Directed Graph Encoding in Quantum Computing supporting Edge-Failures*

D. Della Giustina[1], C. Piazza[1], B. Riccardi[1], and R. Romanello[1]

Dept. of Mathematics, Computer Science and Physics, University of Udine, Italy
{dellagiustina.davide,riccardi.brian,romanello.riccardo}@spes.uniud.it,
carla.piazza@uniud.it

Abstract. Graphs are one of the most common data structures in classical computer science and graph theory has been widely used in complexity and computability. Recently, the use of graphs in application domains such as routing, network analysis and resource allocation has become crucial. In these areas, graphs are often evolving in time: for example, connection links may fail due to temporary technical issues, meaning that edges of the graph cannot be traversed for some time interval and alternative paths have to be followed. In classical computation, where graphs are represented as adjacency matrices or lists, these problems are well studied and ad-hoc visit procedures have been developed. For specific problems quantum computation, through superpositions and entanglement has provided faster algorithms than their classical counterpart. However, in this model, only reversible operations are allowed and this poses the quest of augmenting a graph in order to be able to reverse edge traversals. In this paper we present a novel graph representation in quantum computation supporting dynamic connectivity typical of real-world network applications. Our proposal has the advantage of being closer than others in literature to the adjacency matrix of the graph. This makes easy dynamic edge-failure modeling. We introduce optimal algorithms for computing our graph encoding and we show the effectiveness of our proposal with some examples.

Keywords: Quantum Walks, Graphs, Edge-Failure

Introduction

Graphs are ubiquitous in computer science and mathematics. They are formal and flexible frameworks used to model a wide class of problems from different fields, ranging from Complexity Theory [24], Flow Theory [2] and Software Verification [19], to more application-oriented domains such as Routing [29], Machine Learning [16] and Social Networks [11]. Randomization plays a central role in collecting informations from ever growing and dynamically evolving networks,

* This work is partially supported by PRIN MUR project Noninterference and Reversibility Analysis in Private Blockchains (NiRvAna) - 20202FCJM and by GNCS INdAM project LESLIE.

and with the advent of quantum computing as a more and more useful tool, it is mandatory to define efficient encodings for graphs in quantum circuits.

Quantum computation is constitutionally reversible being based upon unitary transformations that are always reversible. Only measurements/observables collapse quantum states into classical ones with a loss of information that is not reversible. As a matter of fact, classical circuit (i.e., classical boolean functions) can be simulated by quantum ones, but it is necessary to pay the encoding in terms of space dimension. In the case of boolean functions the standard technique for doing this consists in representing a function $f : \{0,1\}^n \to \{0,1\}^m$ as the reversible function $f' : \{0,1\}^{n+m} \to \{0,1\}^{n+m}$ with $f'(x,y) = (x, y \oplus f(x))$.

So, on the one hand, quantum computation allows to exactly solve in polynomial time problems that are in the time complexity class EXP for classical computation, thus proving that $QP \neq P$ (see, e.g., [10, 27]). Moreover, it allows to solve with bounded probability of error in polynomial time the factorization problem for which at present there are no classical polynomial algorithms with bounded probability of error. The impact of superposition and entanglement in such speed-up has been evaluated in the literature from different perspectives (see, e.g., [5, 6, 17]).

On the other hand, the possibility of using only reversible operators causes an increase of space requirements as shown in the simple case of boolean functions and imposes to redefine each step of classical algorithms in terms of linear unitary (reversible) transformations.

The theory of quantum walks, the quantum counterpart of random walks, has been first introduced in [1]. As for the classical case [8, 22], there is a distinction between continuous and discrete time models [30]. The former is introduced in [13] and further investigated in [7, 20]: in this case, the main tool is the exponentiation of a suitable hermitian matrix derived from the adjacency matrix of the graph. The latter is described in [1] as a *coined* walk, where a suitable matrix – the coin – is introduced to implement the random choice while maintaining unitarity of the encoding matrix. For further details and applications description we refer the reader to [26, 31].

The aim of our work is to encode graphs in the quantum formalism, so that algorithms, such as random walks, can be efficiently developed also in dynamic settings where edges / nodes can be temporary unavailable. We extend the work done in [28] by providing a procedure to optimally compute a unitary matrix from the line of an eulerian graph. In this way, the obtained graph representation has its focus on the edges rather than on the nodes, as it was in [1, 20]. In the general case of non-eulerian graphs, we describe an embedding which can be implemented in the quantum framework through projectors. The projectors allow us to *hide* the edges added from the embedding. Moreover, they can be fruitfully exploited for managing edge failures, without recomputing the encoding matrices.

The paper is structured as follows. In Section 1 we briefly recall some basic definitions from Quantum Computation, Graph Theory and Quantum Random Walks. An optimal procedure to obtain a unitary matrix from any eulerian multigraph is presented in Section 2. The following section has the goal to show how to embed any multigraph into an eulerian one. In Section 4 we describe the quantum circuit obtained using the results from the previous sections. The efficient management of edge failures in explained in Section 5. Finally, Section 6 illustrates some comparisons with related literature with examples.

1 Preliminaries

1.1 Quantum Computing

The most used model of Quantum Computation relies on the formalism of state vectors, unitary operators and projectors. At high level we can say that state vectors evolve during the computation through unitary operators, then projectors are used to remove part of the uncertainty on the internal state of the system.

