Planning Qubit Movements in Fully Connected Star Graphs

by Hans Keur
Student number: 4223292
Submitted to the department of EEMCS

in partial fulfilment of the requirements for the degree of
Master of Science in Electrical Engineering
with specialisation in Signals and Systems
at

DELFT UNIVERSITY OF TECHNOLOGY

Supervised by prof. dr. S.D.C. Wehner

August 28, 2017

Faculty of Electrical Engineering,
Mathematics and Computer Science
Delft University of Technology
Delft, The Netherlands

QuTech
Delft University of Technology
Building 22, Faculty of Applied Sciences
Lorentzweg 1
Delft, The Netherlands

Abstract

Many scientists and engineers in the field of quantum computing and engineering are collaborating in order to realize a quantum computer. A quantum computer uses qubits to do calculations. These are clustered in processing units, which are interconnected by optical links. This might be implemented as qubits in diamonds with optical fibres between the diamonds.

Within a processing unit, the qubits can be modelled as nodes in a star graph. The optical connections between different diamonds can be modelled as connections between the centres of the star graphs, such that the centres with their interconnections form a complete graph. We call the overall graph a fully connected star graph. This topology is derived from the interactions possible in a quantum processor based on NV-centres. Each NV-centre is assumed to have multiple memory qubits around it, forming a processing unit.

Nowadays, a quantum processing unit does not comprise enough qubits to do interesting calculations. This is why there are interconnections between the processing units; by these connections, they can perform a large task together. Such a task is described by a quantum circuit. It describes (among others) in which order pairs of two qubits should be together in a processing unit. For example, if we have four qubits numbered from 1 - 4, it might be that firstly, an interaction between qubits 1 and 2 is desired and after that one between qubits 1 and 3. It should be ensured that in the first place, qubits 1 and 2 are in the same processing unit and when the subtask is performed with these qubits, then qubits 1 and 3 should be brought together to do the next subtask. Or to give a more practical example, a wall of a house is built by firstly, ensuring that there is a basement which is strong enough, secondly, the first row of stones is placed and then the higher stones will be placed. Similarly, in a quantum computer, firstly calculation A should be done, then calculation B, such that after X calculations the result Y is obtained.

In order to execute a quantum circuit, multiple pairs of qubits should be brought together in different quantum processing units, such that multiple tiny calculations can be done in parallel. In order to bring two qubits together and to distribute the pairs of qubits among the available processing units, planning (sorting) is required.

To bring pairs of qubits together, the qubits can be moved around in the graph by means of swaps. A SWAP operation within a processing unit (PU) implies that the qubit in the centre of the PU is interchanged with another qubit in the PU, which is said to be in a

leaf. A SWAP operation between different PUs is possible too, by means of a teleportation procedure. This procedure is considered much more costly than a swap within a PU.

Several swaps between PUs can be done in parallel. This is called a time step, or shortly a step. The less steps are used, the faster the quantum circuit is executed. In practice, as we are interested in fast computations, we often want to minimize the number of steps. However, minimizing the execution time results in more costs of swaps within processors, which means that more energy (electrical power) is required.

Going back to our problem, the aim is to minimize the costs of the routing process of the qubits. These costs are described in terms of the number of swaps and the number of steps. At the moment, no efficient algorithm exists to sort the qubits especially in fully connected star graphs. However, this work demonstrates efficient solutions in the form of algorithms to several slightly different versions of the *routing via matching* model introduced by Alon et al [1], which is applied to fully connected star graphs. Thereby, both minimizing the number of swaps and minimizing the number of steps is considered. The proposed algorithms minimize either one of these factors or both.

It is assumed that the costs for exchanging two qubits between centres are much higher than swapping qubits between a centre and a leaf. This is motivated by fact that the teleportation process between centres takes much more time than executing some gate within a processing unit. Even stronger, the execution time of gates within the processors are neglected.

It is shown that qubit moves in fully connected star graphs can be scheduled efficiently, such that the number of moves is minimized. Guidelines are given to minimize the execution time too, where only interactions between PUs are counted.

Contents

List of Figures

List of Tables

1 Acknowledgements

Before others, my thanks are directed to the everlasting God of our universe, by Whose providence I have been able to accomplish this work. That His blessings may rest on it!

I want to thank my supervisor Stephanie Wehner for her good supervision. It was a pleasure to be supervised by her and to learn from her. I thank her for giving encouragements from time to time, to improve certain skills. I appreciate her valuable comments on my work.

My thanks go to David Elkouss for showing his interests in my research projects and for attending my thesis committee.

I thank Carmina Almudéver for her willingness to attend my thesis committee. I thank her and many others for their valuable recommendations to improve my work.

From the first days I worked in the group of theorists at QuTech, I highly valued the good atmosphere within it. It was a pleasure to observe the respect shown for each other. I thank all group members and especially my office mates, for their willingness to help me solving issues and for their encouragements to improve my skills.

Als laatste wil ik in het bijzonder mijn ouders bedanken voor jullie meeleven bij het volbrengen van dit werk. Dat dat en dit werk gezegend mag worden!

2 Introduction

At least in China [2] and the Netherlands [3], quantum scientists and engineers are working on the realisation of quantum networks and quantum computers. These devices use qubits to do calculations. One kind of qubit candidate is the Nitrogen Vacancy (NV) centre in diamond [4]. Its atomic structure is such that the NV-centre can act as a communication qubit and several surrounding carbon-13 atoms as memory qubits. The communication between the NV-centres is done optically, by means of laser pulses via optical links [4]. The interconnections between multiple NV-centres can be implemented with glass fibres, such that a cluster of small quantum processors can be formed.

Quantum computers are being built because of their special properties, like the ability to provide theoretically, perfectly secure communications [5] and data storage [6]. Secondly, many scientists and engineers believe that the quantum computer would be able to perform many computations much faster than any classical computers. However, it is still one of the open questions in this field if that is true or not. In many cases, this is equivalent to the question whether the computational complexity classes BQP and P are equal. Assuming that the classes BQP and P are not equal or that the polynomial hierarchy does not collapse, many scientists would like to show *quantum supremacy* (see e.g. [7]), by showing the successful execution of certain quantum circuits [7] [8]. Ultimately, this cannot be done without a properly working quantum computer.

2.1 Problem Examples

One of the problems in the field of quantum computation is the problem of planning the execution of a quantum circuit given specific quantum hardware. This is known as the *circuit placement problem*, which is known to be NP-complete, in general [9]. One of the goals is to minimize the circuit execution time. Another goal might be to minimize the number of quantum gates used to perform the circuit. In this work, closely related problems will be faced, which will be explained in more detail below. Firstly, we consider the topology of the quantum processor and after that, the planning of the qubit movements to perform a series of quantum gates is addressed.

2.2 Qubits in a Fully Connected Star Graph

In this work, a special topology is considered, which we call the fully connected star graph (Def. 3.1). This models a quantum processor based on NV-centres in diamond. It is a graph which consists of k star graphs, which represent the different processing units (PUs). A star with m nodes is a graph with a centre node and $m-1$ leaves, as illustrated in Fig. 1.

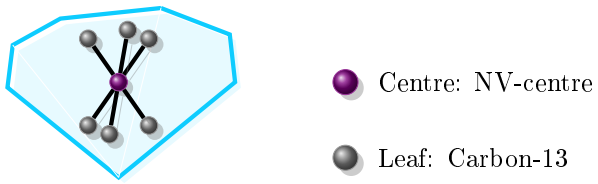


Figure 1: *A quantum processing unit can be based on NV-centres. Such processing unit has one NV-centre and several Carbon-13 atoms as spin qubits around it.*

The centres of the k star graphs are connected with all other centres such that they are part of a complete (fully connected) graph. Examples are shown in Fig. 2 and Fig. 8.

