Quantum Information Processing (2024) 23:194
https://doi.org/10.1007/s11128-024-04388-9

An algorithm based on quantum phase estimation for the identification of patterns

Dimitris Ntalaperas[1] · **Andreas Kalogeropoulos**[1] · **Nikos Konofaos**[1]

Received: 21 December 2023 / Accepted: 11 April 2024 / Published online: 16 May 2024

Abstract
The quantum phase estimation algorithm has been utilized by a variety of quantum algorithms, most notably Shor's algorithm, to obtain information regarding the period of a function that is appropriately encoded into a unitary operator. In many cases, it is desired to estimate whether a specific state exhibits a certain pattern quickly. In this paper, we exhibit a methodology based on the QPE algorithm to identify certain patterns. In particular, starting from a properly encoded state, we demonstrate how to construct unitary operators whose eigenvectors correspond to states with proper diagonals. QPE will then output an eigenphase equal to zero with certainty for these states, thereby identifying this class of matrices. For partial matches, our algorithm, based on the tolerance threshold used, will show areas of high similarity, and it will outperform classical ones when the number of mismatches defined by the tolerance is significantly low when compared to the size of the diagonal.

Keywords Quantum pattern matching · Quantum phase estimation · Quantum computing · Shor's algorithm

1 Introduction

In many applications, we are interested in determining some characteristics of a dataset, such as if it contains certain pre-defined patterns. For example, when searching for large text data, a dot plot matrix [1] can be constructed to indicate the positions of the

Dimitris Ntalaperas, Andreas Kalogeropoulos and Nikos Konofaos contributed equally to this work.

✉ Dimitris Ntalaperas
ntalaperas@csd.auth.gr

Andreas Kalogeropoulos
agkaloger@csd.auth.gr

Nikos Konofaos
nkonofao@csd.auth.g

[1] Computer Science Department, Aristotle University of Thessaloniki, Thessaloniki 54124, Kentriki Makedonia, Greece

matches. In this dot plot matrix, diagonals will indicate matches; if there is a quick algorithm to identify whether diagonals exist, it can be used to isolate areas of interest in the input data; Schutzhold [2], for example, designed a quantum algorithm that can be used to identify the locations of patterns within such a structure, while Prousalis and Konofaos [3] have shown that pairwise sequence alignment can be improved using quantum pattern recognition algorithms in dot plots. In general, even at the classical level, many problems can be solved by mapping instances of the problem into an operator and by solving the problem of estimating the eigenvalues of a given operator in its abstract form [4, 5] or in a matrix representation form [6]. Generally, the perturbations of unitary eigenvalues [7] and the repercussions of their physical manifestations are a topic of wide interest [8].

In the present work, we present a method that uses the quantum phase estimation (QPE) algorithm [9] to try to find patterns in the aforementioned fashion that identify matrices that contain proper diagonals. The methodology is based on a modification of the traditional QPE; whereas in the traditional QPE, we compute the eigenphases of the states, in our approach, the input state can be arbitrary. Our algorithm constructs a unitary operator U that has eigenvectors that correspond to states that encode perfect diagonals. We show that when we apply QPE to this operator and with an input state that encodes a perfect diagonal, our algorithms will produce, as expected, an eigenphase with perfect certainty, thus correctly identifying the pattern. Moreover, when the state differs slightly from an eigenstate, our algorithm produces values close to the eigenphase with a bounded error. Thus, it can be used to identify partial matches and areas of imperfect matches. These imperfect matches may have applications when we are interested in sequences of data that have common characteristics, such as in multiple sequence alignment (MSA) [10] where we are interested in relating sequences that indicate similar functions or common ancestry and can therefore exhibit variations in certain locations. Though our approach is mainly targeted at matrix diagonals in the present work, it should be pointed out that the methodology finds applications in pairwise sequence alignment (PSA) where we are interested in relating two sequences where dot-matrix representations have been used in classical techniques such as the Needleman–Wunsch (NW) [11] and Smith-Waterman (SW) algorithms [12]. In these algorithms, each element of the matrix is assigned a specific score, based on a dynamic programming routine, and the alignment is found after a back-tracking technique. The proposed method—even in the classical sense—is by definition faster, since it does not have to back-track to identify the alignment. NW [13] and SW [14] algorithms are often used in combination with MSA as an alignment refiner. Furthermore, our methodology can be extended to handle different kinds of patterns with the proper redefinition of the operator, e.g., in typical pattern recognition where exact matches are of interest.

In the landscape of quantum algorithms, our algorithm falls within the category of quantum pattern matching, for which several approaches have already been established, based both on methods extending Grover's algorithm for searching in an unsorted database [15] and on more modern approaches-based quantum machine learning techniques.

The work on this document is structured in the following manner: Sect. 2 gives an overview of the existing theory and specifically on the core aspects of Shor's

algorithm that are being used in our work. Section 3 then describes the main idea behind the proposed methodology in terms of the algorithm description and the operator definition. Section 4 describes the case of partial matches in the compared data strings and how the algorithm can still be applied to identify areas of high correlation between the source and target data. Some examples in the form of test runs are conducted in Sect. 5, which show how the algorithm would behave in a noise-free setting in the different scenarios of perfect and imperfect matches. An analysis of the performance of the algorithm in terms of the running time is given in Sect. 6; this section also includes a comparative analysis with classical algorithms for sequence alignment and with other quantum computing approaches for pattern matching. An analysis for the number of one and two qubits required for transpiling our circuit in modern hardware, together with an estimation of the accumulated error and some targeted mitigation measures is listed in Sect. 7. Finally, in Sect. 8, the results of the present work are summarized together with a discussion on how to extend these to different patterns.

2 Theory

The algorithm presented in our work is based on Shor's algorithm [16]; for completeness, the most basic parts of Shor's algorithm that are also relevant to the present work will be briefly highlighted.

One of the main breakthroughs of Shor's algorithm was that it implements a polynomial algorithm to find the period r of a periodic function f, defined by:

$$f(x) = a^x \bmod N. \tag{1}$$

The period r is the smallest integer such that:

$$a^r \bmod N = 1. \tag{2}$$

Shor's algorithm computes the period by first defining a unitary operator U such that:

$$U|y\rangle = |ay \bmod N\rangle. \tag{3}$$

Due to the periodicity of the mod function, this operator, when applied successively for r iterations, cycles an eigenstate of U back to itself. For example, with $a = 3$, $N = 35$ and if we start with state $|1\rangle$ we have:

$$\begin{aligned} U^1|1\rangle &= |3\rangle \\ U^2|1\rangle &= |9\rangle \\ U^3|1\rangle &= |27\rangle \\ &\vdots \\ U^{r-1}|1\rangle &= |12\rangle \\ U^r|1\rangle &= |1\rangle \end{aligned} \tag{4}$$

It is straightforward to verify that if U is applied on a superposition of all the states in the above cycle, the superimposed state defined by:

$$|u_0\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |a^k \bmod N\rangle \tag{5}$$

would also be an eigenstate of U, with an eigenvalue equal to one

For example, for the cyclic state:

$$|u_0\rangle = \frac{1}{\sqrt{12}}(|1\rangle + |3\rangle + |9\rangle + \cdots + |4\rangle + |12\rangle) \tag{6}$$

if the operator acts on $|u_0\rangle$, it will produce:

$$\begin{aligned} U|u_0\rangle &= \frac{1}{\sqrt{12}}(U|1\rangle + U|3\rangle + U|9\rangle + \cdots + U|4\rangle + U|12\rangle) \\ &= \frac{1}{\sqrt{12}}(|3\rangle + |9\rangle + |27\rangle + \cdots + |12\rangle + |1\rangle) \\ &= |u_0\rangle. \end{aligned} \tag{7}$$

thus making $|u_0\rangle$ an eigenstate of U, with eigenvalue equal to one. The above process can be repeated if we allow the superposition of states that have a relative phase. In the general case, $|u_s\rangle$ can then be defined to be of the form:

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi sk}{r}} |a^k \bmod N\rangle \tag{8}$$

In a similar fashion, when U is applied to $|u_s\rangle$, we obtain:

$$U|u_s\rangle = e^{\frac{2\pi s}{r}} |u_s\rangle \tag{9}$$

It can be seen that for $0 \leq s \leq r-1$, all u_s are composed of states that, except state $|1\rangle$, which is formed for k=0, pick up phases that correspond to the r-roots of unity. Choosing a smaller example for the purposes of depiction with $a = 7$, $N = 15$ (in which $r = 4$), and ignoring the overall normalization factor of $1/\sqrt{4}$, the eigenstates of U can be formed as:

$$\begin{aligned} |u_0\rangle &= |1\rangle + |7\rangle + |4\rangle + |13\rangle \\ |u_1\rangle &= |1\rangle + e^{-\frac{2\pi}{4}}|7\rangle + e^{-\frac{4\pi}{4}}|4\rangle + e^{-\frac{6\pi}{4}}|13\rangle \\ |u_2\rangle &= |1\rangle + e^{-\frac{4\pi}{4}}|7\rangle + e^{-\frac{8\pi}{4}}|4\rangle + e^{-\frac{12\pi}{4}}|13\rangle \end{aligned}$$

$$|u_3\rangle = |1\rangle + e^{-\frac{6\pi}{4}}|7\rangle + e^{-\frac{12\pi}{4}}|4\rangle + e^{-\frac{18\pi}{4}}|13\rangle \tag{10}$$

where we have omitted simplification in the exponents to demonstrate the relevant rotation in each successive s. Adding all these states, we can note two things:

- The states add up to the state $|1\rangle$
- The superposition consists of an equal superposition of the eigenstates of U, each one with eigenphase equal to $\phi = \frac{s}{r}$.

