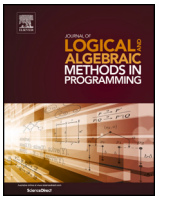
J. Log. Algebraic Methods Program. 136 (2024) 100925

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

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



# Quantum encoding of dynamic directed graphs ☆

D. Della Giustina, C. Londero, C. Piazza, B. Riccardi, R. Romanello *

*Department Mathematics, Computer Science, and Physics, University of Udine, Via Le Scienze, 206, Udine, 33100, Italy*

ARTICLE INFO

*Keywords:*
Quantum walks
Graphs
Edge-failure

ABSTRACT

In application domains such as routing, network analysis, scheduling, and planning, directed graphs are widely used as both formal models and core data structures for the development of efficient algorithmic solutions. 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 graphs have been implemented both explicitly through adjacency matrices/lists and symbolically as ordered binary decision diagrams. Moreover, ad-hoc visit procedures have been developed to deal with dynamically evolving graphs.

Quantum computation, exploiting interference and entanglement, has provided an exponential speed-up for specific problems, e.g., database search and integer factorization. In the quantum framework everything must be represented and manipulated using reversible operators. This poses a challenge when one has to deal with traversals of dynamically evolving directed graphs. Graph traversals are not intrinsically reversible because of converging paths. In the case of dynamically evolving graphs also the creation/destruction of paths comes into play against reversibility.

In this paper we propose a novel high level graph representation in quantum computation supporting dynamic connectivity typical of real-world network applications. Our procedure allows to encode any *multigraph* into a unitary matrix. We devise algorithms for computing the encoding that are optimal in terms of time and space and we show the effectiveness of the proposal with some examples. We describe how to react to edge/node failures in constant time. Furthermore, we present two methods to perform quantum random walks taking advantage of this encoding: with and without *projectors*. We implement and test our encoding obtaining that the theoretical bounds for the running time are confirmed by the empirical results and providing more details on the behavior of the algorithms over graphs of different densities.

## 0. Introduction

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., [1,2]). Moreover, it allows to solve with bounded probability of error in polynomial

☆ This work is partially supported by PRIN project NiRvAna - 20202FCJM CUP G23C22000400005 and GNCS INdAM project "Algoritmi Quantistici su Grafi" CUP E53C22001930001.

* Corresponding author.
*E-mail addresses:* dellagiustina.davide@spes.uniud.it (D. Della Giustina), londero.christian001@spes.uniud.it (C. Londero), carla.piazza@uniud.it (C. Piazza), riccardi.brian@spes.uniud.it (B. Riccardi), riccardo.romanello@uniud.it (R. Romanello).

https://doi.org/10.1016/j.jlamp.2023.100925
Received 11 January 2023; Received in revised form 23 October 2023; Accepted 23 October 2023

Available online 27 October 2023

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., [3–5]).

However, quantum computation being mainly based on reversible (unitary) operator imposes an increase in space requirements when non-reversible operations have to be implemented/simulated. For instance, a standard technique for quantum simulation of classical circuits (i.e., classical Boolean functions) consists in representing the Boolean function $f : \{0,1\}^n \to \{0,1\}^m$ as the reversible function $f' : \{0,1\}^{n+m} \to \{0,1\}^{n+m}$ with $f'(x,y) = (x, y \oplus f(x))$.

Graphs and graph traversals 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 Automata Theory [6,7], Complexity Theory [8], Flow Theory [9] and Software Verification [10,11], to more application-oriented domains such as Routing [12], Machine Learning [13] and Social Networks [14]. In the case of ever growing and dynamically evolving networks, exhaustive visits become unfeasible and have to be replaced by random walks. With the advent of quantum computing as a more and more useful tool, it is mandatory to define efficient encodings for graphs and random walks in quantum circuits.

The theory of quantum walks, the quantum counterpart of random walks, has been first introduced in [15]. As for the classical case [16,17], there is a distinction between continuous and discrete time models [18]. The former is introduced in [19] and further investigated in [20,21]: 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 [15] 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 [22,23].

The aim of our work is to encode graphs in the quantum formalism, so that algorithms, such as random walks, can be developed also in dynamic settings where edges/nodes can be temporarily unavailable. We represent graphs as unitary matrices with a novel approach. Our technique can be applied to any graph, while state of the art encodings requires the input to have particular properties. Moreover, our encoding can be computed in polynomial time.

We extend the work done in [24] 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 [15,21]. In the general case of non-eulerian graphs, we describe an embedding which can be exploited in the quantum framework relying also on 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. However, projectors are not unitary operators and they break the sequence of unitary operations that usually represent quantum algorithms. Projectors can be interpreted as *part of a measurement* that is usually adopted to make the state collapse from quantum to classical.

The encoding with projectors is implemented in FREEQO[1] which allows us to discuss some experimental results. The focus is on the efficiency of the computation of the encoding. The obtained matrices are then transpiled to investigate on the scalability in terms of quantum circuits.

This work extends what was already introduced in [25]. In particular, we provide two solutions for avoiding projectors (see Section 7): the first one based on an Integer Linear Programming model used to balance the input graph with only edges that are already present. The second one solves the problem of the projectors by equipping the quantum walk with an additional set of qubits used to count the number of *illegal* traversed edges. Moreover, we implement the tool FREEQO and use it for the experimental evaluation.

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 is explained in Section 5. To show the differences between our method and the other ones in the literature, in Section 6 we compute and compare the encoding of a given graph using different techniques. In Section 7 we propose two different methods to avoid the use of projectors. Some experimental evaluations are presented in Section 8.

## 1. Preliminaries

### 1.1. Quantum computing

The most used model of Quantum Computation relies on the formalism of state vectors, unitary operators and projectors. Intuitively, 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 bits can have value 0 or 1 qubits are unitary vectors 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

[1] Available at https://github.com/RiccardoRomanello/FREEQO.