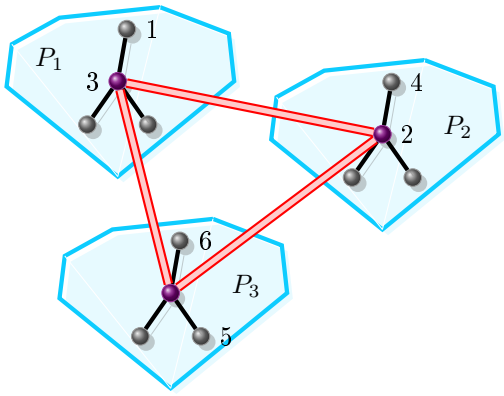


Figure 2: *This is a fully connected star graph with $k = 3$ stars and each star has $m - 1 = 3$ leaves. The stars have the labels $P_1, \ldots, P_3$. There are six data qubits in this graph, numbered $1, \ldots, 6$. Each star (processing unit) has 2 data qubits. The unlabelled nodes carry spare qubits for communication purposes between the processing units.*

Some of the nodes in the fully connected star graph are carrying data qubits. We say a qubit is a data qubit if their state should be preserved; they are mostly involved in calculations. We call the other qubits spare qubits, since these are used for communication purposes between the processors. In Fig. 2, the six data qubits are labelled $1, \ldots, 6$ and are equally distributed among the PUs $P_{1,2,3}$. The unlabelled nodes carry spare qubits.

2.3 Execution of a Quantum Circuit

Given the FCS graph with some data qubits, a quantum circuit can be executed on it. A simple circuit is illustrated in Fig. 3. This can be mapped onto the FCS graph in Fig. 2. The processors P_1 and P_2 have 2 data qubits each, numbered from $1, \ldots, 4$. It is assumed that the given circuit has only 1-qubit and 2-qubit gates, since any quantum circuit can be decomposed into e.g. the universal set of gates $\{H, S, T, CNOT\}$, which consists of only 1-qubit and 2-qubit gates. Furthermore, a 1-qubit gate can be applied on any qubit in the graph. However, 2-qubit gates can only be applied within a processor. If a 2-qubit gate should be performed on qubits in different PUs, then these qubits should be brought together into one processor in order to apply the gate.

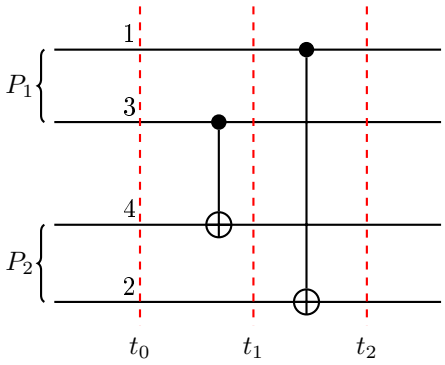


Figure 3: *The processors P_1 and P_2 have 2 qubits each, numbered from $1, \ldots, 4$ as shown. Firstly, at time t_1, a 2-qubit gate on qubits 3 and 4 should be performed. Furthermore, a gate is executed on qubits 1 and 2 within time t_2.*

How the circuit in Fig. 3 can be mapped onto the FCS graph, can be answered by considering the current and desired distribution of the qubits over the nodes in the graph. An example is given in Table 1. Within time t_1, a gate is executed on qubits 3 and 4. Besides, a gate is executed on qubits 1 and 2 within time t_2. In the desired state $\mathbf{y}$ in Table 1, these pairs of qubits are each in one processor. This means that as soon as the desired state is reached, both gates in the circuit can be executed.

In this report, we do not care about how a certain quantum circuit looks like. However, the problem is faced how to obtain the wished qubit distribution $\mathbf{y}$ from the current distribution $\mathbf{x}$ in a fully connected star graph. To reach the desired placement of qubits, some qubits should

Table 1: *An example of the initial distribution of qubits* $\boldsymbol{x}$ *and desired distribution* $\boldsymbol{y}$ *for the graph illustrated in Fig. 2, to execute the circuit in Fig. 3. The violet coloured, circled numbers are the labels of the centres. The other number are the labels of the leaves. The data qubits are numbered as in Fig. 2 by* $1, \ldots, 6$*. A '-' sign indicates a spare qubit.*

PU	P_1				P_2				P_3			
Node	(1)	**2**	**3**	**4**	(5)	**6**	**7**	**8**	(9)	**10**	**11**	**12**
Current **x**	3	1	-	-	2	4	-	-	-	6	-	5
Desired **y**	2	1	-	-	3	4	-	-	-	6	-	5

be moved around. This is done by SWAP operations. Swaps within processors can be executed easily. However, swapping qubits between different processors is done by teleportation (see Section 3.1), which is much more costly.

2.4 Swaps in Practice

In practice, swapping two qubits in different stars/processors can be executed as shown in Fig. 4. Firstly, entanglement (----) is created between the NV-centres of the two corresponding processors, e.g. P_1 and P_2. Then, a swap with the spare leaf in each processor moves the entangled qubits to a leaf at both sides. Thereby, two spare qubits are moved from the leaves to the NV-centres, between which entanglement can be created for a second time. Thirdly, the two entangled qubits in processor P_2 are exchanged. This results in a state where the NV-centre of the one processor is entangled with a memory qubit in the other processor. Lastly, the two qubits are teleported to their destinations.

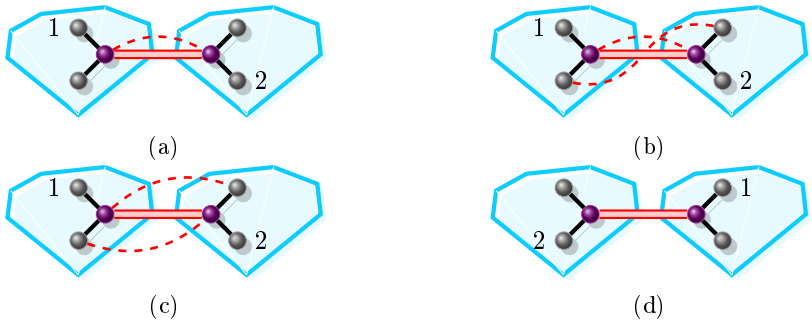


Figure 4: *A* SWAP *operation between distant qubits is performed in multiple steps, though in one time step. Firstly, entanglement is created between both centres (a). Secondly, each entangled qubit is swapped with the spare qubit (b), after which entanglement is created between the centres for the second time. Thirdly, the two entangled qubits in exactly one of the processors should be exchanged (c). Lastly, the two qubits can be teleported to the other side (d).*

Whether a swap within a processing unit or between them is performed, the same symbols are used in both cases (see Fig. 5). Consider the example begin and final distribution of qubits as given in Table 1. Knowing swaps, Fig. 6 shows how the desired distribution **y** can be reached from the current one **x** by doing SWAP operations.

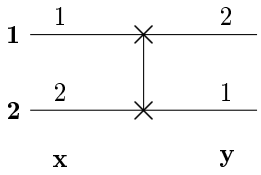


Figure 5: *The qubits 1 and 2 carried by two wires **1** and **2** can be swapped, such that the qubits will occur in reversed order at the output **y** with respect to the input **x**.*

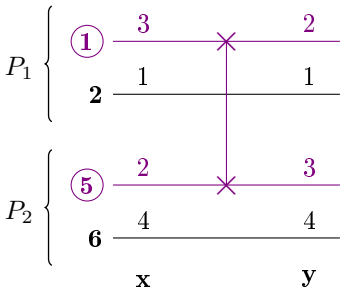


Figure 6: *The current qubit placement **x** from Table 1 can be transformed into the desired placement **y** by doing several* SWAP *operations. The bold numbers at the left side correspond to the wire numbers. A number above a wire indicates which qubit occupies the wire at that point.*

2.5 Problem

Now, our first goal is to minimize the number of quantum operations. These operations are firstly, the gates within processors and secondly, the moves of data qubits between the PUs. Details about the minimum number of operations are given in sections 5.5, 5.13 and 7.2. Our second goal is to minimize the time. In this sense, qubit moves between processors are much more costly than applying gates within a PU. This is motivated by the fact that moving qubits between PUs is done by teleportation (see Section 3.1), which is a relatively time-intensive procedure compared to applying gates in a processor. Therefore, we will neglect the gate execution time within processors and count only the moves between PUs. More about minimizing the time is stated in sections 6.1 and 7.3.