Thus, if we perform the QPE algorithm on the U operator with an initial state equal to $|1\rangle$ and measure the result, we will obtain $\phi = \frac{s}{r}$ with equal probability for each s; from this result, r can then easily be obtained by the continuous fraction algorithm.

3 Methodology

As mentioned, the motivation behind the proposed algorithm lies in applying a QPE pipeline in the dataset to gauge the existence of potential patterns. The following Sects. 3.1 and 3.2 explain the motivation and execution of the algorithm, respectively. Section 3.3 shows how the operator U can be constructed.

3.1 Motivation

In its simplest form, QPE works by assuming that an eigenstate $|\psi\rangle$ of a unitary operator U has been prepared. If the state is encoded in a register of size m and a register of size n is used to store the phase estimation, then QPE works by first performing a Hadamard operation on the output registered followed by a series of $controlled - U$ operations:

$$|0\rangle^{\otimes n}|\psi\rangle^{\otimes n} \xrightarrow{H^{\otimes n} I^{\otimes m}} \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle|\psi\rangle \xrightarrow{C-U^{2^j m \otimes n}} \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} e^{2\pi j\theta}|j\rangle|\psi\rangle. \tag{11}$$

Knowing that a pattern exists and searching for characteristic values is the way that Shor's algorithm works (i.e., we know that a period should exist, and we try to find its value). In many problems, the reverse problem arises, i.e., trying to verify if certain patterns exist.

The core idea of our approach is to apply QPE to a state and check if a pure phase is an output. If it is, then the state has the pattern that we search for. We thus need to solve the following:

Problem definition*: Let a pattern P, be a perfect pattern and let D be a dataset. Construct U such that its eigenstates correspond to representations of D that exhibit the pattern P.*

Our work will focus on the case where the pattern is perfect diagonals, and we will provide a brief discussion of how it can be extended to general use cases in Sect. 8.

3.2 Algorithm

The main goal of the algorithm is to construct an operator that has eigenstates that correspond to diagonal states, where here we assume the simplest case of square matrices with proper diagonals for simplicity.

We start the process by defining a sufficient representation of the problem: Define an $N \times N$ matrix which contains N^2 elements. Matrices can be encoded in a quantum register of size equal to $\log N^2 = 2 \log N$, if we follow the simple scheme of basis state encoding (i.e., encoding the various matrices in base states of the quantum register; see [17] for a review of the most common quantum encoding schemes).
For example, for $N = 2$ we have the following schema:

$$\begin{aligned}
\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} &\rightarrow |00\rangle \\
\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} &\rightarrow |01\rangle \\
\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} &\rightarrow |10\rangle \\
\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} &\rightarrow |11\rangle. \qquad (12)
\end{aligned}$$

where we have used Qiskit's [18] convention for numbering indexes in registers, with the rightmost qubit denoting the least significant bit.

Arbitrary matrices can, of course, be encoded via superpositions, e.g.,

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \rightarrow |00\rangle + |11\rangle. \qquad (13)$$

where the normalization factors have been omitted.

We are now defining the operator U:

$$U|x\rangle = |(x + N + 1) \bmod (N(N + 1))\rangle, \qquad (14)$$

with $x \in \{0, 1, \ldots, N - 1\}$.

Intuitively, this operator adds $N + 1$ to each state; with N+1 being the distance between diagonal elements, it can be seen that this operator "moves" diagonal elements down a row. The mod operation is present to ensure that the last element is cycled back to the first place. For the simple $N = 2$ case, the operator is defined as:

$$U|x\rangle = |(x + 3) \bmod 6\rangle \qquad (15)$$

and its operation on the diagonal elements can be depicted as:

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \rightarrow |00\rangle \xrightarrow{U} |11\rangle \rightarrow \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \rightarrow |11\rangle \xrightarrow{U} |00\rangle \rightarrow \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}. \tag{16}$$

where in the 2nd equation the element $|11\rangle = |3\rangle$ was mapped to $|(3+3) \bmod 6\rangle = |00\rangle$.

An obvious eigenstate of operator U is the one taken from a superposition of states that single out diagonal elements. This superposition, which we denote by $|d\rangle$, maps a proper diagonal matrix (equal to the identity matrix in the simple case with 0/1 elements discussed here), such as the example in Eq. 13. If U is applied to $|d\rangle$, it leaves the phase invariant. Thus, $|d\rangle$ is an eigenstate of U with an eigenvalue(eigenphase) equal to one(zero).

3.3 Construction of U

It can be seen both intuitively and by the precise definition of U that the operator basically performs a permutation of the basis states, as it adds the number $N + 1$ to all states and cycles the last N states back to the first ones.
Using pseudocode, the operator U can be dynamically constructed using the code depicted in algorithm listing 1 (please refer to code listing 1 in "Appendix A" for the Python code using IBM's Qiskit framework):

Algorithm 1 Compute the controlled-U operator

```
N ⇐ size^2
U ⇐ Matrix(N, N)
modulo ⇐ size mod size * (size − 1)
permutation ⇐ 0
for i do between (modulo, (size*(size-1) + modulo):
    U[i mod (size * (size − 1)] = 1
    permutation ⇐ permutation + 1
end for
for r do between (size*(size-1), N);
    U[r][r] ⇐ 1
end for
return U
```

It is to be noted that for the operator U to be able to cycle back the state of the last element to the first position, it should, by its definition, act on a bigger space than that corresponds to the matrix states (an example can be shown in Fig. 1). By composing the above operator, we can construct the QPE circuit, depicted in Fig. 2 for the 4×4 case. The *Encoder* sub-circuit is responsible for preparing the input state by performing basic encoding (see Listing 2 in "Appendix A" for sample code that creates the appropriate superposition). This sub-circuit is non-unitary and is presented here for simulation needs. In a real-world scenario, the encoding would be the outcome of a quantum algorithm that would encode the input data as necessary. (See [19] for an example of how to encode dot plots that originate from genetic sequence data).

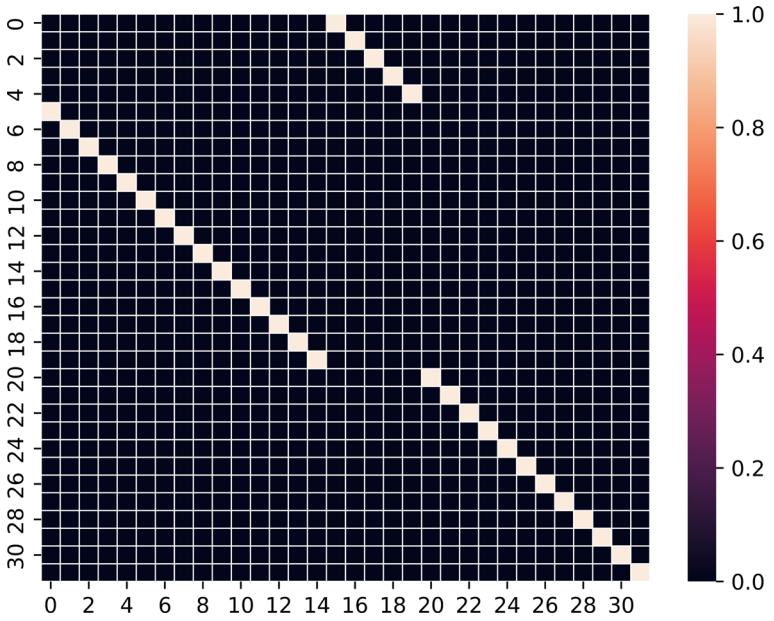

Fig. 1 Example of an operator U acting on a 16 × 16 state

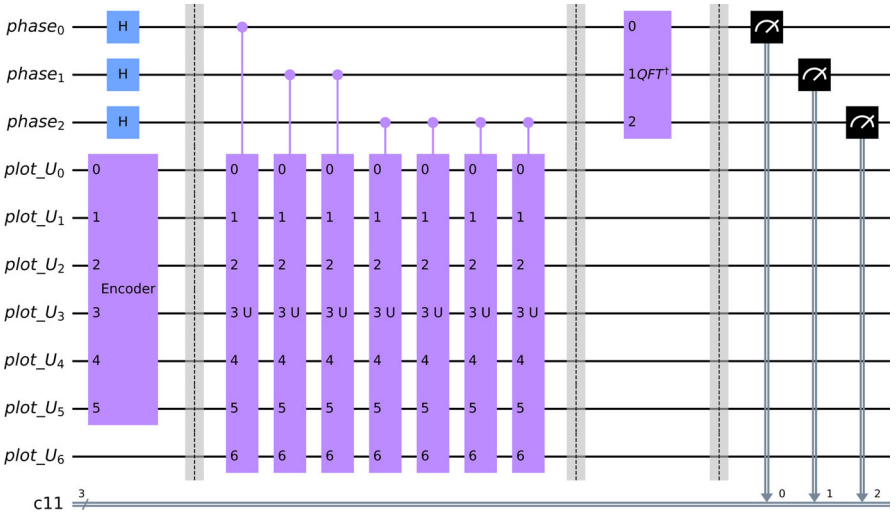

Fig. 2 Example circuit for estimating whether there are main diagonals in the input matrix

Assuming a proper encoding of a state that represents a matrix with a proper main diagonal, the operation of the circuit is straightforward: The state is an eigenstate of operator U with an eigenvalue equal to one; therefore, the circuit will produce the eigenphase 0 with 100% probability. In a real use case scenario, if a batch process

produces dot plots by comparing test and target datasets, a quantum routine can classify which parts of the compared datasets have a perfect match.