The state of the system is represented by a unitary vector over $\mathbb{C}^n$ with $n = 2^m$ for some $m \in \mathbb{N}$. The concept of a bit of classical computation is replaced by that of a qubit. While a bit can have value 0 or 1 a qubit is a unitary vector of $\mathbb{C}^2$. When the two components of the qubit are the complex numbers $\alpha = x + iy$ and $\beta = z + iw$, the squared norms $|\alpha|^2 = x^2 + y^2$ and $|\beta|^2 = z^2 + w^2$ represent the probability of measuring the qubit thus reading 0 and 1, respectively. In the more general case of m qubits the unitary vectors range in $\mathbb{C}^n$ with $n = 2^m$. Adopting the standard Dirac notation we denote a column vector $v \in \mathbb{C}^n$ by $|v\rangle$, and its conjugate transpose $v^\dagger$ by $\langle v|$. A *quantum state* is a unitary vector

$$|\psi\rangle = \sum_h c_h \, |v_h\rangle$$

for some basis $\{|v_h\rangle\}$. When not specified, we refer to the *canonical basis*. Further details can be found in [23].

Unitary operators and projectors are linear operators, so before introducing their definitions, we fix the notation on matrices. We rely on the same notation also for graphs represented through adiacency matrices in classical computation.

Let M be a matrix, $M_{i,j}$ is the element in the i-th row and j-th column of M. Moreover, we denote by M_i the entire i-th row. Whenever we refer to the product of two rows of a matrix, we refer to the *dot product* (scalar product) between the former and the conjugate transpose of the latter, i.e., $M_i M_j = \sum_k M_{i,k} M^*_{j,k}$, where $M^*_{j,k}$ is the complex conjugate of $M_{j,k}$.

Unitary operators are a particular class of reversible linear operators. They preserve both the angles between vectors and their lengths. In other terms, unitary operators are transformation from one orthonormal basis to another. Hence, they are represented by unitary matrices. Let U be a square matrix over $\mathbb{C}$. U is said to be *unitary* iff $UU^\dagger = U^\dagger U = I$. We describe the application of a unitary matrix U to a state $|\psi\rangle$ by writing

$$|\psi'\rangle = U \, |\psi\rangle$$

meaning that the state $|\psi\rangle$ becomes $|\psi'\rangle$ after applying the operator U.

In order to extract informations from a quantum state $|\phi\rangle$ a *measurement* must be performed. The most common measurements are projectors. Let $|u\rangle$ be a quantum state. The *projector* operator P_u along the direction of $|u\rangle$ is the linear operator defined as:

$$P_u = \frac{|u\rangle\langle u|}{\langle u|u\rangle}$$

where $|u\rangle\langle u|$, being the product between a column vector and a row one both of size n, returns a matrix of size $n \times n$. Since throughout the paper we only use unitary vectors, the term $\langle u|u\rangle$ is always 1 and can be ignored.

1.2 Graph Theory

Graphs are a standard data structure in computer science for the representation of binary relations. They are used in Computability Theory for representing the flow of computations [24], in Network Theory to describe the network topology [2], in Routing [29], etc. We report below some standard definitions on graphs, while we refer the reader to [15] for further details. A directed *graph* G is a pair (V, E) where V is a non empty set of *nodes* and $E \subseteq V \times V$ is a set of *edges*. We say that an edge (u, v) is *directed* from u to v and that u is *adjacent* to v. We also say that u is the *source* and v is the *target* of (u, v). Two edges of the form (u, v) and (v, w) are said to be *consecutive*. We say that a graph is *undirected* iff E is symmetric. The *underlying undirected graph* of G is a new graph whose edge set is the symmetric closure of E.

In classical computation, graphs are usually stored as either adjacency matrices or adjacency lists. In this paper we mainly refer to the adjacency matrix representation. Given a set of nodes V, we assume a fixed order over the elements of V, i.e., each node of V can be identified as an integer between 1 and $|V|$. For a graph $G = (V, E)$ with $|V| = n$, the *adjacency matrix* of G is a (0-1)-matrix M of size $n \times n$ such that $M_{u,v} = 1$ iff $(u, v) \in E$.

A *multigraph* $G = (V, E)$ is a graph in which many edges can connect the same pair of nodes. So, if there are two edges from u to v these are two distinct primitive objects. In other terms, E is a set (of edges) and two functions $s, t : E \to V$ assign to each edge both a source node and a target one. For the sake of readability, we will use the notation (u, v) also on multigraphs to refer to a generic edge having source u and target v. The aim of this paper is providing a quantum representation for graphs. However, it is useful to refer to multigraphs which allow us to overcome some technical issues.

Definition 1 (Incoming and Outgoing edges). *Let $G = (V, E)$ be a multigraph, $v \in V$. We define the set of its* incoming *edges as:*

$$\delta^-(v) = \{i : i \in E \text{ and } i \text{ has target } v\}$$

and the set of its outgoing *edges as:*

$$\delta^+(v) = \{k : k \in E \text{ and } k \text{ has source } v\}$$

Moreover, the indegree *of v is defined as $d^-(v) = |\delta^-(v)|$ and analogously its* outdegree *is defined as $d^+(v) = |\delta^+(v)|$.*

The *balance* of a node v is $b(v) = d^+(v) - d^-(v)$. A multigraph $G = (V, E)$ is *balanced* iff for each node $v \in V$, $b(v) = 0$. This property is crucial for directly encoding multigraphs in terms of Quantum Computation since it ensures the possibility of reversing walks over the graph [1]. The following is a well known result from Graph Theory which basically states that the global balance of a multigraph is always null.