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 [26].

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 adjacency 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 transformations 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 information 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. Graphs

Graphs are a standard data structure in computer science for the representation of binary relations. We report below some standard definitions on graphs, while we refer the reader to [27] 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:

$$\delta^-(v) = \{i : i \in E, t(i) = v\}$$

and the set of its *outgoing* edges as:

$$\delta^+(v) = \{k : k \in E, s(k) = v\}$$

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 [15]. 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. 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.

The following result formalizes the equivalence between balanced and eulerian multigraphs [28].

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

In [24] 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:

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

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 [24] 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 (connected) multigraph and $\overrightarrow{G}$ be its line graph. Then $\overrightarrow{G}$ is the graph of a unitary matrix iff $G$ is eulerian.*

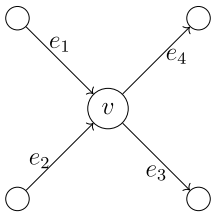
### *1.3. Quantum random walks*

In classical computation, when we visit a graph by randomly choosing 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.

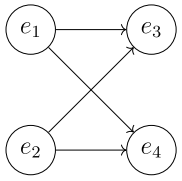
The visit, in both classical and quantum computation, can be made at either continuous or discrete time. In the case of continuous time QRWs, the Hamiltonian of the system is provided and the system evolves through time using matrix exponentiation [21,29]. On the other hand, in discrete time QRWs, the most common method is through *coined* walks [15]. 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 [24] lies at the heart of the transformation. However, the focus in [24] 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, it is linear in the size of the resulting matrix—we take into account the complexity of *producing* the resulting matrix and we show that the algorithm takes the same time. 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.




(a) A balanced node $v$ inside a graph $G$.




(b) The portion of $\overrightarrow{G}$ induced by $v$.

$$\begin{array}{c} \\ e_1 \\ e_2 \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & \cdots \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & \cdots \\ 0 & 0 & 1 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \end{array}$$

(c) Effect of $v$ on $\widetilde{M}$.

**Fig. 1.** How the neighborhood of a node inside $G = (V, E)$ is reflected in the matrix $\widetilde{M}$.

### 2.1. From a graph to its line

Let $G = (V, E)$ be an eulerian multigraph. The function LINEARIZE in Algorithm 1 returns the adjacency matrix $\widetilde{M}$ of the line graph $\overrightarrow{G}$. It can be easily checked that its time complexity is $\Theta(|E|^2)$.
Notice that, from this point, we will always refer to the set of nodes with $V$, to the set of edges with $E$, and to the input multigraph with $G$. Moreover, all the variables declared inside pseudo-code may be used inside statements or proofs.

**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
```

Intuitively, the algorithm works as follows. For every $i = (u, v) \in E$ we find all its consecutive edges, identified by $k$, and we set the entry $i, k$ to 1. We now point out some structural properties of the matrix $\widetilde{M}$ returned by Algorithm 1 that follow from the fact that $G$ is eulerian.

**Lemma 4.** *Let $i = (u, v)$ and $j = (u', v')$ be two edges of $G$. 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 5.** *For every edge $i = (u, v)$ of $G$, there are exactly $d^+(v)$ edges $j$ such that $\widetilde{M}_{i,j} \neq 0$. Moreover, there are exactly $d^-(v)$ rows equal to $\widetilde{M}_i$.*

**Example 1.** We give an example of how Lemma 4 and 5 work. Let $G = (V, E)$ be an input graph and let $v \in V$ be a node of $G$. In Fig. 1a we depicted $v$ with 2 incoming edges and two outgoing edges—$d^+(v) = d^-(v) = 2$. Fig. 1b shows how edges $e_1, e_2, e_3, e_4$ are connected in $\overrightarrow{G}$. By Lemmata 5 we know that there are exactly $d^+(v) = 2$ edges $j$ such that $\widetilde{M}_{e_1,j} \neq 0$. Since $\widetilde{M}$ is the adjacency matrix of $\overrightarrow{G}$, then we can conclude that: $\widetilde{M}_{e_1,e_3} = 1$ and $\widetilde{M}_{e_1,e_4} = 1$. By Lemmata 4, it holds that $\widetilde{M}_{e_1} = \widetilde{M}_{e_2}$ since $e_1$ and $e_2$ have the same target $v$.
The *submatrix* of $\widetilde{M}$ induced by $v$ is depicted in Fig. 1c. □

In Lemmata 4 and 5 we introduce a relationship between rows describing edges that have a common target. We can generalize the result considering both the case of edges incident to the same node and edges incident to different nodes.

**Lemma 6.** *Let $i, j$ be two edges of $G$. Let $NZ_i = \{k : \widetilde{M}_{i,k} \neq 0\}$ and similarly $NZ_j = \{k : \widetilde{M}_{j,k} \neq 0\}$. Then, either $NZ_i = NZ_j$ or $NZ_i \cap NZ_j = \emptyset$.*

**Algorithm 2** Compute a unitary matrix from the line graph.