4 Partial matches

In a typical real-world scenario, data will rarely demonstrate perfect matches. Patterns may often exhibit partial matches that will appear as diagonals with gaps. In contrast, many patterns may also exhibit off-diagonal matches; in a text search problem, for example, matches can occur in an off-diagonal position clearly due to chance; the smaller the alphabet, the higher the chance of random off-diagonal matches. In such cases, we would like to isolate areas that have a big correlation. Although the algorithm will not produce a zero phase result deterministically, it can be seen that states that are close to proper diagonals have a big overlap with states that correspond to proper diagonals states that have already been studied.

More specifically, let $|d\rangle$ be the eigenstate corresponding to a full-match main diagonal, i.e.,

$$|d\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j(N+1)\rangle = |0\rangle + |N+1\rangle + \cdots + |(N-1)(N+1)\rangle \tag{17}$$

and let $|d_m\rangle$ be an eigenstate corresponding to a partial-match main diagonal, i.e., a diagonal with m gaps/mismatches

$$|d_m\rangle = \frac{1}{\sqrt{N-m}} \sum_{j=0}^{N-1} \sum_{\mu=1}^{m} \delta_{\mu j} |j(N+1)\rangle \tag{18}$$

with $\delta_{\mu j}$ the Kronecker delta. Then, exploiting the orthogonality property $\langle i|j\rangle = \delta_{ij}$, we obtain

$$\langle d_m|d\rangle = \frac{1}{\sqrt{N}} \frac{1}{\sqrt{N-m}} \sum_{j=0}^{N-1} \sum_{\mu=1}^{m} \delta_{\mu j} = \sqrt{1 - \frac{m}{N}} \tag{19}$$

The last relation clearly implies that for $m << N$, we expect the phases projected by the QPE to be close to 0, while for a non-negligible number of gaps, m, we expect them to deviate significantly from 0. Equation 19 reveals that if ω is the phase corresponding to the overlap between $|d_m\rangle$ and $|d\rangle$ it holds that

$$\omega = \arcsin\left(\sqrt{\frac{m}{N}}\right) \tag{20}$$

The total measure of the overlap can be retrieved by taking the norm of the inner product defined in (19):

$$|\langle d_m|d\rangle|^2 = 1 - \frac{m}{N} \tag{21}$$

Next, we consider the case of a "noisy" pattern. Such a pattern consists of a perfect match that is accompanied by off-diagonal matches. Such mismatches can occur commonly in practice due to randomness, as off-position letters can match at various positions. In practice, the larger the alphabet is, the rarer these off-diagonal occurrences will be. Amino acid sequences, for example, will have much less "noise" than DNA or RNA sequences. For these patterns, the state will be represented by the following state $|\Psi_m\rangle$

$$|\Psi_m\rangle = \frac{1}{\sqrt{N+m}} \left(\sum_{j=0}^{N-1} |j(N+1)\rangle + \sum_{\mu=1}^{m} |j(N+1)+k_m\rangle \right) \tag{22}$$

with k_m representing the entries $|k\rangle$ corresponding to "noise." By the same technique as before, we arrive at

$$1 - \frac{m}{N+m} \leq |\langle \Psi_m|d\rangle|^2 \leq 1 - \frac{1}{N+m}. \tag{23}$$

depending on which of the m non-diagonal matches $|k_m\rangle$ are mapped by U to their diagonally subsequent entries $|k_m + N + 1\rangle$.

Combining the two terms, the one corresponding to errors due to gaps with the overlap defined in Eq. 21 and the one corresponding to errors due to mismatch defined in Eq. 23, we can state the following:

Theorem 1 *Let a dot plot contain k gaps and l off-diagonal matches for a total of $m = k + l$ mismatches. If $|d_{k,l}\rangle$ is the state that encodes the dot plot, and $|d\rangle$ is the state that encodes the perfect dot plot, the two states differ with a factor that scales as $O(\frac{m}{N})$.*

Based on Theorem 1, we can define tolerance thresholds that can be used to classify whether a sampling output corresponds to a match or not. These thresholds will depend on the use case at hand.

5 Example runs

In this section, we will investigate some examples run on Qiskit simulation that depict the various cases that can be encountered in practice and how the output can be evaluated for match or mismatch. We will use the case of an 8×8 matrix and consider the case of a proper diagonal, of gaps in the diagonal and of off-diagonal matches. The circuit to be run for all cases is depicted in Fig. 3. Each simulation consisted of 1024 shots and was performed by the Aer simulator of Qiskit with a zero noise model.

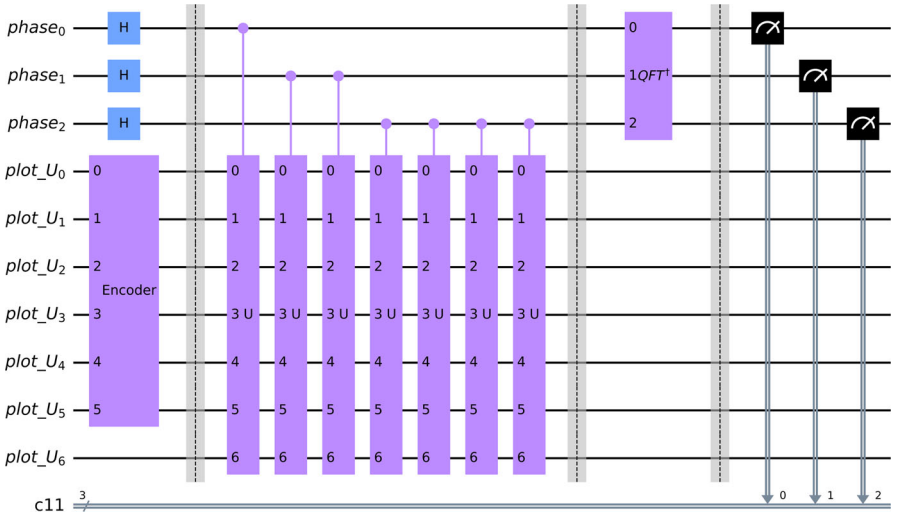

Fig. 3 Circuit for detecting perfect matches on proper diagonals in an 8×8 matrix

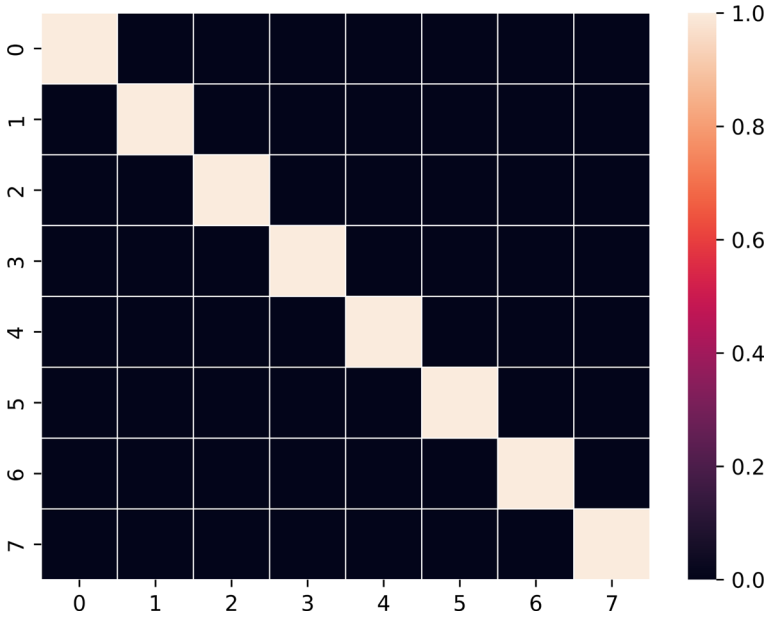

Fig. 4 Heatmap of a dot plot containing a proper diagonal

5.1 Perfect diagonal

Figure 4 depicts a heatmap corresponding to a dot plot with proper diagonal. Figure 5 depicts the results of the simulation. As expected, all runs produced a phase equal to zero, indicating the presence of a proper diagonal. Although 1024 shots were used, it