Theorem 1. *Let $G = (V, E)$ be a multigraph. Let $B^+ = \{v \in V \ : \ b(v) > 0\}$ and $B^- = \{v \in V \ : \ b(v) < 0\}$. Then,*

$$\sum_{v \in B^+} b(v) + \sum_{v \in B^-} b(v) = 0$$

As for graphs, also on multigraphs a *path* is a sequence of distinct adjacent nodes. A *cycle* is a path where the last node coincides with the first one. A multigraph that does not contain cycles is said to be acyclic. While in the case of graphs, paths can be equivalently defined as sequences of consecutive edges, in the case of multigraphs the definition based on consecutive edges is more informative. In order to avoid confusion, we refer to such definition using the term *tour*. *Eulerian tours* traverse each edge of the multigraph exactly once. A multigraph is *eulerian* iff it admits an eulerian tour which is also a cycle.

The notions of balanced multigraphs and eulerian multigraphs coincide when we consider only connected graphs. We introduce here some more notions about connectivity that will be useful in our technique for encoding multigraphs in quantum data structures. Two nodes of a graph are *mutually reachable* iff there exists a path from the first to the second and viceversa.

A multigraph G is said to be *strongly connected* if each pair of nodes $u, v \in V$ are mutually reachable. It is said to be weakly connected, or just *connected*, if its underlying undirected graph is strongly connected. The following result formalizes the equivalence between balanced and eulerian multigraphs [9].

Theorem 2. *A connected multigraph G is balanced iff it is eulerian.*

In [28] it has been proved that there exists a relationship between eulerian graphs and unitary matrices. Such relationship involves the notion of line graph.

Definition 2 (Line Graph). *Let $G = (V, E)$ be a multigraph. The* line graph *of G is the graph $\overrightarrow{G} = (\overrightarrow{V}, \overrightarrow{E})$, where:*

$$\overrightarrow{V} = E \qquad \overrightarrow{E} = \big\{\big(i, k\big) : i, k \in E \text{ and } i \text{ and } k \text{ are consecutive}\big\}$$

The elements of a unitary matrix are not necessarily 0 and 1. To associate a graph to a matrix, we first introduce the notion of *support* of a matrix. Given a matrix M its support is the (0-1)-matrix M^S where $M^S_{i,j} = 1$ iff $M_{i,j} \neq 0$.

Definition 3 (Graph of a Matrix). *The* graph of a matrix M *is the graph whose adjacency matrix is* M^S.

In [28] it has been proved that the line graph of an eulerian multigraph has an adjacency matrix that is the support of a unitary one.

Theorem 3. *Let* G *be a multigraph and* $\overrightarrow{G}$ *be its line graph. Then* $\overrightarrow{G}$ *is the graph of a unitary matrix iff* G *is eulerian or the disjoint union of eulerian components.*

Since the aim of this paper is to implement quantum random walks on graphs, it is useless to consider disconnected multigraphs. Therefore, in what follows we will consider only connected multigraphs.

1.3 Quantum Random Walks

In classical computation, when we visit a graph choosing randomly the next edge to cross, we say that we are making a *random walk* on the graph. In quantum computing, the counterpart of this concept is the *quantum random walk (QRW)*. Since the only way to change a quantum state is through unitary matrices, the most reasonable way to implement a QRW is based on the ability of encoding the adjacency properties of any graph inside a unitary matrix.

The visit, in both classical and quantum computation, can be made at either continuous or discrete time. In the case of continuous time QRWs, the Hamiltonian of the system is provided and using matrix exponentiation the system evolves through time [20]. On the other hand, in discrete time QRWs, the most common method is through *coined* walks [1]. Such method is based on a pair of Hilbert spaces: the first one is for the graph nodes, while the second one is for the coin. Encoding the graph with this technique generates an unitary matrix which makes the computation reversible.

2 Unitary Matrix of an Eulerian Multigraph

In this section we introduce a procedure which allows us to transform any eulerian multigraph G into a unitary matrix, encoding the adjacency properties of G. The procedure passes through the construction of the line graph of G, then Theorem 3 from [28] lies at the heart of the transformation. However, the focus in [28] is on the proof of the result, more than on the algorithmic construction of a unitary matrix. Instead, in this section we are going to actually build a unitary matrix starting from a line graph adjacency matrix. Such unitary matrix is not unique and our construction is parametric with respect to a family of unitary matrices which are required as input. As for the computational complexity of the technique, we anticipate that it is linear in the size of the resulting matrix. Notice that it is common usage to build unitary matrices by columns. In our approach, since we edit the adjacency matrix, the resulting unitary will be by rows. In this way we keep a closer relationship to the initial graph. Therefore, the reader should be aware that in the circuit and in the examples we will use the conjugate transpose of the matrix.

2.1 From a Graph to its Line

Let $G = (V, E)$ be an eulerian multigraph. The function LINEARIZE in Algorithm 1 returns the adiacency matrix $\widetilde{M}$ of the line graph $\overrightarrow{G}$. It can be easily checked that its time complexity is $\Theta(|E|^2)$.