2.6 Related Work

The general form of our problem where the number of steps should be minimized, is described for the first time by Alon et al [1] by his *routing via matching* model. In his work, he states that the routing number $rt(G)$ (see Section 3.4) for a complete graph G is equal to 2. This result might be used to determine the routing number for the FCS graph (Section 6.1).

Other related work includes [10], which is based on the same model, where several graph topologies are considered. One of them is the fully connected graph, which is closely related with the FCS graph. Moreover, their problem P_1 [10, p. 7] is a slightly different version of the problem faced in the current report. However, in this paper, this problem is tackled for a topology which is not considered before -to the best of our knowledge. Furthermore, our problem is closely related to the *direct routing* model, introduced in [11]. It is a model where each data packet (qubit) moves along a direct path to its destination, without being buffered at intermediate nodes. However, [11] does not consider fully connected star graphs neither.

[12] considers the general case of routing permutations in connected graphs (graphs where each pair of nodes is connected via a path). It gives better bounds than before on the complexity of verifying the routing time $rt(G, \pi)$ (see Section 3.4) of a given permutation π in a graph G. Meanwhile, several notions are set forth like when a cycle (Def. 4.4 in Section 4.5) is *individually routable* and when a pair of cycles is *mutually routable*, which can be used in planning the qubit moves. The authors show -among others- how to route qubits in a complete graph in only 2 steps, though, not how to route qubits in an FCS graph efficiently.

2.7 Motivation and Contribution

Sorting algorithms can be modified such that they can be used to plan the qubit moments in FCS graphs. Several optimal, classical sorting algorithms are known to sort numbers on restricted topologies like complete graphs, grids or 1D-nearest neighbour (nodes on a line) topologies. These include Merge sort, Heapsort and Selection sort. However, they are not optimal for the FCS graph for several reasons. Firstly, an abstraction (Section 4.4) can be made of the FCS graph to a complete graph with abstract nodes each comprising m nodes from a star in the original graph. To a certain extent, this abstraction can be seen as m complete

graphs, by which the number of steps can be optimized when we consider single moves/half swaps. However, handling this graph as m complete graphs cannot be used to optimize the number of steps with full swaps. So existing, efficient algorithms for complete graphs cannot be used to optimize the number of steps. Secondly, other graph topologies for which efficient algorithms exist, differ too much from the FCS graph to be applicable to the FCS graph.

This report describes methods to reach an arbitrary permutation $\mathbf{y}$ of the current qubit distribution $\mathbf{x}$ by using either full swaps of qubits or by using half swaps/qubit moves. Thereby, several variants of the permutation problem are considered. Firstly, an algorithm can be optimized to produce a schedule using the minimum number of swaps and secondly, the produced routing scheme can be optimized in terms of the execution time in practice. Both cases are discussed and solutions are proposed to (nearly) optimize the number of swaps (Section 5) and to minimize the number of moves (Section 7). Besides, guidelines are given to minimize the number of steps using swaps and it is conjectured that the number of steps via qubit moves can be optimized too, by using the proposed theory. These solving methods (algorithms) require time and classical memory space which is polynomial in the size of the FCS graph, which makes these algorithms efficient in this sense.

Our contributions are divided as follows over this paper.

Section 3.

- Some background information on the problem is given regarding quantum state teleportation

  in theory and

  in practice.
- This is supplemented with notations and
- Definitions.

Section 4.

- After posing a generalized problem of planning qubit movements (Section 4.2), a simple routing algorithm is proposed (Algorithm 1 in Section 4.3) which solves this problem.
- The FCS graph is abstracted (Section 4.4) to a move graph, after which
- Properties of the move graph are discussed (Section 4.5).

Section 5.

- Starting with the problem of planning qubit movements via full swaps, it is shown that in order to reach the optimal number of swaps, the abstracted graph should be decomposed into as many cycles as possible (Sections 5.3 - 5.5).
- Then a method is proposed to solve this problem in most cases (Sections 5.6 - 5.8).
- This has been implemented in the language C and compared to the aforementioned simple algorithm, in Section 5.9, and discussed further in Section 5.10.
- This is followed by presenting and discussing a simple, optimal routing algorithm which sorts the qubits via swaps (Section 5.12).
- It is shown how to minimize the total number of swaps, so including the swaps within the processing units/stars.

Section 6.

- Guidelines are proposed to minimize the number of steps (Section 6.2).

Section 7.

- The problem of routing qubits via moves/half swaps is set out and the minimum number of moves is shown (Section 7.2).
- Assuming that each star has 2 spare qubits, the optimal number of steps is given in Section 7.3.
- To reach this optimum, a special function is needed, which is proposed and discussed in Sections 5.6 - 5.7.
- An algorithm is presented solving the routing problem via moves, being optimal in both the number of moves and steps (Section 7.6).
- Under the condition that each star has at least one qubit, guidelines are proposed to minimize the number of steps, while using the minimum number of moves (Section 7.8).
- The application of the proposed algorithms to other graphs than the fully connected star graph is discussed in Section 7.9.

Section 8.

- A summary is given,
- Conclusions and
- Open research questions.

2.8 Outline

Firstly, some background information on the routing problem is given in section Section 3. Since existing algorithms are not fully suitable for our purpose, new permutation routing algorithms are demanded. The exact problem definition is given in Section 4.2, after which a simple, non-optimal in-place sorting algorithm is presented (Section 4.3). Section 4.4 gives an abstraction of the problem, thereby focussing on the relevant parameters to optimize. By changing the parameters, the problem can appear in different contexts and several cases are addressed. In the first case, full swaps with qubits are considered in Section 5 and an algorithm is presented to minimize the number of full swaps (Algorithm 6 in Section 5.12). This algorithm uses a certain function as basis, which has been implemented and is discussed in Sections 5.6 - 5.8. In Sections 5.13 and 6.1, notes are given for further optimizations.

Then a fully connected star graph where qubit moves are allowed is addressed in Section 7. Under certain conditions, the number of moves (half swaps) is minimized, together with the number of time steps. A list of rules is composed and used in an efficient routing algorithm (Section 7). With this, both the number of moves and steps are optimized. A more general case where moves are allowed, is discussed in the following Section 7.8. In addition, the application of the algorithms to other graphs than the fully connected star graph is addressed. Finally, the results are discussed and conclusions are listed in Section 8. To finalize, this is supplemented by several open questions.

3 Background on the Routing Problem

As stated in the introduction, a quantum computer would be useful to do certain calculations faster than any classical computer, like searching entries in databases [13]. In order to do so, quantum hardware is required. Therefore, a good architecture is being searched nowadays, which lays down which pairs of qubits can interact with each other.

In this work, a special topology (see Fig. 2 or Fig. 8) is addressed, which is derived from the interactions possible in NV-centre based quantum processors -as written before. In order to execute a quantum circuit with this processor architecture, an algorithm is needed which plans which gate should be executed in which time slot. Thereby, it is assumed that the quantum circuit consists of only 1 and 2-qubit gates. A 1-qubit gate can be executed on any qubit and therefore, their execution does not have to be planned. In contrast, 2-qubit gates can only be used on qubits in nodes with an edge in between, for which planning is demanded in order to execute the circuit as fast as possible.