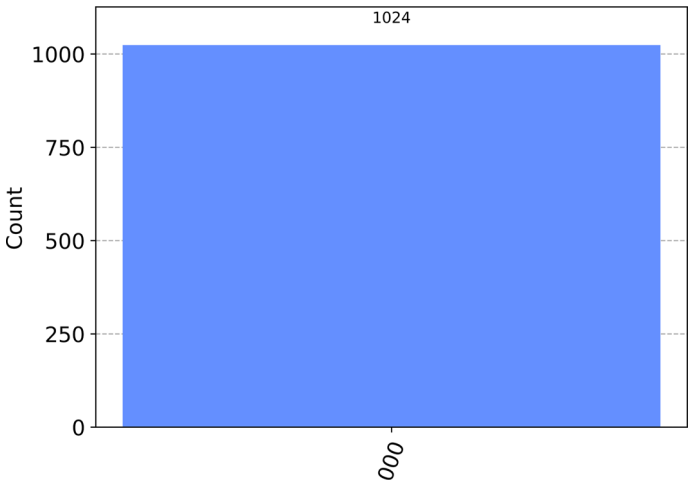

Fig. 5 Simulation results for a proper diagonal

is clear that when there is a perfect diagonal, very few runs are needed to identify that there is a perfect match. The reason for this is that, in a perfect match, the algorithm produces the value of zero with certainty, and we only need to exclude the case of a false positive (i.e., a result of zero occurring when there is not a match). This can be estimated as:

$$P(0|not\ perfect\ match) \approx \omega^t \tag{24}$$

where ω, defined in Eq. 20, is the overlap of the state with the state $|d\rangle$ corresponding to a state of perfect diagonal and t the number of precision qubits that are used to measure the eigenphase of the operator. The larger the distance between the input state and the zero states is, the lower the measure of the overlap ω is, and thus, the closer the probability is to zero. This relation is further amplified by an increased number of t.

5.2 Gaps in the main diagonal

For a gap in the main diagonal, consider the example depicted in the heatmap in Fig. 6. There is one gap, so the mismatch, according to Eq. 21, is expected to be around $1 - \frac{1}{8} = 0.875$. Figure 7 depicts the results of the simulation. The result indicates a state resembling one that is close to a perfect diagonal within a factor of $893/1024 \approx 0.87$, which is close to what is computed by theory. Depending on the similarity threshold that is defined, this result can be accepted to indicate a match or not.

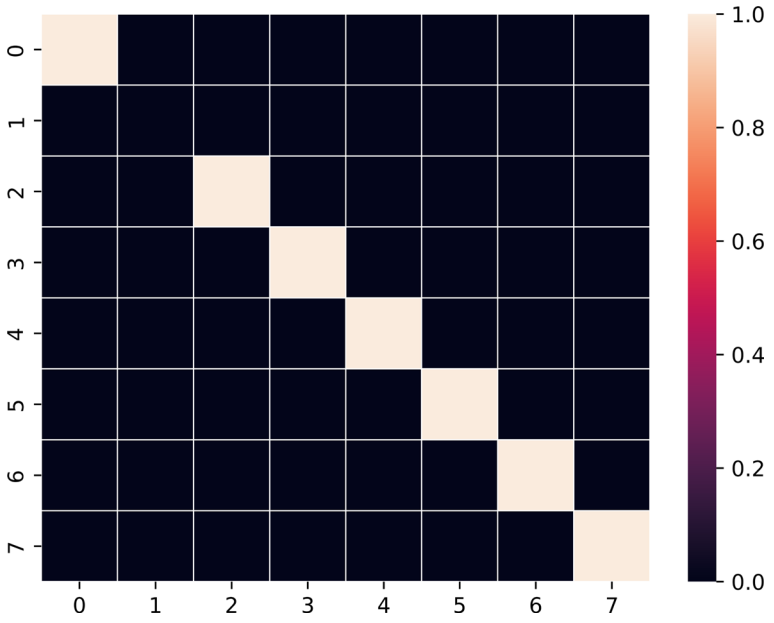

Fig. 6 Heatmap of a dot plot containing a diagonal with a single gap

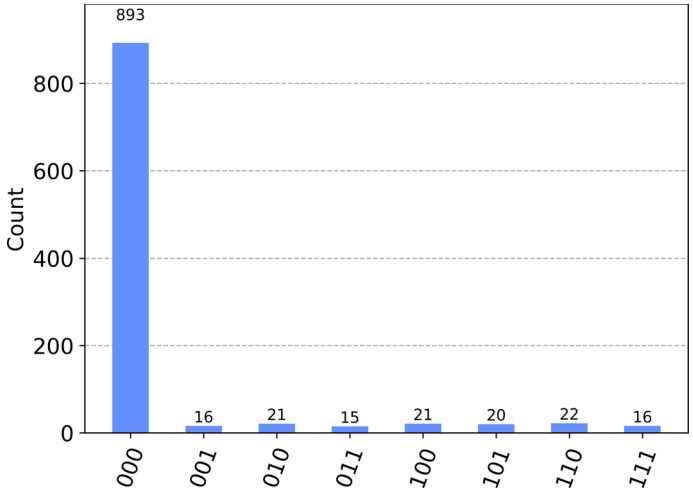

Fig. 7 Simulation results for a dot plot with a single gap in the diagonal

5.3 Off-diagonal noise

The last case concerns off-diagonal matches. Figure 8 depicts a sample dot plot, where there are two off-diagonal matches indexed at positions $(0, 4)$ and $(4, 0)$. When dot plots are generated by comparing text data, off-diagonal matches generally come in

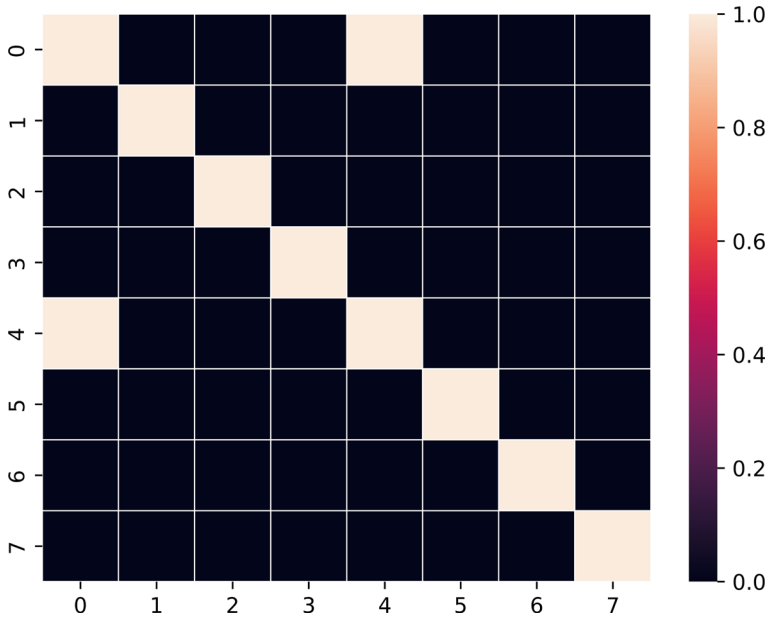

Fig. 8 Heatmap of a dot plot containing noise with 2 off-diagonal matches

pairs; if (i, j) position denotes a match, so does position (j, i). If, however, the diagonal denotes matches via another procedure other than dot plots, this is not necessarily true. Figure 9 denotes the results of running the circuit for a total of 1024 shots and obtaining the counts. The similarity is $859/1024 \approx 0.83$, which is within the bounds defined in Eq. 23.

6 Performance

In this section, an analysis of the complexity of the algorithm will be provided. The analysis will focus on two axes, namely a standalone analysis that will be a running time estimation for the algorithm (Sect. 6.1) and a comparative analysis with the most known classical sequence alignment strategies (Sect. 6.2) as well with the most established approaches for quantum pattern matching (Sect. 6.3).

6.1 Running time complexity

The complexity of the algorithm depends on the accuracy needed for the phase estimation and the execution time of the reverse QFT.