```
1: function UNITARIZE(M̃, 𝒰)
2:     M̂ ← M̃
3:     Q ← V                                   ▷ V is the set of nodes of the input graph
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 $NZ_i = NZ_j$ there is nothing to prove. Suppose $NZ_i \neq NZ_j$ and assume, for the sake of contradiction, that $k = (v, w) \in NZ_i \cap NZ_j$. By definition of $NZ_i$ and $NZ_j$, it has to be $\widetilde{M}_{i,k} = 1 = \widetilde{M}_{j,k}$. Hence $i$ and $j$ share $v$ as common target. By Lemma 4, $\widetilde{M}_i = \widetilde{M}_j$ and $NZ_i = NZ_i$. This is a contradiction. □

In what follows we say that two rows $\widetilde{M}_i$ and $\widetilde{M}_j$ of $\widetilde{M}$—corresponding to the edges $i$ and $j$ of $G$—are *disjoint* if $NZ_i \cap NZ_j = \emptyset$, where $NZ_i, NZ_j$ 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 $\widetilde{M}$ (the matrix resulting from Algorithm 1) into a unitary matrix $\widehat{M}$ whose support is $\widetilde{M}$. 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 $G = (V, E)$. The input of the function is the adjacency matrix $\widetilde{M}$ and a family of unitary matrices $\mathcal{U}$. The family has to satisfy:

$$\forall v \in V \ \exists U \in \mathcal{U} \ \text{ of size } d^-(v) \times d^-(v) \tag{1}$$

Roughly speaking, let $V$ be the set of nodes of the input graph. We require that in $\mathcal{U}$ there is at least one matrix per each possible node in-degree. This because Algorithm 2 will use such matrices to turn $\widetilde{M}$ into a unitary matrix.

At each iteration of the while loop, a node $v$ is extracted from $Q$. For each of its incoming edges $i \in \delta^-(v)$ row $\widetilde{M}_i$ is edited through the procedure SPARSESUB exploiting a unitary matrix $U$ of suitable dimension. In SPARSESUB$(r, r')$ the notation $r_k, r'_h$ refers to the $k$-th element of $r$ and the $h$-th element of $r'$, respectively. In particular, SPARSESUB$(r, r')$ replaces the $k$-th element of $r$ with $r'_h$.

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

**Lemma 7.** *At each iteration of the for loop at line 8 of Algorithm 2 it holds that after applying* SPARSESUB$(\widehat{M}_i(\delta^+(v)), U_c)$*, the subvector* $\widehat{M}_i(\delta^+(v))$ *is equal to* $U_c$.

**Proof.** From Lemmata 4 and 5 it follows that $\widetilde{M}_{i,k} = r_k \neq 0$ (line 17) holds iff $k \in \delta^+(v)$. By construction, $U_c$ has size $n = d^-(v) = d^+(v) = |\delta^+(v)|$. The result follows immediately. □

**Theorem 8.** $\widehat{M}$ *is a unitary matrix.*

**Proof.** It is sufficient to show that the rows of $\widehat{M}$ form an orthonormal basis. By Lemma 7, 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 2.** We now provide an example to clarify the structure of $\widehat{M}$ after the procedure UNITARIZE. For sake of readability, we omit the scalar multipliers of the matrices of $\mathcal{U}$ inside $\widehat{M}$.

The reader may notice that the choice of $\mathcal{U}$ is not unique. Any set that satisfies (1) is a candidate.

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

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

The reason for this particular family $\mathcal{U}$ is to show that there can be more than one unitary matrix with the same size inside such family—In this case we have two different $2 \times 2$ unitaries. Such particular property results in the fact that $\widehat{M}$ is not unique. For example, any one of the following can be obtained through Algorithm 2:

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

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

## 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—such multigraph will be given as input to Algorithm 1. We recall that, since we are interested in quantum random walks, we take into account only 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
```

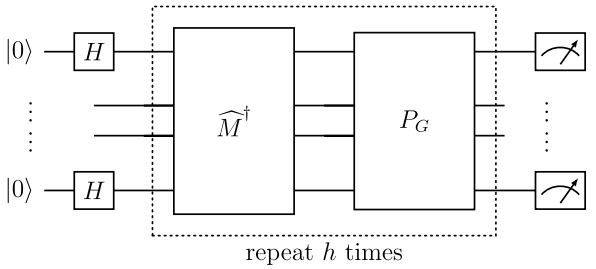



**Fig. 2.** Quantum circuit for the walk.

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 $\Theta(|V|+|E_\perp|) \subseteq O(|V|+|E|)$, since we never add more edges than the existing ones. This is the technical point where the use of multigraphs helps us: copies of existing edges, as well as completely new edges, may be added by the procedure.

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 $G$ 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 $G'$. Because of that, edges from $E_\perp$ could be traversed during the walk, while we only want edges from $E$ to be used to visit the graph. 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 3.** The following toy example shows 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

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 Fig. 2.

In particular, in the circuit in Fig. 2 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, in Section 7, we propose and discuss two different methods to get rid of the projectors.

## 5. Edges/nodes failures

The overall procedure described in the previous sections has a time computational complexity $\Theta(|E|^2)$ 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., [21]). 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

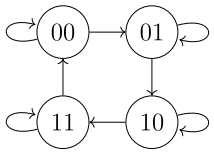



**Fig. 3.** Four nodes cyclic graph.

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 Fig. 2) 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 recognized 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 $\Theta(d^+(v) + d^-(v))$.

So, both node and edge failures can be handled in constant time. Even if our procedure has a running time for computing the graph encoding that may be worse than other proposals, we do not need to recompute the unitary matrix in case of dynamic changes.

## 6. Comparisons against existing methods

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 [21] 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 [15] 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.

### 6.1. Continuous time walk from [21]

Let $G$ be the cyclic graph depicted in Fig. 3 and $M$ be its adjacency matrix. Let $\pi = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$ be the unique non-negative eigenvector of $M$, $\Pi = \text{DIAG}(\pi)$ 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 $\Pi$ and $L$ need to be recomputed. However, in [21] the focus was not on developing visit algorithms but on defining a measure of similarity between graphs.

### 6.2. Discrete time walk from [15]

We consider again the graph of Fig. 3. 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 labeled – 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 labeling.

### 6.3. Our method

We now want to obtain a unitary matrix for $G$ in Fig. 3 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($\widetilde{M}, \mathcal{U}$) to obtain $\widehat{M}$

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

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

$$|000\rangle = (00,00) \qquad |001\rangle = (00,01) \qquad |010\rangle = (01,01) \qquad |011\rangle = (01,10)$$
$$|100\rangle = (10,10) \qquad |101\rangle = (10,11) \qquad |110\rangle = (11,11) \qquad |111\rangle = (11,00)$$

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

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

(a) A cyclic state evolution. (b) An infinite line state evolution.

**Fig. 4.** The two possible evolutions for a Quantum State.

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

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 $G_U = (V_U, E_U)$:

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

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 exist, then $|\psi\rangle$ will be a part of an infinite line of states, see Fig. 4.

## 7. Handling the projectors

In the proposed method, we encode any kind of graph into a unitary matrix. In doing so, we have been forced to edit the initial graph and this eventually led to the introduction of the projector $P_G$. While the projector is a key component to achieve constant time editing in case of edge-failure, it is a non-unitary operation. In this section we will discuss some options to get rid of $P_G$.

### 7.1. Directed Chinese postman problem

By Theorem 2, in order to obtain an eulerian graph it is sufficient to *balance* each node. In Algorithm 3 we did this by adding edges from nodes with a negative balance to nodes with a positive one. Even though the algorithm adds the minimum number of edges, it can add edges that were not in the initial graph and that must be projected during the walk. To avoid the use of projectors, a solution may be to only add copies of existing edges: this is a technique used to solve the Directed Chinese Postman Problem (DCPP) [30,31].

In the DCPP, a postal carrier must pick up the mail at the post office, deliver them along blocks on the route and finally return to the post office. To make the job easier and more productive, every postal carrier would like to cover the route with as little traveling as possible.