During the circuit execution, the distribution of the qubits over the FCS graph changes over time. Given some distribution $\mathbf{x}$, another distribution $\mathbf{y}$ should be reached as fast as possible. In this work, only the time for the qubit teleportations (in fact the measurements) is counted, but not the time demanded by the gates within the processors. More about the teleportation procedure follows in the next section.

3.1 Quantum State Teleportation

Using the quantum properties of atomic particles, quantum entanglement can be used to teleport the quantum state of a qubit, see Fig. 7. The teleportation procedure is as follows. Firstly,

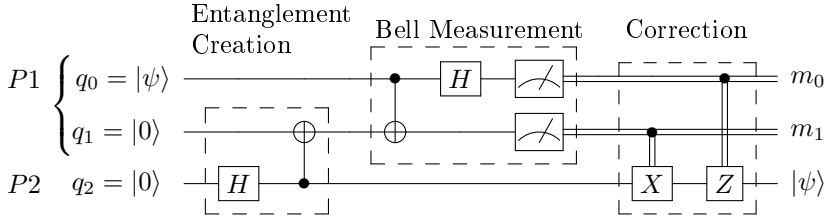


Figure 7: *This is a schematic representation of the procedure to teleport one qubit $|\psi\rangle$, using a shared Bell state ($\beta_{00} = 1/\sqrt{2}\,(|00\rangle + |11\rangle)$) and two classical bits m_1 and m_2.*

entanglement is created between two distant qubits, such that the two qubits share the entangled state $\beta_{00} = 1/\sqrt{2}\,(|00\rangle + |11\rangle)$, written conform the Dirac notation. Let these qubits be q_1 and q_2, being present in quantum processors 1 and 2, respectively. When there is another qubit $q_0 = |\psi\rangle$ in processor 1 to be teleported to processor 2, then a Bell measurement is performed using one entangled qubit q_1 and the qubit q_0 to be teleported. After the measurements, the state of qubit q_0 is not $|\psi\rangle$ any longer. Finally, by using the measurement outcomes, qubit q_2 should be corrected, such that its state becomes equal to $|\psi\rangle$. This correction follows from Fig. 7.

It follows that to teleport a qubit from processor P_1 to P_2, both PUs should have at least one spare qubit. Furthermore, the entangled qubit q_1 and data qubit q_0 should be in the one and the same PU, in nodes with an edge in between, otherwise the Bell measurement cannot take place.

Applying this to the FCS graph, in a star, each edge is connected with the centre node. This means that there is no disjoint set of edges in a star comprising more than 1 edge. Therefore, only one qubit can be teleported from a particular PU at the same time.

3.2 Teleportation In Practice

The described method to teleport a quantum state is valid in theory, though in practice, the procedure includes some more operations. One procedure is described in [4], which is roughly as follows.

Consider two distant qubits q_1 and q_4. From both sides, an entangled qubit is sent to a box in the middle. Qubit q_2 is sent from the side of qubit q_1 and q_3 from the side of q_4. Their states are $|\psi_{12}\rangle = |\psi_{34}\rangle = \beta_{00}$. Then qubit q_1 will get entangled with q_4 as soon as the qubits q_2 and q_3 are entangled with each other. However, the probability of getting wished entanglement is not necessarily 1.

After measurements in the box, qubits q_2 and q_3 will share one of the four Bell states with equal probability. If the state is $|\psi_{23}\rangle = \beta_{00}$, then the state is as desired and achieved with probability 1/4. If the state is $|\psi_{23}\rangle = \beta_{01} = 1/\sqrt{2}\,(|01\rangle + |10\rangle)$, then a local correction with an X-gate will reach the wished state β_{00}. However, if the state is $|\psi_{23}\rangle = \beta_{1x}$, then the state β_{00} cannot be reached when the procedure is done using only linear optics -which is often done. In

that case, entanglement between qubits q_1 and q_4 is created in the box with probability 1/2. More complex optical materials would be required to raise this probability of success. For more details on the teleportation procedure, see e.g. [4].

Performing the Bell measurement illustrated in Fig. 7 is the most time-consuming process during the teleportation process. It costs much more time than performing gates within a PU, which is the reason why we only count the teleportations and not the gate execution times in the processors, when solving the routing problem.

3.3 Notations

Some notations used in this report are summarized below.

$[n]$	The set of integers $1, 2, \ldots, n$.
$G = (V, E)$	A graph G having vertex set V and edges E.
$G' = (P, E')$	A graph G' having (abstract) nodes P_i and edges E'.
P_i	The ith processor (node) in move graph G'.
(i, j)	The directed edge from node P_i to P_j.
w_{ij}	A weight $\in \mathbb{Z}_{\geq 0}$ of edge (i, j), indicating the number of qubits to move from node P_i to P_j.
W	The move matrix containing the weights $w_{ij} = [W]_{ij}$.
μ_i^+	The number of data qubits to move outwards from node P_i
μ_i^-	The number of data qubits to move inwards to node P_i
μ	The total number of qubits to move around through the move graph G'.
$\mathcal{C}$	A collection of cycles (Def. 4.4).
ζ	The number of cycles in cycle decomposition $\mathcal{C}$.
β	The number of swaps required to solve Problem 2 with move graph G'.
M	The number of moves (half swaps) required to solve Problem 3 with graph G'.
$\mathcal{N}_i^+$	The set of out-neighbours of node P_i.
$\mathcal{N}_i^-$	The set of in-neighbours of node P_i.
$d_{NU}(G')$	The number of steps required to solve Problem 3 with move graph G'.

3.4 Definitions

Definition 3.1. A *Fully Connected Star graph* (FCS graph) is a graph consisting of star graphs from which only the centres are connected with each other, such that the connections between the centres form a fully connected graph.

Fig. 8 shows an example of an FCS graph with $k = 4$ stars, each having $m = 4$ nodes.

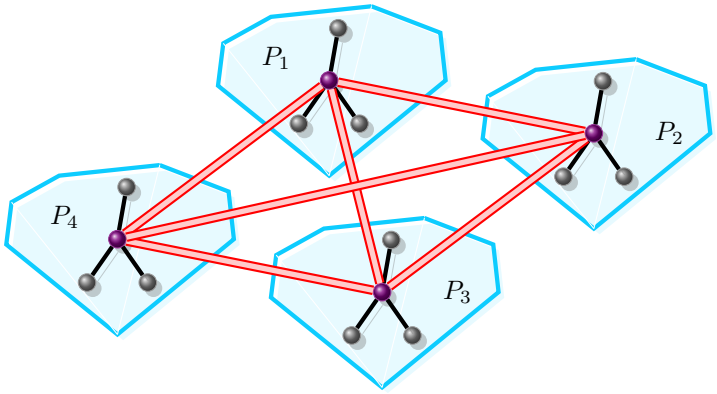


Figure 8: *This figure shows a fully connected star graph with* $k = 4$ *centers and each centre has* $m-1 = 3$ *leaves. The four stars are labelled as* $P_1, \ldots, P_4$.

The *routing time* $rt(G, \pi)$ for a given permutation *pi* in graph G is the minimum number of time steps needed to route the qubits to reach the desired permutation.

Now, let us define the distance between two nodes in the move graph.

Definition 3.2. The *distance* between two nodes in move graph G' is equal to the length of the shortest path between them via only edges (i, j) satisfying $w_{ij} > 0$.

Furthermore, we define the set of shortest paths Q_{ij}.

Definition 3.3. Given the move graph $G' = (P, E')$, the *set of shortest paths* Q_{ij} from node P_i to P_j is the set

$$Q_{ij} := \bigcup \{(v_1, v_2, \ldots, v_l) \in P \times P \times \cdots \times P : l - 1 = l_{min}\} \tag{1}$$

where l_{min} is the length of the shortest path in G' from node P_i to P_j.