For the QPE part and as a first approximation, the $controlled - U$ operation can be considered as given oracle operations and be assigned a cost of $O(1)$ (we will see in Sect. 7 how it can be decomposed in basis gates). Under this assumption, the QPE algorithm can be shown to require a total of $t = n + p$ qubits, where n is the

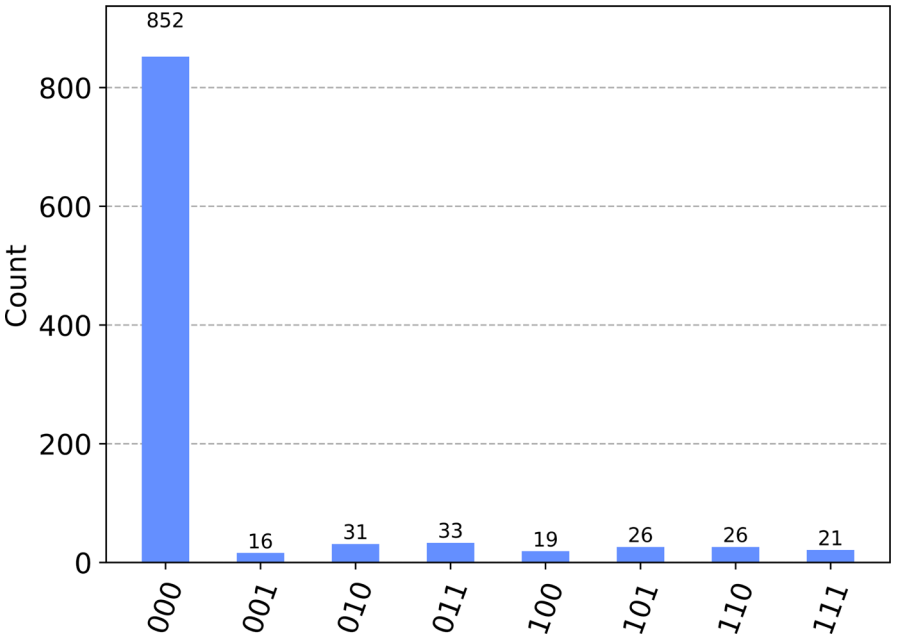

Fig. 9 Simulation results for a noisy dot plot with 2 off-diagonal matches

total number of qubits required to encode the state and p is the number of qubits used to encode the eigenphase. The number p is related to the additive error of the phase estimation (to achieve a smaller error, a greater number of $controlled - U$ operations is required), which scales as $p \approx O(\log \frac{1}{\epsilon})$. To achieve this additive error, the number of $controlled - U$ operations required is $O(\frac{1}{\epsilon})$ [20].

When compared to classical algorithms, and ignoring the cost of constructing the dot plot, these typically have to check a number of elements that is proportional to the size N of the diagonal, and they run in $O(N)$ steps. For perfect diagonals, where a single run is required, the quantum algorithm presented clearly outperforms the classical ones. As the tolerance threshold is increased, an even greater number of running shots is required to obtain the sampler output that reproduces the output distribution faithfully. According to Theorem 1, if the acceptance threshold, defined by the acceptable number of mismatches m (gaps plus off-diagonal matches), scales lower than $O(N)$, then we have an asymptotical speedup. If, however, m scales similarly to N, then there is no speedup. A 50% similarity acceptance criterion combined with a small window size in a DNA matching problem, for example, may lead to dot plots that need a number of shots that scale as $O(N)$ before constructing a reliable output distribution.

6.2 Relevance to sequence alignment algorithms

When viewed in relation to existing algorithms for pairwise sequence alignment, our algorithm should be seen as an accelerating routine for other algorithms that utilize

dot plots to identify sequence matches. If, for simplicity, we assume a sequence of equal matches equal to N, a dot plot algorithm would require:

- $O(N^2)$ steps for generating the dot plot
- $O(N^2)$ steps for analyzing the dot plot
- $O(N^2)$ space for storing the dot plot

These steps may, of course, involve further analysis depending on the specifics of the problem at hand, such as whether the search includes gaps and mismatches. Another class of algorithms is the ones based on dynamic programming, such as the SW [12] or the NW [11] algorithm. Dynamic algorithms have been analyzed in [21], and they also generally require $O(N^2)$ time to run, while the space requirements can be linear for certain strategies. Finally, algorithms based on heuristics, such as FASTA [22] or BLAST [23], are widely used in practice. These algorithms, although they still have a complexity of $O(N^2)$, in practice they typically perform way better than algorithms that depend on exhaustive search, although they do so with a possible loss of accuracy in their results due to their heuristic nature.

As mentioned in Sect. 6, our algorithm can potentially lower the number of steps required for analyzing the dot plot, especially if the number m of gaps and mismatches is significantly lower than N. Regarding the dot plot generation, although not treated in this work, this can be achieved by various techniques. Clapis [19], for example, uses Quantum Programmable Logic Array (QPLA) techniques to dynamically construct the dot plot; by leveraging the Espresso algorithm [24], Clapis construction, while still polynomial, is still significantly shorter than the one obtained using QPLA directly. One drawback, however, is the need for multi-controlled gates, which typically behave very badly under transpilation into one- and two-qubit gates. Another approach, currently followed by many other problems, is to reduce the steps and number of qubits needed to encode the dot plot by using custom encoding schemas that go beyond the typical basis state encoding. Such a schema could be the quantum autoencoder [25], which utilizes quantum machine learning techniques to compress the input data so that the dot plot is then generated in fewer steps. Of course, under this new encoding the exact formulation of our operator U would have to change, but, since it is defined based on properties of its eigenspace, we expect that the change would involve its representation only. This approach is a work in progress currently by the authors. Generally, as long as the dot plot generation remains quadratic, it will dominate the time complexity of any dot plot approach. As such, the quantum acceleration offered by our algorithm is not expected to have such a big impact, at least in the near future, in challenging established classical algorithms such as FASTA and BLAST. For a subclass of problems, however, that do not involve full alignment but only identification of regions of high similarity, the algorithm may have a practical impact.

6.3 Relevance to other quantum algorithms for pattern matching

Quantum pattern matching algorithms can be broadly separated into two categories, the ones that are based on the query model and that typically leverage the approach of Grover's algorithm, and the ones that depend on machine learning techniques.

From the first category the original algorithm of Grover allows searching in an unstructured database with a time complexity equal to $O(\sqrt{N})$; supplemented with new techniques that allow the efficient construction of equal superposition even when the size of the state is not an exact power of 2 (see, for example, [26]), the application of Grover's algorithm can have a wide range of text searching problems. More related to the problem of searching sequences, we can identify the algorithm proposed by Niroula and Nam [27] which searches for a pattern of length M within a text of size N in $O(\sqrt{N})$ steps, using $O(N + M)$ space. Schaller and Schutzhold on the other hand proposed an algorithm for image template recognition that similar to other approaches achieves a $O(\sqrt{N})$; however, they also proposed a noise filtering methodology that can identify similar images that differ by some noise, in a similar fashion that our algorithm tries to identify and isolate partial matches.

Compared to the above algorithms, our approach achieves a linear speed-up in the best-case scenario. However, for a certain domain of use cases, it can be argued that our algorithm has certain advantages:

- While generic pattern matching does not assume a structure of the input data, our algorithm is performed on data that is pre-processed, such as dot plots. The mechanics of preprocessing are encoded in the U operator, and the characteristics of the dataset are taken into account by the search algorithm, such structure is, by definition, ignored by algorithms that tackle the unstructured search problem.
- Our algorithm can be versatile to accommodate various preprocessing schemes. Although demonstrated for the dot plot use case, different representations and/or function definitions may accommodate different use cases; for example different operands in the addition module N could lead to detection of rows (addition of one), columns (addition of N) or even checkboard patterns (addition in mod2 arithmetic).

Quantum machine learning (QML) techniques, on the other hand, have shown great promise in providing efficient alternative algorithms for various problems. Most of the popular quantum machine learning approaches, such as the quantum neural networks (QNNs), make use of feature maps, circuits that are used to efficiently encode, map and reduce the dimensions needed to encode information. In contrast with the query model, various feature maps that are of practical use can be implemented very efficiently in NEAR-term hardware. One remarkable result by Huang et al. [28] demonstrates a methodology for constructing quantum kernels that cannot be modeled classically in an efficient manner. The extent to which these kernels are meaningful (i.e., they can be used to extract useful information from datasets and thus be used by a machine learning algorithm) is still a matter of research. If such kernels are found, computational tasks that are performed by classical ML techniques are expected to be significantly accelerated in the quantum setting. As classical neural networks show remarkable efficiency in finding patterns, the corresponding QNNs with asymptotically faster kernels can realize even greater breakthroughs.

Contrasting our algorithm with QML, it has little overlap, since its mechanics and methodology follow a different path. However, QML can be used for the construction of smaller and more NEAR-term friendly circuits of implementing the U operator as mentioned in Sect. 6.2.

7 Implementation in NEAR-term hardware and error analysis

While error advancements in hardware, as well as algorithm design and error mitigation techniques [29], have led to an increase of the size of problems that can be run on real quantum processors, the size of problems that can be tackled in real quantum hardware is still much lower compared to classical implementations. As the number of qubits grows with each year and as the threshold theorem [30] establishes the possibility of arbitral scaling once certain error thresholds have been achieved, there is a good possibility to witness practical quantum computations in the near future. Currently, most of the algorithms that run for sufficient depth to justify anticipation for practical applications in the near future are those that are specifically targeted to run on NEAR-term hardware, such as ML algorithms that are based on feature maps [31] that can be easily implemented in NEAR-term hardware or Variational Quantum Eigensolvers based on hardware efficient ansatzes [32] .