Formally, we have a directed multigraph $G = (V, E)$ where each node is an intersection and each edge represents a street. A solution to the problem is a minimum-length cyclic tour that traverses each edge at least once and starts at $v_0$ (the postal office). If $G$ admits an eulerian cycle, then it is a solution to the DCPP. Otherwise, some edges must be traversed more than once or, equivalently, they must be copied so that the resulting multigraph $G^*$ is eulerian. It has been proved that DCPP admits a solution if and only if $G$ is strongly connected [32]. This constraints the graphs we can encode. Nevertheless, it is a reasonable assumption that enables us to remove the (costly) projectors from the circuit.

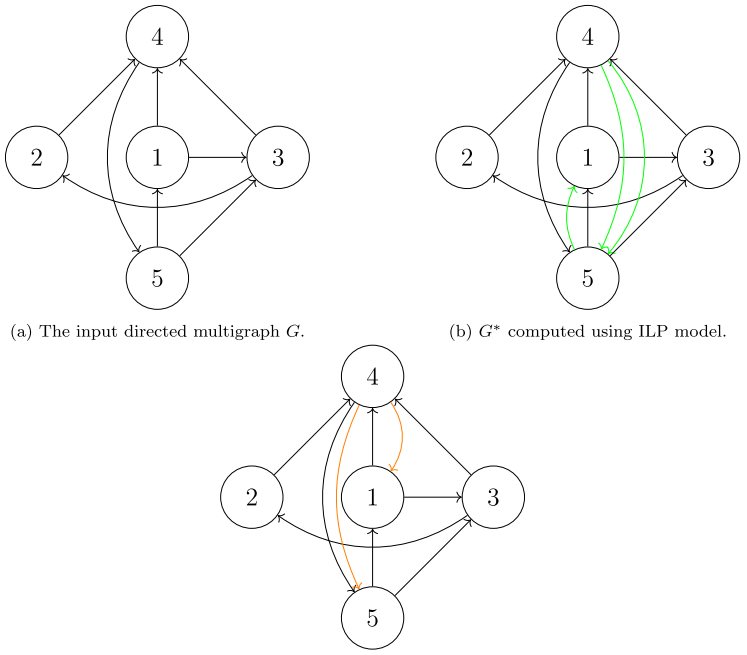
The problem is tackled via Integer Linear Programming. The optimization model (Fig. 5) minimizes the number of copies $x_e$, for each edge $e \in E$, that are required to make $G$ balanced. The program models a minimum cost flow problem, which is solvable in polynomial time [33].

Given an optimal solution $x^*$, the graph $G^*$ is obtained from $G$ by adding $x^*_e$ copies of edge $e$, for all $e \in E$. This new graph is balanced by construction, hence it admits an eulerian circuit.

**Example 4.** In Fig. 6a, we depicted a graph $G$ where nodes 1, 4, 5 are not balanced. Since the graph is strongly connected, the model admits a solution from which $G^*$ is obtained, as shown in Fig. 6b. On the other hand, using Algorithm 3, we would obtain a different graph $G'$, which is depicted in Fig. 6c. The main difference between $G^*$ and $G'$ is that in the former only copies of existing edges have

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \sum_{e \in \delta^-(v)} x_e - \sum_{e \in \delta^+(v)} x_e = b(v), && \forall v \in V \\ & x_e \in \mathbb{N}, && \forall e \in E \end{aligned}$$

**Fig. 5.** The ILP model for balancing the graph.




(a) The input directed multigraph $G$.

(b) $G^*$ computed using ILP model.

(c) $G'$ from Algorithm 3.

**Fig. 6.** Multigraphs of Example 4.

been added, while in the latter we added the edge (4, 1) which was not present in $G$. Notice that in the execution of Algorithm 3 we considered nodes in $B^+$ and $B^-$ sorted by label. □

By giving $G^*$ as input to Algorithm 1, we obtain a matrix that can be used without projectors during the QRW. Nevertheless, projectors are mandatory in case the edge/node failure as in Section 5. In the next subsection we take care of this issue.

### 7.2. Counters

We introduced projectors to avoid traversing edges that were not in the initial graph during the QRW. Since we use the quantum circuit formalism, the length $k$ of the walk must be finite and fixed before compiling the circuit. Therefore, we know that all tours that will be considered during the computation have length exactly $k$. A tour that never traverses edges in $E_\perp$ is called *legal*, otherwise it is called *illegal*: only legal walks must contribute to a measurement.

In order to do so, we will augment the state of the walk with $m = \lceil \log(k+1) \rceil$ additional qubits to *count* the number of edges in $E_\perp$ that have been traversed by a particular tour in the multigraph. If this counter is nonzero, the walk is illegal.

Given an $m$-qubit register, the unitary matrix that increments its value by 1 is a $2^m \times 2^m$ permutation that sends each element to its "successor":

$$\begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & & \cdots & 1 & 0 \end{pmatrix} \tag{2}$$

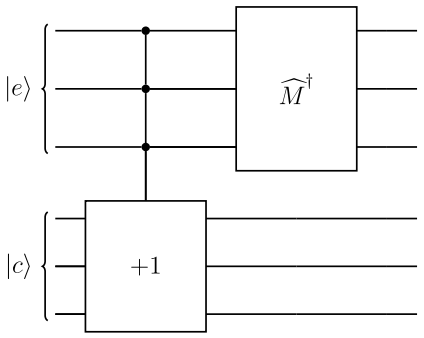



**Fig. 7.** Step of the walk with counters.

The augmented state will evolve as follows:

$$\begin{cases} |\psi_0\rangle & = |e\rangle\,|0\rangle \\ |\psi_{t+1}\rangle & = (\widehat{M}\otimes I)\cdot C_m\,|\psi_t\rangle \end{cases}$$

where $C_m$ is a matrix defined as:

$$C_m\,|e\rangle\,|c\rangle = \begin{cases} |e\rangle\,|c\rangle & \text{if } e\in E, \\ |e\rangle\,|(c+1) \mod 2^m\rangle & \text{if } e\in E_\perp \end{cases} \tag{3}$$

Roughly speaking, $C_m$ increments the counter whenever an edge in $E_\perp$ is traversed and does nothing otherwise. It can be proved that $C_m$ is unitary—blocks acting on $e\in E$ are identities and blocks acting on $e\in E_\perp$ are of the form of (2).

**Example 5.** Let $E=\{e_0,e_2\}$ and $E_\perp=\{e_1\}$ be the two set of edges. Consider the case of a QRW of length $k=1$. Thus, $m=1$ and the counter takes value $c\in\{0,1\}$. Basis states $|e_i\rangle\,|c\rangle$ are column vectors with a single 1 in position $2\cdot i+c$, e.g. $|e_0\rangle\,|1\rangle=(0\,1\,0\,0\,0\,0)^\dagger$ and $|e_2\rangle\,|0\rangle=(0\,0\,0\,0\,1\,0)^\dagger$.

Consider the augmented states sorted by this encoding. In our example, the matrix $C_1$ has the following form:

$$C_1=\begin{pmatrix} I_2 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & X & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I_2 \end{pmatrix}=\begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&0&1&0&0\\ 0&0&1&0&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix}$$

Notice that $C_1$ is a block diagonal matrix. The top-left and bottom-right $2\times 2$ identity matrices handle the states of the form $|e_0\rangle\,|\cdot\rangle$ and $|e_2\rangle\,|\cdot\rangle$. The $2\times 2$ center block is the permutation matrix acting on states $|e_1\rangle\,|\cdot\rangle$ as an increment modulo 2. □

By generalization of Example 5 and by assuming the "order by encoding" of the augmented states, one can prove that $C_m$ is a block diagonal matrix whose blocks have size $2^m\times 2^m$ and are either the identity or the increment modulo $2^m$.

Notice that, with $m\geq 1$ qubits, it is possible to recognize a legal or illegal walk of length up to $2^m-1$.

Following Equation (3), after applying $C_m$ the state is evolved by applying $\widehat{M}$ on the edge-qubits and by leaving the counter-qubits unchanged. Fig. 7 depicts the circuit implementing a single step of the visit.

