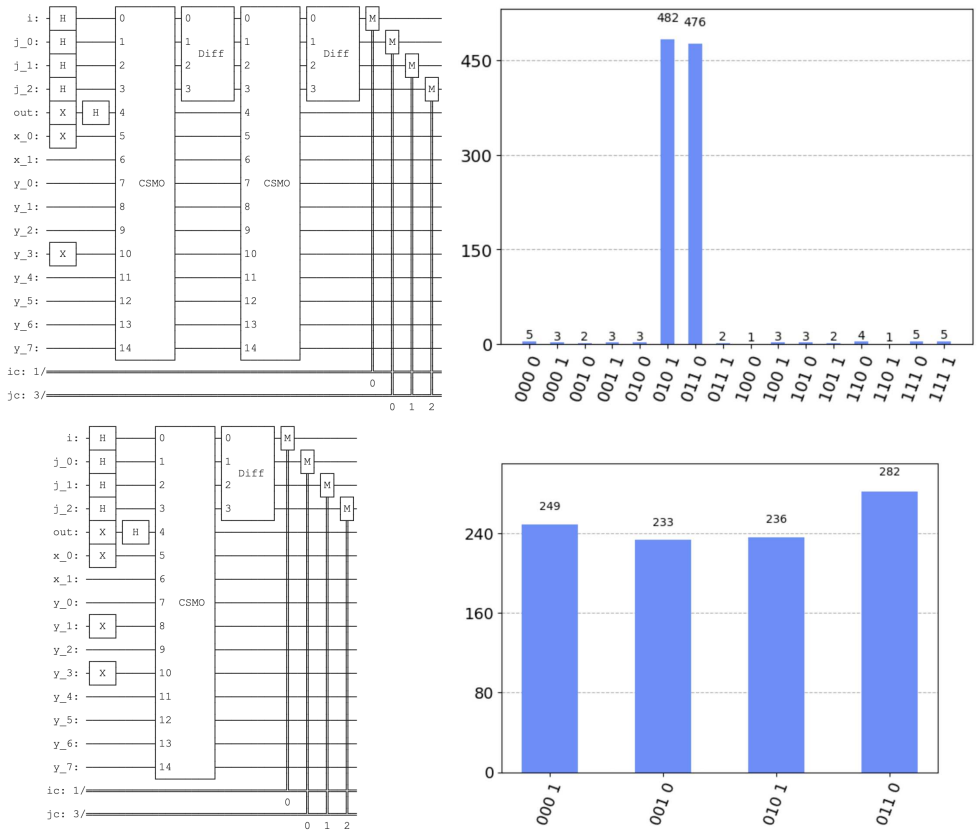

Figure 12. Two runs of the quantum Cyclic String Matching algorithm and the graph of the related results. Above: searching for the string `10` within the text `00010000`. There are 2 occurrences, in position 2 with rotation equal to 1, and in position 3 with rotation equal to 0. Two executions of Grover's algorithm are necessary to identify one of the two occurrences. Bottom: Searching for the string `10` within the text `01010000`. There are 4 occurrences, in positions 0,1,2 and 3 with rotation equal to 1, 0, 1 and 0, respectively. In this case, a single iteration of Grover's algorithm is sufficient to identify one of the 4 solutions.