Unfortunately, for our algorithm, as well as for many of the well-known quantum algorithms, such as Shor's and Grover's algorithms, transpilation in existing hardware is not straightforward. The main reason is that both of these algorithms use the query model of computations for operators that entangle a large number of qubits. Under the query model, function f is implemented via an operator U defined by the transformation:

$$U_f(|x\rangle|y\rangle) \rightarrow |x\rangle|y \oplus f(x)\rangle \tag{25}$$

U_f performs a mapping between basis states that is 1-1 in order for it to be unitary. As such, it essentially performs a permutation of the input qubit states. Such permutation matrices can require a large number of single- and two-qubit gates, which is further increased when the limitation of qubit connectivity is taken into account. This section will provide a gauge of our algorithm's requirements in terms of single- and two-qubit gates, as well as an estimation of computational error and methods by which this can be reduced.

7.1 Gate count analysis

As mentioned, the main obstacle in running the algorithm in NEAR-term hardware is the transpilation of the U operator defined in Eq. 14. As proven in [33], a general n-qubit U gate can be decomposed in a number of basic operations that has an upper bound of $\Theta(n^3 4^n)$. Currently, there is no known efficient process for transpiling permutation matrices in one- and two-qubit gates that can produce circuits that have such a depth that practical applications can be run. However, some known methods combined with the characteristics of the algorithm can be exploited to lower the required number of qubits and the depth requirements of the circuit. Before we describe such approaches, we first present some transpilation results using the maximum optimization level of Qiskit's transpiler (see "Appendix A" for a listing of the source code that produced these results). The results can be seen in two cases:

Table 1 Number of gates produced by Qiskit transpiler when the circuit is transpiled against the Mumbai backend

Problem size	RZ	SX	X	CX	Depth
2	425	257	24	241	625
4	8793	5774	–	6787	14,390
6	36,633	24,206	–	28,927	60,790
8	146,727	96,158	–	121,609	243,040

Table 2 Number of gates produced by Qiskit transpiler when the circuit is transpiled against a fully connected backend

Problem size	RZ	SX	X	CX	Depth
2	285	182		108	{383
4	7302	5024		2577	{9285
6	30,951	21,554		11,211	{42,200
8	125,124	85,528		45,261	{165,291

- For the case of qubit topology that resembles one found in typical transmon-based processors in Table 1. For this case, the coupling map of the IBM's 27-qubit Mumbai processor was used. As the connectivity is limited, the transpiler has to account for the extra gates needed to conform to connectivity constraints that are not present in the original circuit design
- For the case of a fully connected qubit topology as depicted in Table 2. Although such topologies may be found in certain architectures (e.g., processors based on Ion-traps), they are unrealistic for most. They serve here to gauge the cost solely due to gate decomposition.

As can be seen by the results, the number of gates scales significantly with the problem size. It is to be noted that the number of gates does not exhibit the typical explosion encountered in generic quantum circuits that are based on Programmable Logic Array (such as those used for encoding in [19]). This is probably due to the fact that the operator has a more "local" behavior, in the sense that, apart from the control qubit, its input gates remain the same throughout the computation. This is also reflected in the fact that the depth of the tranpiled circuits is comparable in the two cases (less than double for a given size).

7.2 Error analysis

In order to estimate the accumulation of errors, we are going to provide some models of how the error accumulates under two models. We will ignore State Preparation And Measurements (SPAM) errors in this analysis, as the way these are introduced does not depend on any specific characteristics of our algorithm. Furthermore, we will ignore the analysis of the non-unitary operator that encodes the dot plot, as this again can be implemented in various ways and will be treated here as a black box; Schutzohld, for example, gives an implementation based on optical hardware in [2], while Clapis provides a circuit for generating the dot plot dynamically together with the analysis of the size of the transpiled circuits in [19].

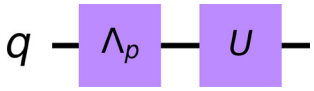

Fig. 10 Incoherent noise model. The gate Λ is introduced to the channel with probability p and the identity gate I with probability $1 - p$

Our analysis will be based on the case of transpilation of a fully connected qubit network (Table 2). We will create a circuit that contains a similar distribution of gates, that is, a number of CX gates equal to 25% of the depth and a number of SX gates equal to 70% of the depth. X gates will be ignored, as they were produced in very small numbers. We will also ignore the RZ gates; these are virtual in the Transmon-based backends of IBM's Quantum Platform and can be implemented by the classical hardware by appending phase shifts in subsequent pulses (see, for example, the explanation in the relevant part of the documentation of the Pulser software package [34]).

7.2.1 Depolarizating noise model

Depolarization is a case of incoherent noise, namely noise that is due to dissipative processes that cause a loss of information and the reduction of the volume space that a qubit occupies in the Bloch sphere. In an incoherent error model, for each gate in the circuit, a gate Λ is introduced in each operation with a certain probability that causes unwanted alteration of the qubit state (Fig. 10). Typical error models of incoherent noise include:

- Bit flip errors, in which $\Lambda = X$ with probability p and which causes shrinking of the qubit's Bloch sphere along the Y- and Z-axes.
- Phase flip errors in which $\Lambda = Z$ with probability p and which causes shrinking of the qubit's Bloch Sphere along the X- and Y-axes.
- Depolarizing errors in which $\Lambda = X, Y, Z$ each with probability $p/4$. These errors cause the shrinking of the qubit's Bloch sphere equally along all axes.
- Pauli errors in which $\Lambda = X, Y, Z$ each with probability p_x, p_y and p_z, respectively.

From the above models, we will adopt the depolarizing error model as this includes the most generic model, including all axes of the Bloch spheres, without having any bias towards any specific axis as the Pauli model does. For the two-qubit gate, the depolarizing error is computed by taking the tensor of two depolarizing errors.

We create a circuit of depth equal to 1000, and we populate it with SX and CX gates with the population numbers mentioned above. To accommodate for the missing non-unitary state preparation circuit, we impose the condition that on the most significant qubit a number of SX gates will be applied that will be a multiple of 4. This way, the noise-free circuit will produce a subspace of the possible output states; the deviations from this subspace can then be used to gauge the accumulation of error under different values of error.

Figure 11 depicts the results for various values of p when the circuit is run on a

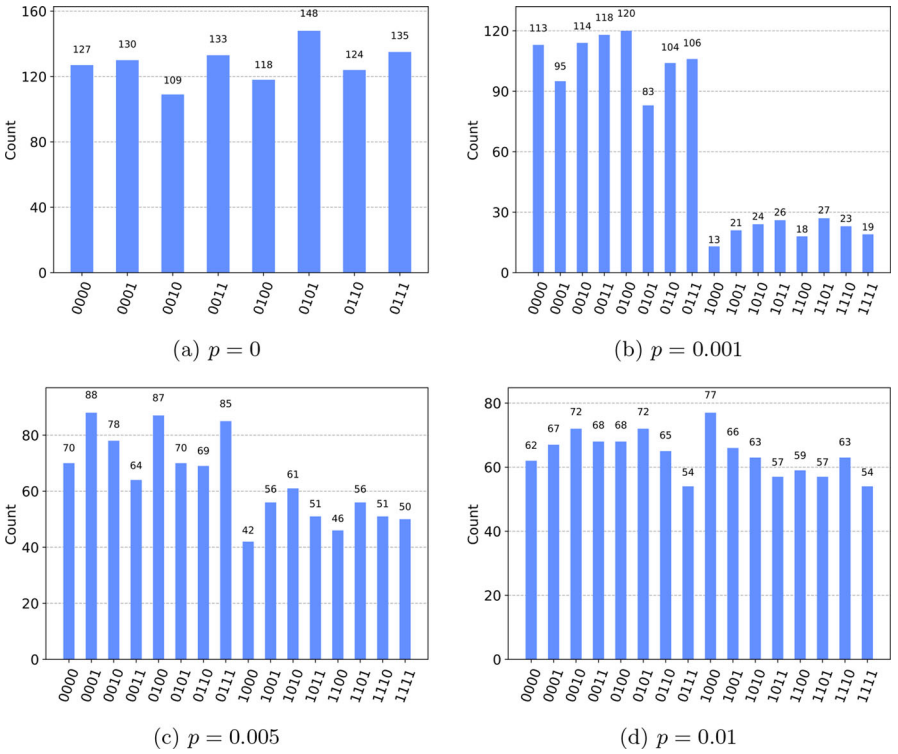

Fig. 11 Results under different values of the depolarizing error p for a fixed circuit depth equal to 1000

circuit of depth equal to one thousand. It can be seen that, for a value of $p = 0.005$, there is already little initial information left in the final distribution, while that for $p - 0.01$ is essentially random. This can be seen more clearly in Fig. 12, in which the entropy of the output is listed versus the values of p is listed. For $p = 0$ the entropy is equal to $H_{p=0} \approx 3$, while for $p \geq 0.005$, the entropy becomes $H_{p=0.005} \approx 4$; as the output register has a size of 4 bits, this value approximates the maximum possible value. To perform practical runs that involve transpiled circuits of the depth of the order of hundreds of thousands or more, the depolarizing error should be smaller than 0.1%, or some other techniques to reduce the problem size are used.