Besides, we define the cut-set.

Definition 3.4. An s-t cut $C = (S, T)$ is a partition of Q_{ij} such that $P_i \in S$ and $P_j \in T$. The *cut-set* X_{ij} is the set

$$X_{ij} := \{(u, v) \in Q_{ij} : u \in S, v \in T\} \tag{2}$$

The capacity $c(S, T)$ can be defined now.

Definition 3.5. The capacity $c(S, T)$ is the weighted sum of the paths going from partition S to T:

$$c(S, T) = \sum_{(u,v) \in X_C} w_{uv} \tag{3}$$

With this, we define the flow f_{ij} from node P_i to P_j as follows.

Definition 3.6. The *flow* f_{ij} from node P_i to P_j is the minimum value of the capacity $c(S, T)$:

$$f_{ij} = \min c(S, T) \tag{4}$$

4 Problem Definition of Routing Qubits in Fully Connected Star Graphs

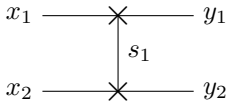
Given a fully connected star (FCS) graph, where each vertex has a unique label and a certain qubit distribution, one could be interested in permuting the qubits with the minimum number of swaps or by using the minimum number of steps. Then the challenge arises to design a classical algorithm which solves these questions. Preferably, this algorithm is as fast as possible and it should use only a limited amount of memory space on a classical computer to come up with a qubit routing scheme by which the number of swaps and/or steps is minimized. Such a classical algorithm which uses only limited amount of memory space in terms of the problem size, to solve the routing problem with a classical computer, is called an *in-place* algorithm. Advantageously, the produced routing scheme uses the least amount of quantum memory during the execution.

4.1 Terminology

Before defining the problem exactly, some definitions are given. As written in the introduction, a SWAP operation is an operation where two qubits are interchanged, under the condition that there is a link between the nodes carrying the qubits.

Definition 4.1. Given a graph $G = (V, E)$ with nodes V and edges E, a swap operation $\text{SWAP}(i, j)$ only acts over some edge $(i, j) \in E$ between nodes i and j by interchanging the qubits v_i and v_j.

Fig. 9 shows an example of a swap operation on two lines 1 and 2, carrying the qubits x_1 and x_2, respectively. If the swap is done, the qubits are swapped and otherwise, they stay in place.



$$(y_1, y_2) = \begin{cases} (x_2, x_1) & \text{if } s_1 = 1 \\ (x_1, x_2) & \text{if } s_1 = 0 \end{cases}$$

Figure 9: *The qubits carried by two wires can either be swapped ($s_1 = 1$), such that the output values are $y_1 = x_2$ and $y_2 = x_1$, or not ($s_1 = 0$), such that the outputs are equal to the inputs ($y_{1,2} = x_{1,2}$).*

Since a swap in a processor can be executed much faster than between PUs, we will ignore the time required for swaps in PUs for the moment. Therefore, at each time *step*, a disjoint collection of swaps between PUs is chosen and the qubits are moved accordingly.

4.2 Problem Definition

Now the actual problem can be written as follows. Let $G = (V, E)$ be an FCS graph with $|V| = n$ vertexes, satisfying the following properties:

1. The graph G consists of exactly k star graphs;
2. Each star has m nodes, such that it has $(m-1)$ leaves;
3. Each node has a unique label $j \in [n]$;
4. Each node j carries a qubit $v_j \in \mathbb{Z}_{\geq 0}$;
5. The pebbles v_i and v_j can only be swapped if nodes i and j are connected by edge $(i, j) \in E$ (sometimes called a *matching*);
6. A swap over an edge which connects the centre of a star with a leaf has costs $a \in \mathbb{R}_+$.
7. A swap (or 2 half swaps/moves) over an edge interconnecting two stars has costs $b \in \mathbb{R}_+,\ b > a$;
8. The centre in the ith star has the label $m(i-1)+1,\ i \in [k]$;
9. The leaves in the ith star have the labels $j,\ m(i-1)+1 < j \leq mi, i \in [k]$.

Note that from properties 1 and 2 follows that the total number of nodes n is equal to

$$n = km \tag{5}$$

As stated before, in order to execute a quantum circuit, the distribution of the qubits over the FCS graph changes over time. Given the current distribution and a wanted distribution, the question is how to swap the qubits such that the desired distribution will be reached, while minimizing the costs for the SWAP operations and/or the time.

Problem 1. In a fully connected star graph with properties 1 - 9 mentioned above, let the initial assignments x_j of all nodes $j \in [n]$ be stacked in an input vector $\mathbf{x} \in \mathbb{Z}^n_{\geq 0}$ and let the permuted, final assignments be $\mathbf{y} \in \mathbb{Z}^n_{\geq 0}$. Given the costs a and b for performing the swaps and costs $c \in \mathbb{R}_+$ for each step, find an in-place routing algorithm which orders the qubits from the

initial distribution $\mathbf{x}$ to the final distribution $\mathbf{y}$, such that the total costs are minimized (Eq. 7).

A mathematical description of the problem is given below. Let the step function be defined as

$$\text{step}(v) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The edge set E_c contains all edges between the nodes $j = m(i-1)+1$, being the centre of the ith star ($i \in [k]$). $\Pi(i,j)$ is the permutation matrix such that $\Pi(i,j)\mathbf{x}$ is the vector $\mathbf{x}$ in which the elements x_i and x_j are swapped. The variables $s(l,i,j)$ are used for the swap operations. Only if the qubits on wires i and j should be swapped in time step l, variable $s(l,i,j)$ is 1, otherwise 0.

$$\min \quad a \sum_{(i,j)\in E\setminus E_c} s(l,i,j) + b/2 \sum_{(i,j)\in E_c} s(l,i,j)[\ \text{step}(v_i) + \ \text{step}(v_j)] + cd \tag{7}$$

s.t.

$$\mathbf{y} = \left(\prod_{l=1}^{d} S_l\right)\mathbf{x} \qquad \mathbf{x},\mathbf{y} \in \mathbb{Z}^n_{\geq 0}$$

$$S_l = \prod_{(i,j)\in E} s(l,i,j)\Pi(i,j) \qquad \forall l \in [d]$$

$$\sum_{j\in V} s(l,i,j) \leq 1 \qquad \forall l \in [d], \forall i \in [n]$$

$$s(l,i,j) + s(l+1,i,j) \leq 1 \qquad \forall l \in [d-1], \forall (i,j) \in E$$

$$s(l,i,j) \in \{0,1\} \qquad \forall l \in [d], \forall (i,j) \in E$$

Note that the one but last condition is not necessary. It states that if a gate uses wires i and j in step l, then it may not be used in the next step $l+1$. This would be inefficient and is implied in the other conditions and the fact that the number of swaps should be minimized.

From now, the case $a = 0$ is considered, such that the number of swaps in the stars will not be optimized. The parameters b and c may vary, depending on the context of the problem. Both the cases $b = 0, c > 0$; $b > 0, c = 0$ and $b, c > 0$ will be addressed later.

4.3 A Simple In-Place Sorting Algorithm

It can be concluded from section 2.6 that there exist several routing algorithms at the moment. However, they are all not specifically intended for our problem, nor directly applicable to the FCS graph. Nevertheless, a trivial sorting algorithm can be given for Problem 1, like Algorithm 1. This algorithm is only considered to set a benchmark on the more advanced routing algorithm which is proposed later.

For an FCS graph $G = (V, E)$, the simple sorting algorithm 'RouteSimple' (Algorithm 1) firstly brings the correct qubits in the first star, then the second star will get the correct qubits, and so on (similar ot the existing algorithm Selection sort). Thereby, qubits are only exchanged over the edges E. Finally, the function returns a list L of the swaps done.