Algorithm 1 Construct the line graph of a given multigraph.

```
1: function LINEARIZE(V, E)
2:     M̃ ← SQUAREMATRIX(|E|)          ▷ Creating an all zero square matrix
3:     for all i ∈ E do
4:         v ← TARGET(i)                ▷ i is of the form (u, v)
5:         for all k ∈ δ⁺(v) do         ▷ k is of the form (v, w)
6:             M̃_{i,k} ← 1
7:     return M̃
8: end function
```

We now point out some structural properties of $\widetilde{M}$ that follow from the fact that G is eulerian.

Lemma 1. *Let $i = (u, v)$ and $j = (u', v')$ be two edges. It holds that:*

$$\widetilde{M}_i = \widetilde{M}_j \quad \textit{iff} \quad v = v'$$

Proof. Let i, j be two edges of G. Consider some edge $k \in E$. By definition of $\overrightarrow{G}$, $\widetilde{M}_{i,k} = 1$ iff i and k are consecutive edges. The same is true for j. So, if $v = v'$ it is immediate to conclude that $\widetilde{M}_i = \widetilde{M}_j$. Otherwise, if $\widetilde{M}_i = \widetilde{M}_j$, then since G is eulerian it cannot be the case that $\widetilde{M}_i$ and $\widetilde{M}_j$ have only 0 elements. This means that $\exists k$ such that $\widetilde{M}_{i,k} \neq 0$. Hence, by hypothesis also $\widetilde{M}_{j,k} \neq 0$. Let $k = (v'', w)$, since $\widetilde{M}_{i,k} \neq 0$ it has to be $v = v''$. Moreover, from $\widetilde{M}_{j,k} \neq 0$ we get $v' = v''$. So, $v = v'$. □

As a consequence of the above lemma we immediately get that each node v induces as many equal rows in $\widetilde{M}$ as its indegree.

Lemma 2. *For every edge $i = (u, v)$, there are exactly $d^+(v)$ edges j such that $\widetilde{M}_{i,j} \neq 0$. Moreover, there are exactly $d^-(v)$ rows equal to $\widetilde{M}_i$.*

In Lemmata 1 and 2 we introduce a relationship between rows describing edges that have a common target. We can generalize the result formalizing it in the case of any pair of rows.

Lemma 3. *Let $i, j \in E$. Let $A = \{k : \widetilde{M}_{i,k} \neq 0\}$ and $B = \{l : \widetilde{M}_{j,l} \neq 0\}$. Then, either $A = B$ or $A \cap B = \varnothing$.*

Algorithm 2 Compute a unitary matrix from the line graph.

```
1: function UNITARIZE(M̃, 𝒰)
2:     M̂ ← M̃
3:     Q ← V
4:     while Q ≠ ∅ do
5:         v ← POP(Q)
6:         U ← 𝒰(d⁻(v))                ▷ Choose some d⁻(v) × d⁻(v) matrix from 𝒰
7:         c ← 1
8:         for all i ∈ δ⁻(v) do
9:             SPARSESUB(M̂_i, U_c)
10:            c ← c + 1
11:    return M̂
12: end function
13:
14: procedure SPARSESUB(r, r′)
15:    h ← 1
16:    for all k ∈ E do
17:        if r_k ≠ 0 then
18:            r_k ← r′_h
19:            h ← h + 1
20: end procedure
```

Proof. If $A = B$ there is nothing to prove. Suppose $A \neq B$ and assume, for the sake of contradiction, that $k = (v, w) \in A \cap B$. By definition of A and B, it has to be $\widetilde{M}_{i,k} = 1 = \widetilde{M}_{j,k}$. Hence i and j share v as common target. By Lemma 1, $\widetilde{M}_i = \widetilde{M}_j$ and $A = B$. This is a contradiction. □

In what follows we say that two rows are *disjoint* if $A \cap B = \varnothing$, where A, B are defined as in the above lemma. In other terms, two rows are disjoint if and only if they correspond to edges having different targets.

2.2 Construction of the Unitary Matrix

We now describe the procedure UNITARIZE which transforms $\widetilde{M}$ into a unitary matrix $\widehat{M}$ whose support is $\widetilde{M}$. The input of the function are the adjacency matrix $\widetilde{M}$ and a family of unitary matrices $\mathcal{U}$. The family has to satisfy:

$$\forall v \in V \; \exists U \in \mathcal{U} \;\; \text{of size } d^-(v) \times d^-(v)$$

The goal of UNITARIZE in Algorithm 2 is to edit $\widetilde{M}$ in order to obtain a unitary matrix which encodes the adjacency properties of G. At each iteration of the while loop, a node v is extracted from Q and, for each edge i directed to v, row $\widetilde{M}_i$ is edited through the procedure SPARSESUB exploiting a unitary matrix of suitable dimension. In particular, SPARSESUB(r, r') replaces the h-th non-zero element of r by r'_h.

Let r be a vector and A be a set of indexes of r, we use the notation $r(A)$ to denote the subvector of r obtained by considering only the indexes in A.

Lemma 4. *After applying* SparseSub *at line 9, the subvector* $\widehat{M}_i(\delta^+(v))$ *is equal to* U_c.