Let $|\psi_k\rangle$ be the last state of the walk and suppose we want to measure the probability of being in a subset $E'\subseteq E$ of a legal walk. The projector to use is the following:

$$P=\sum_{e\in E'}|e,0\rangle\,\langle e,0|$$

Consider the failure of an edge $\overline{e}\in E$. We can think of $\overline{e}$ as an illegal edge belonging to $E_\perp$. Therefore, the matrix $C_m$ needs to be edited to reflect this event—identity block for $\overline{e}$ must be turned into the increment modulo $2^m$. Exactly $2\cdot 2^m\leq 4k$ bit-flips are sufficient.

The failure of a node can be handled as the failure of all its incoming and outgoing edges.

## 8. Experimental evaluation

The aim of this section is to experimentally investigate on the effectiveness of our encoding described in Section 2 and Section 3.

The overall procedure we want to test takes a multigraph $G=(V,E)$ and a family of unitary matrices $\mathcal{U}$ as input and performs the following operations:

**Table 1**
Experimental results: averages are over 10 random graphs per each pair $(n, p)$.

| $n$ | $p$ | AVG. SIZE | AVG. NEW EDGES | AVG. NNZ | AVG. TIME (S) |
|---|---|---|---|---|---|
| 10 | 0.3 | 34 | 8 | 133 | $< 0.01$ |
| 10 | 0.5 | 55 | 8 | 311 | $< 0.01$ |
| 10 | 0.7 | 69 | 7 | 487 | $< 0.02$ |
| 25 | 0.3 | 215 | 32 | 1945 | 0.119 |
| 25 | 0.5 | 336 | 34 | 4633 | 0.250 |
| 25 | 0.7 | 448 | 31 | 8104 | 0.429 |
| 50 | 0.3 | 820 | 92 | 13831 | 1.186 |
| 50 | 0.5 | 1310 | 98 | 34688 | 3.029 |
| 50 | 0.7 | 1805 | 90 | 65457 | 5.908 |
| 100 | 0.3 | 3212 | 254 | 104720 | 17.699 |
| 100 | 0.5 | 5219 | 298 | 274053 | 51.520 |
| 100 | 0.7 | 7188 | 254 | 518098 | 96.914 |

1. Apply Algorithm 3 on $G$. This step returns the set $E_\perp$.
2. Apply Algorithm 1 obtaining the matrix $\widetilde{M}$.
3. Use $\widetilde{M}$ and $\mathcal{U}$ as input for Algorithm 2.

The total time complexity of the procedure is $O(|E|^2)$.
We will measure the following quantities and we will give some related statistics:

- The running time of the procedure,
- The number $|E_\perp|$ of edges added through the balancing procedure. This will give an insight on how distant is a random graph to become eulerian,
- The number of nonzero elements in the final matrix,
- The size of the transpiled circuit.

### 8.1. Random graphs and test cases generation

Test cases are generated randomly. We adopted the Erdös-Renyi-Gilbert model [34] to generate random graphs of different sizes and topologies. An Erdös-Renyi-Gilbert graph $G(n, p)$ is a directed graph with $n$ nodes where each edge $(u, v)$ has probability $p$ to appear in $G$.

The generation of tests is done in `C++` using the publicly available library `boost`.[2]

We generated a sample of 120 random graphs: 10 for each pair $(n, p)$ with $n = 10, 25, 50, 100$ and $p = 0.3, 0.5, 0.7$, see Tables 2, 3, 4.

We decided to focus our attention on equiprobable QRWs—when sitting on a node $u$, every outgoing edge has probability $\frac{1}{d^+(u)}$ of being traversed. To ensure this, we adopted the family of unitaries $\mathcal{U} = \{\mathrm{DFT}(n) : n \in \mathbb{N}\}$, where:

$$\mathrm{DFT}(n) = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(n-1)} \\ \vdots & & & & \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{pmatrix}$$

and $\omega = e^{\frac{2\pi i}{n}}$.

### 8.2. Results

Results are presented in Table 1. Columns $n$ and $p$ are defined as in Section 8.1. Given $n$ and $p$, the columns AVG. SIZE, AVG. NEW EDGES, AVG. NNZ, and AVG. TIME contain, respectively, the mean of the following quantities: number of edges of the graph after balancing (number of rows of the encoding matrix), number of edges added (size of $E_\perp$), number of nonzero entries in the constructed unitary matrix, and running time for mapping a graph into its associated unitary matrix.

We include the nonzero entries column since it provides a good explanation of the average time results. In fact, Algorithm 3 only edits elements that are not zero.

The obtained results are encouraging on the effectiveness of the proposed method. For instance, in graphs of size 100 and density 0.7—which have roughly 7000 edges—the average number of added edges is around 250 which is 3.5% of the initial number of edges. In the worst case the method could double the number of edges of the input graph but, the examples show that in the average we are far from that scenario, as it emerges looking at the average number of new edges in Table 1. As a consequence also the average number of nonzero entries of the unitary matrix and the average time are far from the worst case.

[2] The documentation is available at https://www.boost.org/doc/libs/1_81_0/libs/graph/doc/erdos_renyi_generator.html.

**Table 2**
Standard deviation of number of added edges and running time with $p = 0.3$.

| | $p = 0.3$ | |
|---|---|---|
| $n$ | STD-DEV NEW EDGES | STD-DEV TIME (S) |
| 10 | 2 | $< 0.001$ |
| 25 | 31 | $< 0.001$ |
| 50 | 64 | 0.04 |
| 100 | 439 | 0.96 |

**Table 3**
Standard deviation of number of added edges and running time with $p = 0.5$.

| | $p = 0.5$ | |
|---|---|---|
| $n$ | STD-DEV NEW EDGES | STD-DEV TIME (S) |
| 10 | 3 | $< 0.001$ |
| 25 | 32 | $< 0.001$ |
| 50 | 215 | 0.12 |
| 100 | 334 | 3.74 |

**Table 4**
Standard deviation of number of added edges and running time with $p = 0.7$.

| | $p = 0.7$ | |
|---|---|---|
| $n$ | STD-DEV NEW EDGES | STD-DEV TIME (S) |
| 10 | 1 | $< 0.001$ |
| 25 | 25 | $< 0.001$ |
| 50 | 126 | 0.060 |
| 100 | 721 | 16.18 |