Algorithm 1 routes the qubits from the initial state $\mathbf{x}$ to the final state $\mathbf{y}$. However, it does not take the number of swaps into account, nor the number of steps. Nevertheless, the algorithm gives an indication of what existing algorithms would achieve and it will be used later in this work to compare the proposed, more advanced algorithm with (Algorithms 6 and 8). Note that only the swaps between stars are counted.

This algorithm has been implemented in C. It can be found online at `https://github.com/jkeur/routeSwaps.git`.

Algorithm 1: RouteSimple

Data: The initial placement $\mathbf{x}$ and desired placement $\mathbf{y}$ of the qubits and the number of stars k in the FCS graph G.
Result: A list L containing the swaps done.

```
1  v := x ;                                          // Use a copy of the initial state
2  m := length(x)/k ;                                // Set #nodes per star
3  foreach p ∈ [k] do                                // For each star p
4      i := m(p − 1) + 1 ;          // i gets the label of the centre node of star p
5      for q := p + 1 to k do                        // For each unordered star q > p
6          if star q has the right numbers then
7              break;
8          while star q has ≥ 1 qubit for star p do
9              j := m(q − 1) + 1 ;     // Set the label of the centre node of star q
10             SWAP(i, l) such that the destination of v_i is not in star p;
11             SWAP(j, l) such that v_j has a destination in star p;
12             SWAP(i, j);
13             ADD2LIST(i, j) ;                      // Add swap (i, j) to the list L
```

4.3.1 Time Complexity Algorithm 1

The time complexity of the algorithm above in the worst case is as follows.

Lemma 4.1. *For an FCS graph with n nodes and k stars, Algorithm 1 runs in time $\mathcal{O}(k \log n)$.*

Proof. The first `for`-loop (line 3) requires time $\mathcal{O}(k)$. The second `for`-loop (line 5) together with the `while`-loop (line 7) requires time $\mathcal{O}(\log(mk))$. This makes the total runtime $\mathcal{O}(k \log n)$. □

4.3.2 Space Complexity Algorithm 1

Regarding the classical memory space requirements, this depends on the variables used. These are the vectors $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{v}$, the single parameters k, m, i, j, l, p and q, and the list of swaps L. The three vectors need space $\mathcal{O}(3n \log n)$. The seven single parameters demand space $\mathcal{O}(\log m + 3 \log k + 3 \log n)$, since their ranges are $[k]$, $[m]$, $[n]$, $[n]$, $[n]$, $[k]$, $[k]$, respectively. Finally, the list L requires space $\mathcal{O}(2m(k-1) \log n)$. This give a total space in the order $\mathcal{O}(\log m + 3 \log k + (5n - 2m + 3) \log n)$.

4.4 An Abstraction of the Fully Connected Star Graph

As the focus lies on the swaps between the stars, Problem 1 can be abstracted as follows. Firstly, each star is represented as a big node P_i ($i \in [k]$) containing the nodes of that star:

$$P_i = \{j | j \in V, m(i-1) + 1 \leq j \leq mi\} \tag{8}$$

These nodes are part of the complete directed graph G', which is defined below.

Definition 4.2. The *move graph* $G' = (P, E')$ is a complete directed graph containing the nodes $P_i, \forall i \in [k]$ and directed edges $(i, j) \in E'$, each corresponding to the direct communication link from node P_i and P_j in practice, and each having a weight w_{ij}, which indicates the number of qubits to be moved from star P_i to P_j in the original graph G ($i, j \in [k]$); and if a qubit in star P_i should end in the centre of star P_j, then the notation $(w_{ij}, 1)$ is used as weight for edge (i, j).

Since the edges E' are the edges between the centres, $|E'| = |E_c|$ holds. In the case that

the original graph G is an FCS graph, $|E'| = k(k-1)/2$ is true for k stars. As example, the move graph of the problem instance in Table 2 for graph G in Fig. 8 is shown in Fig. 10.

Table 2: *An example of the initial placement* $\boldsymbol{x}$ *and target placement* $\boldsymbol{y}$ *of the qubits in the graph illustrated in Fig. 8. It is an example of Problem 1 for an FCS graph with* $k = 4$ *stars, each having* $m = 4$ *nodes. The violet coloured, circled numbers are the labels of the centres.*

PU	P_1				P_2				P_3				P_4			
Node	①	**2**	**3**	**4**	⑤	**6**	**7**	**8**	⑨	**10**	**11**	**12**	⑬	**14**	**15**	**16**
Current **x**	15	8	7	1	2	9	14	16	5	12	3	13	11	6	4	10
Desired **y**	3	8	6	1	15	7	10	16	5	12	14	2	11	13	4	9

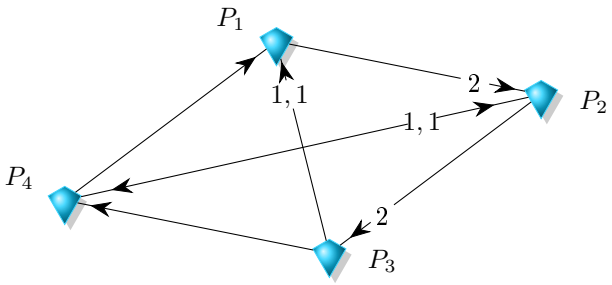


Figure 10: *This is the move graph* G' *for the problem instance given in Table 2 for graph* G *in Fig. 8. Only the edges* (i, j) *satisfying* $w_{ij} > 0$ *are shown. If no number is given at an edge, its weight is 1. If one number is written at edge* (i, j)*, then this indicates its weight* w_{ij}*. Lastly, if two numbers are given, then the first is the edge weight* w_{ij} *and the second number denotes that one qubit from star* P_i *should end in the centre of star* P_j*.*

4.5 Properties of this Abstraction

The edge weights w_{ij} of the graph G' can be stacked in a matrix W.

Definition 4.3. The *move matrix* is the matrix $W \in \mathbb{Z}_{\geq 0}^{k \times k}$, where element w_{ij} is the weight of edge $(i, j) \in E'$, i.e. it represents the number of qubits that should be moved from star P_i to P_j in the original graph G ($i, j \in [k]$). Furthermore, w_{ii} is the number of qubits to stay in star P_i.

One property of this matrix is as follows.

Lemma 4.2. *For the move matrix* W*, it holds that*

$$\sum_{i,j} w_{ij} = n \tag{9}$$

Proof. For each star $P_i \in P$, a qubit therein should either stay there or should be moved to some other star. In the first case, the number of qubits is given by w_{ii}. The number of qubits in the latter case, is the sum of the other weights in the ith row of the matrix W: $\sum_{j \neq i} w_{ij}$. Since there are n qubits, the sum of the weights is n. □

Another property which can be derived from the move matrix, is the total number of qubits to move μ.

Lemma 4.3. *Given a move matrix W, the total number of qubits to move μ is equal to*

$$\mu = \sum_{i,j,j \neq i} w_{ij} \tag{10}$$

Proof. This follows from Eq. 9 and the fact that $\sum_i w_{ii}$ is the total number of qubits which are in the correct star: $\mu = n - \sum_i w_{ii}$. □

In the move graph G' in Fig. 10, it can be observed that one qubit has to go from node P_2 to P_4 and one qubit in the reverse direction. This is called a cycle of length 2.

Definition 4.4. A *cycle* in graph G' is a simple closed path containing only edges $(i, j) \in E'$ satisfying $w_{ij} > 0$.

For example, cycles in the move graph illustrated in Fig. 10 include: $\{(2,4),(4,2)\}$ with length 2, $\{(1,2),(2,3),(3,1)\}$ having length 3 and $\{(1,2),(2,3),(3,4),(4,1)\}$ being a 4-cycle.