Proof. By definition of $\widetilde{M}$, $r_k \neq 0$ holds iff $k \in \delta^+(v)$. By construction, U_c has size $n = d^-(v) = d^+(v) = |\delta^+(v)|$. The result follows immediately. □

Theorem 4. $\widehat{M}$ *is a unitary matrix.*

Proof. It is sufficient to show that the rows of $\widehat{M}$ form an orthonormal basis. By Lemma 4, each row of $\widehat{M}$ is a row of some unitary matrix interleaved by zeros. Hence, the product of each row with itself is 1. Let $i, j \in E$ be distinct edges. If $\widetilde{M}_i$ and $\widetilde{M}_j$ are disjoint, also $\widehat{M}_i$ and $\widehat{M}_j$ are. Therefore $\widehat{M}_i\widehat{M}_j = 0$. Otherwise, by construction of Unitarize, $\widehat{M}_i$ and $\widehat{M}_j$ refer to the same unitary matrix. Since their product depends only on their non-zero elements, and since they correspond to unitary rows, $\widehat{M}_i\widehat{M}_j = 0$. □

Since each line of $\widehat{M}$ is edited once and $\mathcal{U}$ can be stored efficiently, we get that Unitarize has time complexity $\Theta(|E|^2)$.

Example 1. We now provide an example to clarify the structure of $\widehat{M}$ after the procedure Unitarize. For sake of readability, we omitted the scalar multipliers from the matrices in $\mathcal{U}$ inside $\widehat{M}$.

$$\mathcal{U} = \left\{ \frac{1}{\sqrt{3}} \begin{pmatrix} 1 & 1 & 1 \\ 1 & \omega & \omega^2 \\ 1 & \omega^2 & \omega^4 \end{pmatrix}, \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right\} \qquad \omega = \exp\left\{\frac{2\pi i}{3}\right\}$$

$$\widetilde{M} = \begin{pmatrix} 1 & 1 & & & & & \\ & & 1 & 1 & 1 & & \\ & & 1 & 1 & 1 & & \\ & & & & & 1 & 1 \\ & & 1 & 1 & 1 & & \\ & & & & & 1 & 1 \\ 1 & 1 & & & & & \end{pmatrix} \qquad \widehat{M} = \begin{pmatrix} 1 & 1 & & & & & \\ & & 1 & 1 & 1 & & \\ & & 1 & \omega & \omega^2 & & \\ & & & & & 0 & 1 \\ & & 1 & \omega^2 & \omega^4 & & \\ & & & & & 1 & 0 \\ 1 & -1 & & & & & \end{pmatrix}$$

3 Embedding Multigraphs into Eulerian ones

In the previous section we showed how to produce a unitary matrix from an eulerian multigraph. The aim of this section is to introduce a procedure called Eulerify, based on Theorem 2, that takes as input the nodes set V of a connected multigraph $G = (V, E)$ and the array b of the balances of the nodes. It gives as output a set of additional edges $E_\perp$ such that the multigraph $G' = (V, E \cup E_\perp)$

is eulerian. We recall that, since we are interested in quantum random walks, we are interested only in connected graphs.

Algorithm 3 Edit the graph to make every node balanced.

```
1: function EULERIFY(V, b)
2:     E_⊥ ← ∅
3:     B^+ ← {v ∈ V : b_v > 0}
4:     B^- ← {v ∈ V : b_v < 0}
5:     while B^- ≠ ∅ do
6:         u ← POP(B^-)
7:         while b_u < 0 do
8:             v ← CHOOSE(B^+)                ▷ Choose without extracting
9:             E_⊥ ← E_⊥ ∪ {(u, v)}
10:            (b_u, b_v) ← (b_u + 1, b_v − 1)
11:            if b_v = 0 then
12:                B^+ ← B^+ \ {v}
13:    return E_⊥
14: end function
```

The procedure takes as input the set of nodes V together with the vector b of size $|V|$ initialized with the balance of the nodes, i.e., the v-th element of b is $b_v = b(v)$. The idea is to iteratively fix each node in deficiency of balance adding edges to nodes in surplus of balance. The choice of the deficient node u is done at line 6. The loop at lines 7–12 is responsible for adding edges from u to surplus nodes v until u is balanced. Theorem 1 both ensures that for each u there are always surplus nodes to choose at line 8 and that when we exit the loop 5–12 the sets B^+ and B^- are empty.

It is clear that the time computational complexity of EULERIFY is proportional to $\Theta(|V| + |E_\perp|) \subseteq O(|V| + |E|)$, since we never add more edges than the existing ones. This is the technical point were the use of multigraphs helps us. We can both add new edges and new copies of existing ones.

The adjacency properties of G could be different from those of G'. In particular, such differences are witnessed by the set $E_\perp$. Our aim is to visit the graph G through a quantum random walk. However, since G' is eulerian, the walk will be performed exploiting the unitary matrix constructed on the line graph of G'. So, we must ensure that such walk only uses edges of E. Therefore, all the edges inside $E_\perp$ are used in the UNITARIZE procedure, but will be forbidden during the walk. In order to do this we will use a projector defined as follows:

$$P_G = I - \sum_{e \in E_\perp} |e\rangle \langle e|$$

Notice that even if in $E_\perp$ we add an edge between two adjacent nodes of G, the projector removes the new edge, while it does not affect the one existing in G.

Example 2. The following toy example show the construction of P_G.

$$E = \{00, 01, 10, 11\} \qquad E_\perp = \{01, 11\} \qquad P_G = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

4 The Quantum Circuit: with and without Projections

In the previous sections we introduced two procedures, namely UNITARIZE and EULERIFY. While the first is in charge of constructing a unitary matrix from an eulerian multigraph, the second one embeds a connected multigraph into an eulerian one. In this section we put the two procedures together and describe the resulting quantum circuit running the quantum walk. Finally, we propose a way to reduce the size of such circuit avoiding the use of projectors in the case of strongly connected graphs.