On the one hand, other *coin-based* encodings have been experimentally tested with the aim of analyzing the *convergence* of the walk (e.g., mixing time and hitting lime). The time required for the generation of the data structure has never been reported. In our case, since edge additions could make the size of the matrix grow critically, we were interested in testing the impact of such event.

On the other hand, if we work in an applicative domain where edge failures in the underlying graph are an issue (see, e.g., [35]), then our approach allows to get an efficient implementation whereas other methods in the literature have to recompute the whole encoding. Therefore it was important for us to show that the initial computation of the matrix is not a bottleneck for the procedure.

Once the matrix has been computed for all the above mentioned cases, it can be *transpiled* in a quantum circuit. Because of physical limitations in state of the art quantum computers, only a small set of *basic* unitaries can be physically applied to qubits. Transpilation is the act of expressing non-basic matrices in terms of basic ones.

In our experiments, we used Qiskit—a python library for creating, simulating and running quantum circuits—to perform the transpilation. The basic gates we adopted are $\{u_1, u_2, u_3, cx\}$, defined as:

$$u_1(\lambda) = U(0,0,\lambda) \qquad cx = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$
$$u_2(\phi,\lambda) = U(\pi/2,\phi,\lambda)$$
$$u_3(\theta,\phi,\lambda) = U(\theta,\phi,\lambda)$$

where:

$$U(\theta,\phi,\lambda) = \begin{pmatrix} \cos\frac{\theta}{2} & -e^{i\lambda}\sin\frac{\theta}{2} \\ e^{i\phi}\sin\frac{\theta}{2} & e^{(\phi+\lambda)}\cos\frac{\theta}{2} \end{pmatrix}$$

The results we obtained can be found in Table 5 and 6. Tables present the following data relating the input matrix and the output (transpiled) matrix:

1. The average size (in terms of number of rows) of the input,
2. The average number of non-zero elements inside the input,
3. The average transpilation time,
4. The average number of basic gates used.

**Table 5**
Results obtained for graphs with 10 nodes.

| $p$ | AVG. SIZE | AVG. NNZ | AVG. TRANSP. TIME (S) | AVG. $u_1$ | AVG. $cx$ | AVG. $u_2$ | AVG. $u_3$ |
|---|---|---|---|---|---|---|---|
| 0.3 | 64 | 1390 | 25.56 | 49 | 2956 | 25 | 3088 |
| 0.5 | 64 | 2620 | 30.34 | 54 | 3666 | 21 | 3834 |
| 0.7 | 128 | 4550 | 111.50 | 102 | 13910 | 24 | 14331 |

**Table 6**
Results obtained for graphs with 25 nodes.

| $p$ | AVG. SIZE | AVG. NNZ | AVG. TRANSP. TIME (S) | AVG. $u_1$ | AVG. $cx$ | AVG. $u_2$ | AVG. $u_3$ |
|---|---|---|---|---|---|---|---|
| 0.3 | 256 | 13658 | 504.14 | 187 | 49484 | 52 | 50414 |
| 0.5 | 512 | 31256 | 2063.03 | 359 | 181142 | 56 | 183314 |
| 0.7 | 512 | 58996 | 3083.59 | 462 | 271472 | 74 | 274395 |

Observing the data, the transpilation time becomes really high as soon as the inputs reach the order of $512 \times 512$ elements (graphs with $n = 25$ nodes and probability $p = 0.5$). On the other hand, the number of gates to use increases already while tackling the problem with 10 nodes and $p = 0.7$.

## 9. 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 this paper we introduced a method to encode any graph into a unitary matrix. The core idea of the algorithm is the embedding of the input graph into an *eulerian* multigraph by adding new edges. This new technique produces a matrix that has a closer relationship to the initial graph topology than other methods in literature (e.g., [21], [15]). Furthermore, we use projectors to efficiently support the addition/deletion of edges and thus react to sudden link-failures in the network.

After comparing our encoding to state of the art techniques, we proposed two alternative methods to avoid the use of (costly) projectors. In the first one, we used an Integer Linear Program based on the Directed Chinese Postman Problem to embed the graph into an eulerian one. This method balances the input by adding copies of pre-existing edges but loses the capability of handling edge failures. In the second one, we add a counter to distinguish legal walks—those that do not traverse newly added edges—from illegal ones. In this case we also proved that, with a constant number of operations, we can still handle edge failures.

Finally, we implemented our encoding and we experimentally tested the algorithms on random generated graphs. The data we collected show the effectiveness of our method. Furthermore, we started to investigate on the complexity of circuit-transpilation: experimental results showed that this phase becomes intractable for graphs with more than 25 nodes. However, the transpiled matrices exhibit some regularities that we want to exploit with graph reduction techniques such as lumpabilities [36–38].

As future work we are interested in studying the limiting distributions of our QRWs. For a restricted class of graphs we are able to obtain results comparable to [15] and simulations suggest that the class can be extended.

## Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Carla Piazza reports financial support was provided by Government of Italy Ministry of Education, University and Research. Grant number is 20202FCJM CUP G23C22000400005. Carla Piazza reports financial support was provided by Francesco Severi National Institute of Higher Mathematics. Grant number is CUP E53C22001930001.

## References

[1] D. Deutsch, R. Jozsa, Rapid solution of problems by quantum computation, Proc. R. Soc. Lond. Ser. A, Math. Phys. Sci. 439 (1992) 553–558, https://doi.org/10.1098/rspa.1992.0167.
[2] D. Qiu, S. Zheng, Revisiting Deutsch-Jozsa algorithm, Inf. Comput. 275 (2020) 104605, https://doi.org/10.1016/j.ic.2020.104605.
[3] A. Ambainis, L.J. Schulman, U.V. Vazirani, Computing with highly mixed states, J. ACM 53 (3) (2006) 507–531, https://doi.org/10.1145/1147954.1147962.