7.2.2 Decoherence model

For this analysis, instead of adopting a stochastic alteration of a qubit, we explicitly estimate the error based on the relaxation times of the qubits. As is typical for this type of analysis, we consider:

- The relaxation time T_1, which is the time it takes for the excited state $|1\rangle$ to decay to state $|0\rangle$
- The dephasing time T_2, which is the time it takes for the qubit to lose its phase information, i.e., the time it takes for state $|+\rangle$ or $|-\rangle$ to become a mixture of

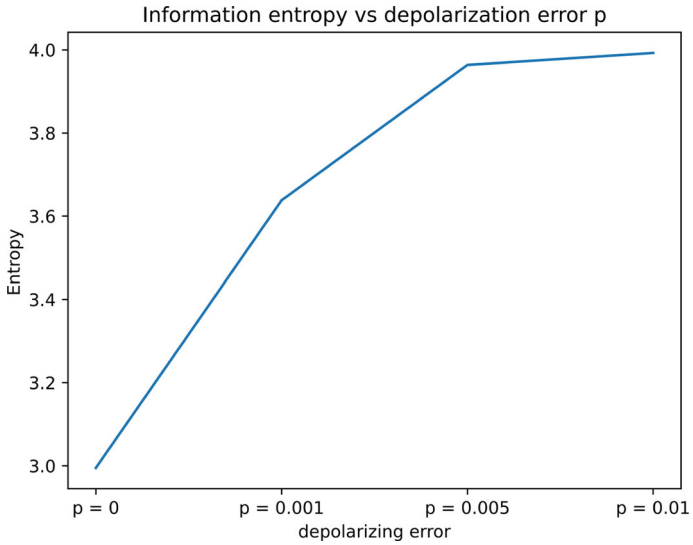

Fig. 12 Information content of the circuit's output for various values of p

phases, from which no information about the relative phase of the initial state cannot be inferred by measurement.

Figure 13 depicts the results of our simulation. It can be seen that for $T1 = 222us$ and $T2 = 138us$, which are times typically encountered in IBM's quantum processors, there is a lot of the initial information still present; the retained information content is even higher for longer decoherence times. The circuit behaves better under the decoherence model than under the depolarizing error model; however, it should still be noted that based on these results, and in the same manner as with the depolarizing model, practical runs that involve transpiled circuits of the depth of the order of hundreds of thousands lie still far ahead.

7.3 Mitigation measures

The noisy quantum computers of the NISQ era depend on error correction [35] and error mitigation [36] techniques to increase the size of the problem instances run. While the majority of these techniques are general for any computational task, specific algorithms can leverage custom techniques to target characteristics of the underlying structure of problems and achieve further lowering of the requirements in terms of qubits and gates and of the accumulated computational error.

One such technique is the Iterative Phase Estimation [37] algorithm that targets the QPE part of our algorithm. IPE can lower the number of the required accuracy qubits by using only one qubit. For each bit of accuracy needed, the controlled operator is performed the required number of times then the qubit is measured. After it is measured, it is reset to $|0\rangle$ and a phase correction gate is applied; the process is repeated until all required values are collected. IPE lowers the number of target qubits; since the

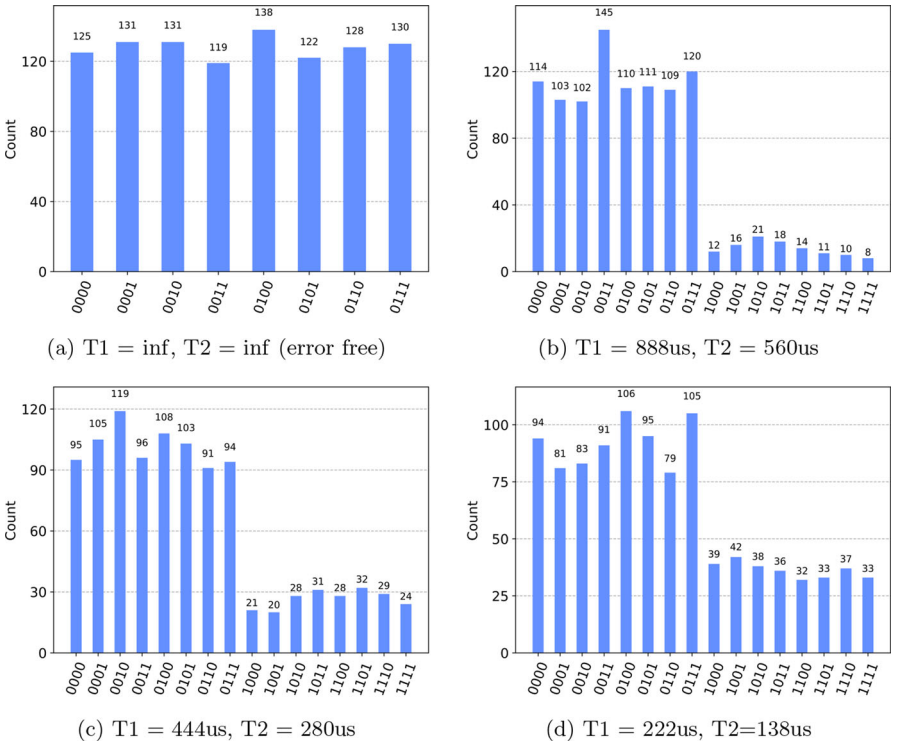

Fig. 13 Results under different times for decoherence error for a fixed circuit depth equal to 1000

operator now has a constant topology (it is coupled to only one control qubit), transpilation can be more efficient. The introduction of mid-circuit measurement, however, means that that circuit has to remain idle for a time defined by the measurement time of the device. This can be greater than the decoherence time, and therefore, additional errors may be introduced.

On another line of attack, one key attribute of the problem of diagonal matching is that matrices can be separated into segments, which can be solved separately and the result combined. This does not hold for other algorithms in general; for example, factorization of a large number cannot be decomposed into smaller problems involving smaller numbers. A large matrix, on the other hand, can be split into diagonal submatrices that can be solved independently; the result can then be post-processed classically via, for example, majority voting to determine whether a pattern was identified or not. This segmentation has two major impacts. The first impact is that the problem can be solved for matrices whose size is significantly larger than the number of qubits of the underlying hardware, thus overcoming one of the major limitations of existing hardware. The second one is that error accumulation can be minimized by having the segments run in different runs, either sequentially or in parallel. The smaller instances will have transpiled circuits of smaller depth than the full one, and therefore, they will accumulate significantly less error. As long as the number of segments is

asymptotically lower than the problem instance size, the algorithm should still retain its quantum advantage.

8 Results and discussion

The present document presented a methodology for identifying regions of potential matches when comparing large datasets. The algorithm makes use of an operator that is defined to have eigenstates that correspond to matched datasets and can identify areas of similarity with a running time that is asymptotically better when the matching threshold is high enough.

In a practical scenario that is expected to run in NEAR-term hardware (i.e., noisy quantum processors that are currently available or are expected to be available in the near future), the input data will be fragmented into chunks that can be represented using the limited number of qubits available in the underlying architecture of the quantum processor. Areas of similarity can be identified, or the outputs can be pooled to conclude the similarity level of the whole dataset.

Although the algorithms were exhibited for a specific case, that of encoding matches between target and reference data using a dot plot, it can be expanded to include other schemes. This would include a re-definition of the operator U in such a way that its eigenspace includes a state that encodes a perfect match. Due to the linearity of the operator, it is expected that for any such operator, an analysis similar to the one conducted in Sect. 4 will hold; i.e., small deviations from a perfect match will lead to states that have a significant overlap with the "perfect" eigenstate, and thus the QPE algorithm will produce an output with high peaks for the phase corresponding to the eigenphase of U.

Author Contributions D.N., A.K., and N.K. captured the main idea and designed the algorithm with equal contribution. D.N. led the implementation of the algorithm in Qiskit and the running of the simulations with the contribution of A.K. and N.K. A.K. performed the error analysis for partial matches with the help of D.N. (experimental verification) and N.K. (validation). N.K. led the final review of the document.

Appendix A: Source code

Procedure for constructing the U operator that cycles back eigenstates corresponding to proper diagonals.

Listing 1 Code for constructing the operator

```
import numpy as np
import math
from qiskit.quantum_info import Operator
from qiskit.extensions import UnitaryGate

 def generate_x_mod_y_AdditionOperators(x, y):
     N = 2 ** int(math.ceil(math.log(y, 2)))
     u = np.zeros(shape = (N,N))
     x_mod_y = x
```

```
    permutation = 0
    for i in range(x_mod_y, (y + x_mod_y)):
        u[i
        permutation += 1

    for r in range(y,N):
        u[r][r] = 1
    U = UnitaryGate(u, label="U")
    return U
```

Non-unitary process for preparing a state that encodes a square matrix consisting of entries equal to 0 or 1.