So far, the routing problem is abstracted by which our problem is fully described by the move graph $G' = (P, E')$. When the locations of the qubits in the stars do not matter, in other words, if it does not matter which qubit is in a centre or in a leaf, then the move matrix W is sufficient to describe our problem.

5 Routing Qubits with the Minimum Number of Swaps

Problem 1 can be restricted to have either only data qubits or both data and spare qubits. As stated in the introduction, a data qubit carries information to be preserved, while the state of a spare qubit may be destroyed. If spare qubits are present, they are used for communication purposes between PUs via teleportation.

This section considers a model in which only full swaps of qubits are allowed. So a SWAP operation over an edge will flip the qubits in the nodes at the edge's endpoints. This holds for both swaps within a processor, as well as for swaps between PUs. In principle, the following problem will be addressed in this section.

Problem 2. This problem is equivalent to Problem 1, where the costs for swaps within PUs are zero ($a = 0$), the costs for swaps between PUs are non-zero ($b > 0$) and a data qubit v_i is carried by each node i in the FCS graph.

5.1 A Practical View on this Problem

The following translation could be made to the NV-centre based quantum processor in reality. Each star P_i with m nodes can be regarded as an NV-centre (centre qubit) having access to $m + 1$ memory qubits (leaves). m of the memory qubits correspond to the m unique values v_j in the star P_i. The other memory qubit is a spare qubit and the NV-centre itself acts as a communication link with other stars, which is modelled as a spare qubit too. In this model, it is assumed that when a data qubit is moved from node P_i to P_j, then always another data qubit will be moved back from node P_j to P_i.

5.2 Approach: an Overview

Solving Problem 2 will be done in several phases. Firstly, it is shown that the move graph $G' = (P, E')$ can be decomposed into cycles. There seems to be a direct relation between the number of cycles in the decomposition and the number of swaps required to sort the qubits from the current to the desired distribution. Having shown this relation, rules are proposed to

find a (nearly) optimal decomposition. These rules can be used successively and repeatedly in an algorithm until a decomposition has been found. Such a decomposition can be used directly by an algorithm, to sort the qubits using the minimum number of swaps.

5.3 Decomposition of the Move Graph into Cycles

Having introduced an abstraction (in Section 4.4) to represent a problem instance of Problem 1 and given the notion of cycles, we will focus on the cycles now. It is known that each permutation of a finite set of objects partitions that set into cycles. This holds for move graphs too.

Lemma 5.1. *Given some move graph $G' = (P, E')$ with corresponding move matrix W, let the number of qubits μ_i^+ to be moved from node P_i to other nodes be*

$$\mu_i^+ = \sum_j w_{ij} \tag{11}$$

Then move graph G' can be decomposed into cycles, such that each node P_i is part of μ_i^+ cycles.

Proof. For each qubit $v_j \in P_i$ to be moved outwards, another qubit v_l should be moved to node P_i. In other words, if μ_i^+ ($\mu_i^+ \leq m$) qubits in node P_i should be moved to other nodes, then μ_i^+ qubits should be moved from other nodes to node P_i. This implies that each node P_i is part of μ_i^+ cycles (with length ≥ 2).

□

Fig. 11 shows an example of a move graph G', which can be decomposed into cycles in two ways. The first decomposition consists of three 3-cycles and the second decomposition has one 3-cycle and one 6-cycle. It is known that for any permutation in a complete graph, the decomposition into cycles is unique. In contrast, this example shows that this is not the case for the move graph from an FCS graph.

5.4 Relation Between the Number of Cycles and Swaps

From Lemma 5.1 follows that every move graph G' can be decomposed into cycles. It is known that the length of a cycle is proportional to the number of swaps required to apply the cyclic

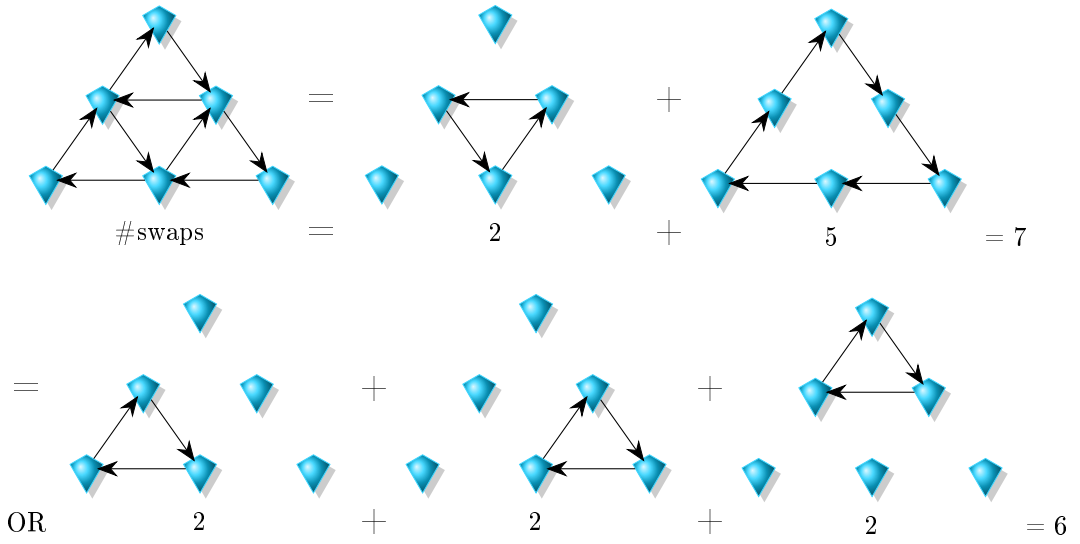


Figure 11: *The move graph in the top left corner can be decomposed into cycles in two ways. The decomposition on top consists of 2 cycles, while the decomposition below has 3 cycles. 7 swaps are demanded to sort the qubits according to the first decomposition and only 6 swaps according to the second decomposition.*

permutation: An n-cycle needs $n-1$ swaps to sort. Similarly, the minimum number of swaps to apply the permutation $j \rightarrow \pi(j)$ on the current qubit distribution is determined by the metric called the Cayley distance. It is well-known that given the current qubit placement j and desired placement $\pi(j)$, the Cayley distance $T(j, \pi(j))$ is the minimum, total number of swaps, transforming j into $\pi(j)$:

$$T(j, \pi(j)) = n - \#\text{cycles in } (j\pi(j)^{-1}) \tag{12}$$

where n is the size of the permutation, being equal to the number of qubits in the FCS graph. In this relation, trivial 1-cycles are included in the number of cycles (which are disregarded in our work, according to Def. 4.4).

"This relation is easy to prove. By invariance, take $\pi(j) - id$. If j is a k-cycle, it takes $k-1$ moves to sort, and disjoint cycles take separate sorting operations." [14, p. 117-118]

5.5 The Optimal Number of Swaps Between Centres

The Cayley distance is now rewritten in terms of the parameters used in this report.

Theorem 5.1. *Given a move graph G' with μ qubits to move (Eq. 10), a decomposition with $\zeta_{\geq 2}$ cycles of length ≥ 2 requires β swaps, being*

$$\beta = \mu - \zeta_{\geq 2} \tag{13}$$

Proof. The total number of cycles ζ -including trivial 1-cycles- is obviously $\zeta = \zeta_1 + \zeta_{\geq 2}$, where ζ_1 is the number of 1-cycles and $\zeta_{\geq 2}$ is the number of the remaining cycles in the cycle decomposition. It was mentioned in the proof of Lemma 4.3 in Section 4.5 that $\mu = n - \sum_i w_{ii}$, which is equal to $\mu = n - \zeta_1$. Since $\beta = T(j, \pi(j))$, Eq. 13 follows:

$$\begin{aligned} \beta = T(j, \pi(j)) &= n - \#\text{cycles in } (j\pi(j)^{-1}) \\ &= n - \zeta_1 - \zeta_{\geq 2} \\ &= n - \sum_i w_{ii} - \zeta_{\geq 2} \\ &= \mu - \zeta_{\geq 2} \end{aligned}$$

□

The Cayley distance tells us, that the more cycles in our decomposition $\mathcal{C}$, the less swaps are needed to sort. From this can be concluded that in order to minimize the number of swaps, the number of cycles in the decomposition should be maximized. In this context, it can be concluded that the first decomposition in Fig. 11, is the best.