Given any connected multigraph $G = (V, E)$, we first check whether it is balanced. If not, we apply the procedure EULERIFY obtaining some G' – which is now eulerian. We also compute a projector P_G which "eliminates" the effect of the edges of $E_\perp$ along the walk. After that, we use the procedure UNITARIZE on G' to obtain a unitary matrix $\widehat{M}$ that would allow us to make one discrete step in the visit of G'. So, a walk on G can be implemented by using each time P_G after $\widehat{M}$ as shown in Figure 1.

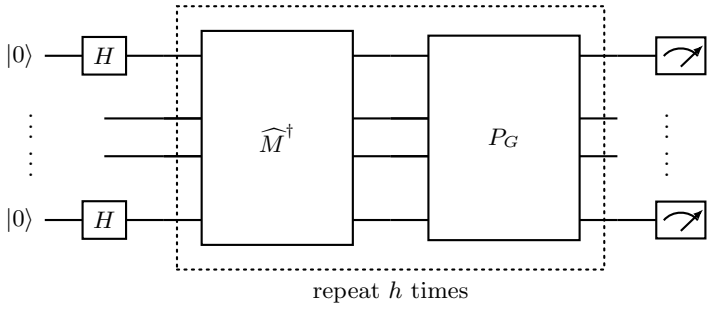


Fig. 1: Quantum circuit for the walk.

In particular, in the circuit in Figure 1 a number of qubits proportional to the logarithm of the edges of G' have to be initialized to $|0\rangle$. Then, an equiprobable superposition of edges is generated using Hadamard operator H. At this point, by applying $\widehat{M}$ and P_G to the state for h times, h steps of the walk on G are performed. Finally, the measurement returns one edge of G whose target node can be interpreted as the last node of the quantum walk.

In terms of circuit complexity, the repeated application of P_G is time consuming. For this reason, we briefly introduce an alternative solution to the problem of embedding a multigraph into an eulerian one. Such alternative solution avoids the use of projections. The idea is that of balancing the multigraph by adding copies of edges that are already present. In this way, the adjacencies of G will not be altered and the projector P_G will not be needed anymore.

First, please notice that if G is not strongly connected, there is no way to embed it into a eulerian multigraph without adding authentic new edges. So, let us assume that G is strongly connected. We are interested in introducing the minimum number of copies of edges to make the graph eulerian: this is in fact an instance of the Directed Chinese Postman Problem. There is extensive literature on the problem, see e.g. [12], but for our purpose it is sufficient to know that for strongly connected graphs it always admits a solution computable in polynomial time. In terms of physical circuit implementation, if we adopt such solution the matrix P_G would disappear from the circuit in Figure 1.

5 Edges/Nodes Failures

The overall procedure described in the previous sections has a time computational complexity proportional to $\Theta(|E|^2)$ and generates matrices of such size. As we will see in Section 6 other methods in literature have the advantage of relying on matrices of size $\Theta(|V|^2)$ (e.g., [20]). However, as a trade off, we show how in our method editing in the original graph G in terms of deletion of either edges or nodes does not result in the matrix $\widehat{M}$ to be recomputed. The only involved cost comes from the update of the projector P_G introduced in Section 3. Editing P_G for a single edge requires time $\Theta(1)$, while it requires time $\Theta(d^+(v) + d^-(v))$ for removing a node. In fact, the application of P_G after every step of the walk (see Figure 1) has the effect of *hiding* some edges.

We know that each step of the visit is a composition of the unitary matrix $\widehat{M}$ followed by an application of the projector P_G. If we suppose that an edge i is temporarily removed from the graph, e.g., a failure in the network occurs, we do not need to edit the matrix $\widehat{M}$. We just need to update a single element of P_G. In terms of matrix operations, we know that i is an element of the basis of the edges. Therefore, we just need to edit P_G setting the i-th element of its diagonal to 0. Notice that P_G only has 1s on some elements of the diagonal and 0 elsewhere. In terms of computational complexity, this operation takes time $\Theta(1)$. At later stages, if the same edge is re-added, e.g., the failure is fixed, it takes again time $\Theta(1)$ to undo the previous edit on P_G. Therefore, our encoding efficiently supports both deleting edges and re-adding them.

In case of edge addition, the matrix $\widehat{M}$ should be entirely recomputed. However, the proposed method can be adapted to be efficient also in the case of edge additions at the cost of a more space-consuming matrix $\widehat{M}$. In particular, consider the case in which we have a set N of edges that could potentially be added to G. This could be the case for a network infrastructure that we know will be strengthened in a near future adding connections that have already been recog-

nized as strategic. At present the graph representing the network is $G = (V, E)$, but in a near future it will become $GN = (V, E \cup N)$. In this case we construct the matrix $\widehat{M}$ for GN and we project away the edges of N. As soon as the new edges are available, we re-introduce them by modifying the projector.

At a high cost in term of space, one could decide to work by assuming that each edge could be added. This is equivalent to consider N as $V \times V$. The reader should be aware that in the worst case the size of GN could be quadratic with respect to the size of G. This occurs when G is sparse.

Finally, if we consider a node failure as a failure of all its incoming and outgoing edges, then also node failure can be easily handled. In fact, if node v fails, the cost of removing all of its edges is exactly $\Theta(d^+(v) + d^-(v))$.