Listing 2 Code for preparing an input state

```
import numpy as np
from qiskit import QuantumCircuit
from qiskit.circuit.library import StatePreparation

def amp_enc_dagger(dotplot2DArray, NoQubits):
    arr = dotplot2DArray.flatten()
    norm = np.linalg.norm(arr)
    SP = StatePreparation(arr / norm)
    qc_enc = QuantumCircuit(NoQubits)
    list_of_index_numbers = list(range(0, NoQubits))
    qc_enc.name = "Encoder"
    qc_enc.append(SP, list_of_index_numbers)
    return qc_enc
```

Procedure for constructing a list of circuits that implement the algorithm. The procedure assumes that a separate process has created the proper encoding of the input states.

Listing 3 Code for constructing a list of circuits that implement the algorithm

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit.circuit.library import QFT

qc_all = []
n_acc=3
for size in range(3,10):
  CU_t = generate_x_mod_y_AdditionOperators(size, size*(size-1))
  phase_reg = QuantumRegister(n_acc, 'phase')
  working_reg = QuantumRegister(CU_t.num_qubits-1, 'plot_U')
  working_meas = ClassicalRegister(CU_t.num_qubits-1)
  phase_meas = ClassicalRegister(n_acc)

  qc_phase = QuantumCircuit(phase_reg, working_reg, phase_meas)
  qc_phase.h(phase_reg)
  qc_phase.barrier()
  for i in range(n_acc):
    for j in range(i+1):
      qc_phase.append(CU_t, \
        [phase_reg[i]] + \
        [work_q for work_q in working_reg[:CU_t.num_qubits - 1]])
  qc_phase.barrier()
```

```
  qc_phase.append(QFT(num_qubits = 3, \
    inverse=True, do_swaps=True, \
    name="$QFT^{\dagger}$"), \
    [phase_q for phase_q in phase_reg])
  qc_phase.barrier()
  qc_phase.measure([phase_q for phase_q in phase_reg],\
    [phase_c for phase_c in phase_meas])
  qc_all.append(qc_phase)
```

Code for transpiling the circuit on FakeMumbaiV2 backend of IBM's quantum platform and a simulated fully connected backend.

Listing 4 Code for transpiling the circuits

```
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
from qiskit.providers.fake_provider import FakeMumbaiV2
from qiskit import transpile

backend = FakeMumbaiV2()
pass_manager = generate_preset_pass_manager(3, backend)

qc_transpiled_real = []
for qc in qc_all:
  qc_t = transpile(qc, backend)
  qc_transpiled_real.append(qc_t)

from qiskit.transpiler import CouplingMap

cmap = CouplingMap.from_full(9)

qc_transpiled_fully = []
for qc in qc_all:
  qc_t = transpile(qc, optimization_level=3,\
    basis_gates=['id', 'rz', 'sx', 'x', 'cx'], \
    coupling_map = cmap)
  qc_transpiled_fully.append(qc_t)

gate_list_tfull = []
for qc in qc_transpiled_fully:
  gate_list_tfull.append(qc.count_ops())
df_f = pd.DataFrame(gate_list_tfull, columns=gate_list_tfull[0].keys())
```

Funding Open access funding provided by HEAL-Link Greece.

References

1. Wilkinson, L.: Dot plots. Am. Stat. **53**(3), 276–281 (1999)
2. Schützhold, R.: Pattern recognition on a quantum computer. Phys. Rev. A **67**(6), 062311 (2003)
3. Prousalis, K., Konofaos, N.: A quantum pattern recognition method for improving pairwise sequence alignment. Sci. Rep. **9**(1), 7226 (2019)
4. Jun, K., Lee, H.: Hubo formulations for solving the eigenvalue problem. Results Control Optim. **11**, 100222 (2023)
5. Cakoni, F., Colton, D., Houssem, H.: Transmission eigenvalues. Not. Am. Math. Soc. **68**(9), 1499–1510 (2021)
6. Kalogeropoulos, A., Tsitsas, N.L.: Excitation of a layered sphere by n acoustic sources: exact solutions, low-frequency approximations, and inverse problems. Q. Appl. Math. **81**(1), 141–173 (2023)
7. Elsner, L., He, C.: Perturbation and interlace theorems for the unitary eigenvalue problem. Linear Algebra Appl. **188**, 207–229 (1993)
8. Bohnhorst, B., Bunste-Gerstner, A., Fassbender, H.: On the perturbation theory for unitary eigenvalue problems. SIAM J. Matrix Anal. Appl. **21**(3), 809–824 (2000)
9. Kitaev, A.Y.: Quantum Measurements and the Abelian Stabilizer Problem (1995)
10. Edgar, R.C., Batzoglou, S.: Multiple sequence alignment. Curr. Opin. Struct. Biol. **16**(3), 368–373 (2006)
11. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. **48**(3), 443–453 (1970)
12. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. **147**(1), 195–197 (1981)
13. Lassmann, T.: Kalign 3: multiple sequence alignment of large datasets. Bioinformatics **36**(6), 1928–1929 (2020)
14. Tang, F., et al.: Halign 3: fast multiple alignment of ultra-large numbers of similar DNA/RNA sequences. Mol. Biol. Evol. **39**(8), 166 (2022)
15. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 212–219 (1996)
16. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134. IEEE (1994)
17. Weigold, M., Barzen, J., Leymann, F., Salm, M.: Data encoding patterns for quantum computing. In: Proceedings of the 27th Conference on Pattern Languages of Programs, pp. 1–11 (2020)
18. Qiskit contributors: Qiskit: An Open-source Framework for Quantum Computing (2023). https://doi.org/10.5281/zenodo.2573505
19. Clapis, J.: A quantum dot plot generation algorithm for pairwise sequence alignment. arXiv:2107.11346 (2021)
20. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information, 10th Anniversary edn., pp. 223–226. Cambridge University Press, Cambridge, England (2010). Chap. 5.2.1
21. Pearson, W.R., Miller, W.: Dynamic programming algorithms for biological sequence comparison. Methods Enzymol. **210**, 575–601 (1992)
22. Lipman, D.J., Pearson, W.R.: Rapid and sensitive protein similarity searches. Science **227**(4693), 1435–1441 (1985)
23. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. J. Mol. Biol. **215**(3), 403–410 (1990)
24. Holdsworth, B., Woods, C.: Digital Logic Design, 4th edn. Elsevier, Oxford (2002)
25. Romero, J., Olson, J.P., Aspuru-Guzik, A.: Quantum autoencoders for efficient compression of quantum data. Quantum Sci. Technol. **2**(4), 045001 (2017)
26. Shukla, A., Vedula, P.: An efficient quantum algorithm for preparation of uniform quantum superposition states. Quantum Inf. Process. **23**(2), 38 (2024)
27. Niroula, P., Nam, Y.: A quantum algorithm for string matching. NPJ Quantum Inf. **7**(1), 37 (2021)
28. Huang, H.-Y., Broughton, M., Mohseni, M., Babbush, R., Boixo, S., Neven, H., McClean, J.R.: Power of data in quantum machine learning. Nat. Commun. **12**(1), 2631 (2021)
29. Huang, H.-L., Xu, X.-Y., Guo, C., Tian, G., Wei, S.-J., Sun, X., Bao, W.-S., Long, G.-L.: Near-term quantum computing techniques: variational quantum algorithms, error mitigation, circuit compilation, benchmarking and classical simulation. Sci. China Phys. Mech. Astron. **66**(5), 250302 (2023)

30. Aharonov, D., Ben-Or, M.: Fault-tolerant quantum computation with constant error. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pp. 176–188 (1997)
31. Dou, T., Zhang, G., Cui, W.: Efficient quantum feature extraction for CNN-based learning. J. Frankl. Inst. **360**(11), 7438–7456 (2023). https://doi.org/10.1016/j.jfranklin.2023.06.003
32. Grimsley, H.R., Economou, S.E., Barnes, E., Mayhall, N.J.: An adaptive variational algorithm for exact molecular simulations on a quantum computer. Nat. Commun. **10**(1), 3007 (2019)
33. Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A., Weinfurter, H.: Elementary gates for quantum computation. Phys. Rev. A **52**(5), 3457 (1995)
34. LLC, M.: Phase Shifts and Virtual Z Gates - Pulser 0.17.0 Documentation. https://pulser.readthedocs.io/en/stable/tutorials/phase_shifts_vz_gates.html
35. Chiaverini, J., Leibfried, D., Schaetz, T., Barrett, M.D., Blakestad, R., Britton, J., Itano, W.M., Jost, J.D., Knill, E., Langer, C., et al.: Realization of quantum error correction. Nature **432**(7017), 602–605 (2004)
36. Endo, S., Benjamin, S.C., Li, Y.: Practical quantum error mitigation for near-future applications. Phys. Rev. X **8**(3), 031027 (2018)
37. Dobšíček, M., Johansson, G., Shumeiko, V., Wendin, G.: Arbitrary accuracy iterative quantum phase estimation algorithm using a single ancillary qubit: a two-qubit benchmark. Phys. Rev. A **76**(3), 030306 (2007)