[4] R. Jozsa, N. Linden, On the role of entanglement in quantum-computational speed-up, Proc. R. Soc. Lond., Ser. A, Math. Phys. Eng. Sci. 459 (2036) (2003) 2011–2032, https://doi.org/10.1098/rspa.2002.1097.
[5] E. Biham, G. Brassard, D. Kenigsberg, T. Mor, Quantum computing without entanglement, Theor. Comput. Sci. 320 (1) (2004) 15–33, https://doi.org/10.1016/j.tcs.2004.03.041.
[6] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, 3rd edition, Pearson international edition, Addison-Wesley, 2007.
[7] C. Piazza, R. Romanello, Mirrors and memory in quantum automata, in: E. Ábrahám, M. Paolieri (Eds.), Quantitative Evaluation of Systems - 19th International Conference, QEST 2022, Proceedings, in: Lecture Notes in Computer Science, vol. 13479, Springer, 2022, pp. 359–380.
[8] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.

[9] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows - Theory, Algorithms and Applications, Prentice Hall, 1993.
[10] Z. Manna, A. Pnueli, Temporal Verification of Reactive Systems - Safety, Springer, 1995.
[11] L. Anticoli, C. Piazza, L. Taglialegne, P. Zuliani, Towards quantum programs verification: from quipper circuits to QPMC, in: S.J. Devitt, I. Lanese (Eds.), Reversible Computation - 8th International Conference, RC 2016, Proceedings, in: Lecture Notes in Computer Science, vol. 9720, Springer, 2016, pp. 213–219.
[12] A.S. Tanenbaum, D. Wetherall, Computer Networks, 5th edition, Pearson, 2011.
[13] S. Haykin, Neural Networks and Learning Machines, 3/E, Pearson Education India, 2010.
[14] D.A. Easley, J.M. Kleinberg, Networks, Crowds, and Markets - Reasoning About a Highly Connected World, Cambridge University Press, 2010.
[15] D. Aharonov, A. Ambainis, J. Kempe, U.V. Vazirani, Quantum walks on graphs, in: J.S. Vitter, P.G. Spirakis, M. Yannakakis (Eds.), Proceedings on 33rd Annual ACM Symposium on Theory of Computing, ACM, 2001, pp. 50–59.
[16] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, 1995.
[17] E. Çinlar, Probability and Stochastics, Springer, 2011.
[18] S.E. Venegas-Andraca, Quantum walks: a comprehensive review, Quantum Inf. Process. 11 (5) (2012) 1015–1106, https://doi.org/10.1007/s11128-012-0432-5.
[19] E. Farhi, S. Gutmann, Quantum computation and decision trees, Phys. Rev. A 58 (2) (1998) 915–928, https://doi.org/10.1103/physreva.58.915.
[20] F. Chung, Laplacians and the Cheeger inequality for directed graphs, Ann. Comb. 9 (1) (2005) 1–19, https://doi.org/10.1007/s00026-005-0237-z.
[21] G. Minello, L. Rossi, A. Torsello, Can a quantum walk tell which is which? A study of quantum walk-based graph similarity, Entropy 21 (3) (2019) 328, https://doi.org/10.3390/e21030328.
[22] G.D. Paparo, M. Martin-Delgado, Google in a quantum network, Sci. Rep. 2 (1) (2012) 1–12, https://doi.org/10.1038/srep00444.
[23] F. Xia, J. Liu, H. Nie, Y. Fu, L. Wan, X. Kong, Random walks: a review of algorithms and applications, IEEE Trans. Emerg. Top. Comput. Intell. 4 (2) (2020) 95–107, https://doi.org/10.1109/TETCI.2019.2952908.
[24] S. Severini, On the digraph of a unitary matrix, SIAM J. Matrix Anal. Appl. 25 (1) (2003) 295–300, https://doi.org/10.1137/S0895479802410293.
[25] D.D. Giustina, C. Piazza, B. Riccardi, R. Romanello, Directed graph encoding in quantum computing supporting edge-failures, in: C.A. Mezzina, K. Podlaski (Eds.), Reversible Computation - 14th International Conference, RC 2022, Proceedings, in: Lecture Notes in Computer Science, vol. 13354, Springer, 2022, pp. 75–92.
[26] M.A. Nielsen, I.L. Chuang, Quantum Computation and Quantum Information, 10th anniversary edition, Cambridge University Press, 2016.
[27] F. Harary, Graph Theory, Addison-Wesley, 1991.
[28] M. Cygan, D. Marx, M. Pilipczuk, M. Pilipczuk, I. Schlotter, Parameterized complexity of Eulerian deletion problems, Algorithmica 68 (1) (2014) 41–61, https://doi.org/10.1007/s00453-012-9667-x.
[29] A.K. Madhu, A.A. Melnikov, L.E. Fedichkin, A.P. Alodjants, R.-K. Lee, Quantum walk processes in quantum devices, Heliyon 9 (3) (2023) e13416, https://doi.org/10.1016/j.heliyon.2023.e13416.
[30] J. Edmonds, Chinese postmans problem, in: Operations Research, 1965, p. B73.
[31] M. Guan, Graphic programming using odd and even points, Chin. Math. 1 (1962) 237–277.
[32] H.A. Eiselt, M. Gendreau, G. Laporte, Arc routing problems, part I: the Chinese postman problem, Oper. Res. 43 (2) (1995) 231–242, https://doi.org/10.1287/opre.43.2.231.
[33] J. Edmonds, E.L. Johnson, Matching, Euler tours and the Chinese postman, Math. Program. 5 (1) (1973) 88–124, https://doi.org/10.1007/BF01580113.
[34] E.N. Gilbert, Random Graphs, Ann. Math. Stat. 30 (4) (1959) 1141–1144, https://doi.org/10.1214/aoms/1177706098.
[35] S. Liu, D. Zhang, X. Liu, T. Zhang, J. Gao, C. Gong, Y. Cui, Dynamic analysis for the average shortest path length of mobile ad hoc networks under random failure scenarios, IEEE Access 7 (2019) 21343–21358, https://doi.org/10.1109/ACCESS.2019.2896699.
[36] G. Alzetta, A. Marin, C. Piazza, S. Rossi, Lumping-based equivalences in Markovian automata: algorithms and applications to product-form analyses, Inf. Comput. 260 (2018) 99–125, https://doi.org/10.1016/j.ic.2018.04.002.
[37] A. Marin, C. Piazza, S. Rossi, Proportional lumpability and proportional bisimilarity, Acta Inform. 59 (2) (2022) 211–244, https://doi.org/10.1007/s00236-021-00404-y.
[38] D. Ressi, R. Romanello, C. Piazza, S. Rossi, Neural networks reduction via lumping, in: A. Dovier, A. Montanari, A. Orlandini (Eds.), AIxIA 2022 - Advances in Artificial Intelligence - XXIst International Conference of the Italian Association for Artificial Intelligence, Proceedings, in: Lecture Notes in Computer Science, vol. 13796, Springer, 2022, pp. 75–90.