6 Related Literature and Examples

In what follows we compare the proposed method with two alternatives from literature. Both techniques tackle the problem of QRWs and therefore they need a way to encode a graph by means of a unitary matrix. The first one [20] regards *continuous time* QRWs. Therefore, the authors use matrix exponentiation to obtain a unitary matrix, typical of continuous time quantum processes. Meanwhile, the second technique [1] deals with *discrete time* QRWs. It labels the edges of the graph in such a way that each node has exactly one incoming and one outgoing edge per label. It then equips each node of degree d with a $d \times d$ unitary matrix – the *coin* – which encodes the possible choices of the next edge to cross.

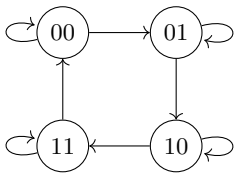


Fig. 2: Four nodes cyclic graph.

6.1 Continuous Time Walk from [20]

Let G be the cyclic graph depicted in Figure 2 and M be its adjacency matrix. Let $\pi = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$ be the unique non-negative eigenvector of M, $\Pi = \textsc{diag}(\pi)$ and $L = \Pi - \frac{1}{2}(\Pi\, M + M^\dagger\, \Pi)$

$$L = \frac{1}{2}\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

The unitary matrix used for the walk at time $t \in \mathbb{R}$ is $U^t = e^{-itL}$.

The matrix is not directly related to the graph topology. Moreover, if an edge-failure occurs, the matrix M changes, and both Π and L need to be recomputed. However, in [20] the focus was not on developing visit algorithms but on defining a measure of similarity between graphs.

6.2 Discrete Time Walk from [1]

We consider again the graph of Figure 2. In the case of this example the *coin* matrix C and the *shift* matrix S are as follows

$$C = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad S = \begin{pmatrix} 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \end{pmatrix}$$

The self loops were labelled – colored – with the same color.

The resulting unitary matrix is

$$U = S \cdot (C \otimes I) = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\ \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 \\ \mathbf{1} & 0 & 0 & 0 & \mathbf{-1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{-1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{-1} & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{-1} \end{pmatrix}$$

In this case the matrix seems to be more descriptive in terms of the initial graph topology. However, in the case of an edge-failure, the procedure must start by recomputing again the edge labelling.

6.3 Our Method

We now want to obtain a unitary matrix for G in Figure 2 with our new method. Since G is eulerian, the set $E_\perp$ is empty. The adjacency matrix of $\overrightarrow{G}$ is

$$\widetilde{M} = \begin{pmatrix} \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

We now apply UNITARIZE($\widetilde{M}, \mathscr{U}$) to obtain $\widehat{M}$

$$\mathscr{U} = \left\{ \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right\} \qquad \widehat{M} = \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{-1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{-1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{-1} \\ \mathbf{1} & \mathbf{-1} & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The above matrix was computed fixing the following encoding of the edges of G

$$\begin{array}{llll} |000\rangle = (00,00) & |001\rangle = (00,01) & |010\rangle = (01,01) & |011\rangle = (01,10) \\ |100\rangle = (10,10) & |101\rangle = (10,11) & |110\rangle = (11,11) & |111\rangle = (11,00) \end{array}$$

Suppose we start in state $|\psi_0\rangle = |000\rangle$. After one step, we are in state

$$|\psi_1\rangle = \widehat{M}^\dagger |000\rangle = \frac{1}{\sqrt{2}} \big(|000\rangle + |001\rangle \big)$$

and after another step we reach

$$|\psi_2\rangle = \frac{1}{2} \big(|000\rangle + |001\rangle + |010\rangle + |011\rangle \big)$$

Notice that since $E_\perp = \varnothing$, the associated projector P_G is the identity matrix.

Suppose that after the second step the edge $|001\rangle$ fails. Using our method we can react to this event by adding the edge $(00,01)$ to $E_\perp$. The matrix P_G is updated as introduced in the previous section, and it becomes $I - |001\rangle \langle 001|$.

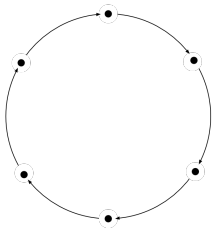
Hence, by applying P_G to $|\psi_2\rangle$ we would delete $|001\rangle$ from the state.

6.4 Random Thoughts on Random Walks

Last but not least, for those who are familiar with Markov Chains, we add a brief note about QRWs distributions. In particular, we give an intuitive argument about the non-existence of a stationary distribution as for the classical case. Let $U \in \mathbb{C}^{n \times n}$ be a unitary matrix. The *graph of the states of* U is $G_U = (V_U, E_U)$:

$$V_U = \big\{ |\psi\rangle \ : \ \langle\psi|\psi\rangle = 1, |\psi\rangle \in \mathbb{C}^n \big\} \qquad E_U = \big\{ (|\psi\rangle, U|\psi\rangle) \ : \ |\psi\rangle \in V_U \big\}$$

It is straightforward to see that, since U is unitary – hence invertible – each $|\psi\rangle \in V_U$ has exactly one incoming and one outgoing edge. Therefore, G_U is disjoint union of cycles and infinite lines. In particular, if there exists a $t \in \mathbb{N}^+$ such that $U^t |\psi\rangle = |\psi\rangle$, then $U^i |\psi\rangle$, $i \in \mathbb{Z}$ forms a cycle. If such t does not exists, then $|\psi\rangle$ will be a part of an infinite line of states.



(a) A cyclic state evolution.

(b) An infinite line state evolution.

7 Conclusions

Quantum computing is becoming more and more important in computer science. The laws of Quantum Mechanics, that rule quantum algorithms, require every operation to belong to a restricted class of linear operators. Therefore, also common data structures like graphs must be encoded in such class of matrices in order to be visited using quantum algorithms. In our proposal we introduced a method to encode any graph into a unitary matrix using the notion of *eulerian* graph, and eventually a projector. This new technique produces a matrix that has a closer relationship to the initial graph topology than other methods in literature (i.e. [20], [1]). Moreover, the presence of a projector is useful to support edge-failures. We showed that by just changing one single element in the projector we can react to the failure of an edge. We also described how to use our technique in order to obtain edge addition.

As future work we are interested in investigating the limiting distributions of our QRWs. For a restricted class of graphs we are already able to obtain the same results of [1]. However, simulations suggest that the class can be extended.

Moreover, as briefly said in Section 4, the Direct Chinese Postman Problem (DCPP) can be exploited in order to avoid projectors at each step. We intend to further study in this direction with particular reference to the problem of keeping the probabilities unchanged after the application of DCPP.

Finally, we are interested in developing graph reduction techniques, such as lumpabilities [3], in order to reduce the quantum circuits size.

References

[1] Dorit Aharonov et al. “Quantum walks on graphs”. In: *Proceedings of the thirty-third annual ACM symposium on Theory of computing*. 2001, pp. 50–59.

[2] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. “Network flows - theory, algorithms and applications”. In: 1993.

[3] Giacomo Alzetta et al. “Lumping-based equivalences in Markovian automata: Algorithms and applications to product-form analyses”. In: *Information and Computation* 260 (2018), pp. 99–125.

[4] Andris Ambainis. "Quantum walks and their algorithmic applications". In: *International Journal of Quantum Information* 1.04 (2003), pp. 507–518.
[5] Andris Ambainis, Leonard J Schulman, and Umesh V Vazirani. "Computing with highly mixed states". In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. 2000, pp. 697–704.
[6] Eli Biham et al. "Quantum computing without entanglement". In: *Theoretical Computer Science* 320.1 (2004), pp. 15–33.
[7] Fan Chung. "Laplacians and the Cheeger inequality for directed graphs". In: *Annals of Combinatorics* 9.1 (2005), pp. 1–19.
[8] Erhan Çinlar. *Probability and stochastics*. Vol. 261. Springer, 2011.
[9] Marek Cygan et al. "Parameterized complexity of Eulerian deletion problems". In: *Algorithmica* 68.1 (2014), pp. 41–61.
[10] David Deutsch and Richard Jozsa. "Rapid solution of problems by quantum computation". In: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439.1907 (1992), pp. 553–558.
[11] David Easley and Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge university press, 2010.
[12] Jack Edmonds and Ellis L Johnson. "Matching, Euler tours and the Chinese postman". In: *Mathematical programming* 5.1 (1973), pp. 88–124.
[13] Edward Farhi and Sam Gutmann. "Quantum computation and decision trees". In: *Physical Review A* 58.2 (Aug. 1998), pp. 915–928. ISSN: 1094-1622.
[14] Chris Godsil and Hanmeng Zhan. "Discrete-time quantum walks and graph structures". In: *Journal of Combinatorial Theory, Series A* 167 (2019), pp. 181–212.
[15] Frank Harary. *Graph Theory*. Addison Wesley series in mathematics. Addison-Wesley, 1971.
[16] Simon Haykin. *Neural networks and learning machines, 3/E*. Pearson Education India, 2010.
[17] Richard Jozsa and Noah Linden. "On the role of entanglement in quantum-computational speed-up". In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 459.2036 (2003), pp. 2011–2032.
[18] Frédéric Magniez et al. "Search via quantum walk". In: *SIAM journal on computing* 40.1 (2011), pp. 142–164.
[19] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems: safety*. Springer Science & Business Media, 2012.
[20] Giorgia Minello, Luca Rossi, and Andrea Torsello. "Can a quantum walk tell which is which? a study of quantum walk-based graph similarity". In: *Entropy* 21.3 (2019), p. 328.
[21] Ashley Montanaro. "Quantum walks on directed graphs". In: *arXiv preprint quant-ph/0504116* (2005).
[22] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge university press, 1995.

[23] Michael A Nielsen and Isaac Chuang. *Quantum computation and quantum information*. 2002.
[24] Christos H. Papadimitriou. "Computational complexity". In: 1993.
[25] Christos H. Papadimitriou. "On the Complexity of Edge Traversing". In: *J. ACM* 23.3 (1976), pp. 544–554.
[26] Giuseppe Davide Paparo and MA Martin-Delgado. "Google in a quantum network". In: *Scientific reports* 2.1 (2012), pp. 1–12.
[27] Daowen Qiu and Shenggen Zheng. "Revisiting Deutsch-Jozsa Algorithm". In: *Information and Computation* 275 (2020), p. 104605.
[28] Simone Severini. "On the digraph of a unitary matrix". In: *SIAM Journal on Matrix Analysis and Applications* 25.1 (2003), pp. 295–300.
[29] Andrew S. Tanenbaum and David Wetherall. *Computer networks, 5th Edition*. 2011.
[30] Salvador Elias Venegas-Andraca. "Quantum walks: a comprehensive review". In: *Quantum Information Processing* 11.5 (2012), pp. 1015–1106.
[31] Feng Xia et al. "Random Walks: A Review of Algorithms and Applications". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 4.2 (2020), pp. 95–107.