5.6 Finding an Optimal Decomposition

To reach the optimal number of swaps (13), a decomposition of move graph G' should be found with the maximum number of cycles. In order to find this, one could search for possible combinations of cycles and pick the best decomposition. This is a intensive approach, and could lead to an exponential search time. However, by using the following rules, an optimal cycle decomposition can be found for many instances of Problem 2.

The rules act on a copy of the move matrix W, say W'. An algorithm implementing these rules should store the cycle decomposition $\mathcal{C}$ and besides, it should store what we call a *path*

table, which is described in rule B.

Rule A: Handle Special Cycles. Repeatedly, remove one of the shortest cycles $C \subseteq E'$ from W' and add it to the cycle decomposition $\mathcal{C}$ if it has at least $|C| - 1$ edges $(i, j) \in C$, for which holds that the shortest path from node P_j to P_i has length $l - 1$, where l is the length of the shortest cycle in W'; and if the flow f_{ji} (Def. 3.6) from node P_j to P_i satisfies $f_{ji} \leq w_{ij}$; thereby, if the cycle comprises an edge (i, l) which can be reached by walking via nodes P_j, as stored in the path table (see rule B), then the cycle should be extended with that path and that path (i, j, l) should be deleted from the path table.

The *deletion of a path* (i, j, l) from the path table means that the weight w for the path, which is stored in a quadruple (i, j, l, w), is decreased by one.

The *deletion of cycle* C from the move graph G' implies that the weights w_{ij} are decreased by one for each edge $(i, j) \in C$ and the weight w_{ii} is increased by one for each node P_i in the cycle. From rule A, it follows that every 2-cycle will be removed. Furthermore, note that the same cycle $C \subseteq E'$ could occur multiple times. This number of occurrences is equal to $\min_{(i,j)\in C} w_{ij}$.

At the moment, we do not know why this rule should be used. However, our implementation (Section 5.9) shows that it often leads to an optimal decomposition (see Section 5.9).

Rule B: Replace edges from nodes having a single in-neighbour. If processor P_j has a single in-neighbour P_i with an edge in between with weight w_{ij}, then the weight of edges (i, l), $l \neq i$ should be increased by w_{jl} and w_{ij} should be decreased by w_{jl}, for each out-neighbour P_l $(l \neq i)$ of P_j, after which the weights w_{jl} $(l \neq i)$ should be set to 0. The quadruple (i, j, l, w_{jl}) for all processors P_i should be stored, e.g. in a table which we call the *path table*. (i, j, l, w_{jl}) means that node P_l can be reached from node P_i by walking via P_j. The number of edges (i, k) for which this holds, is equal to w_{ij}.

This rule is valid, since when node P_j can only be reached from P_i, then all out-neighbours of P_j can indirectly be reached from node P_i. An example where rule B can be used and not rule A is shown in Fig. 12.

Rule C: Replace edges from nodes having a single out-neighbour. If processor P_j has a single out-neighbour P_l with an edge in between with weight w_{jk}, then the weight of edges (i, l), $i \neq l$ should be increased by w_{ij} and w_{jl} should be decreased by w_{ij}, for each

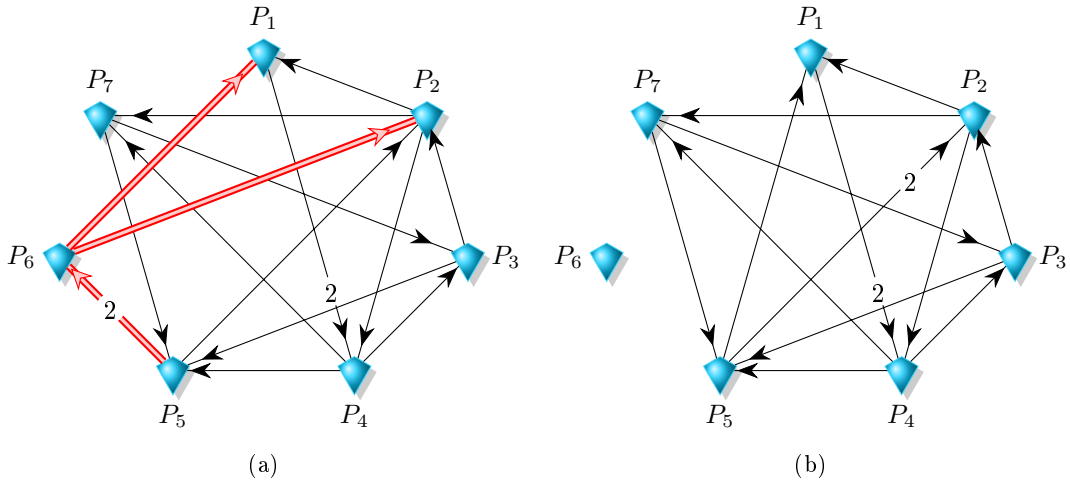


Figure 12: *This figure shows an example (a) of a move graph G' where rule B can be applied, while rule A cannot be used: Rule B can be applied on node P_6 and its adjacent edges. Then (b), rule A can be applied on cycle $\{1, 4, 5\}$, etc.*

in-neighbour P_i $(i \neq l)$ of P_j, after which the weights w_{ij} should be set to 0. The quadruple (i, j, l, w_{ij}) for all processors P_l should be stored in the path table.

It is legal to use rule C, since when node P_j is reached, then there is no other way than going to its single out-neighbour P_l.

5.7 A Function Giving an Optimal Decomposition

Rules A - C mentioned in the previous section, are presented in the form of a function (Algorithm 2) in order to find an optimal cycle decomposition of the move graph, described by the move matrix W. Rules A, B and C are set out in Algorithm 3, Algorithm 4 and Algorithm 5, respectively.

5.8 Complexity of Algorithm 2

The produced routing scheme by an algorithm like Algorithm 6 might be efficient in the number of swaps to do between the quantum PUs, although for practical reasons, the computation time to find such efficient scheme on a classical computer should be minimized too. This motivates a verification of the computational complexity of Algorithm 2, which can form a basis for routing algorithms for FCS graphs. Therefore, the computational complexity of the rules A, B and

Algorithm 2: GetDecomposition

```
   Data: Move matrix W.
   Result: A decomposition C of the move graph G' into cycles.
1  W' := W;
2  T := ∅;
3  C := ∅;
4  while W' ≠ 0 do
5  |  RuleA(W', C, T);
6  |  if W' ≠ 0 then
7  |  |  RuleB(W', T);
8  |  if W' ≠ 0 then
9  |  |  RuleC(W', T);
10 |  if W' ≠ 0 and rules A - C cannot be applied then
11 |  |  remove one of the shortest cycles;
```

C will be examined. Besides, the algorithm should use a limited amount of classical memory space to run, which is measured by the space complexity.

5.8.1 Time Complexity

We examine the time complexity of Algorithm 2, starting with rule A, after which the other rules are addressed.

Rule A. This rule starts with initializations on lines 2 - 3, which need time $\mathcal{O}(1)$.

To verify the time complexity of the rest of the algorithm, firstly, the `for`-loop on line 9 is considered. This `for`-loop is executed l times for each l-cycle. To verify the condition on line 10, time $\mathcal{O}(lk)$ is required. This gives in total time $\mathcal{O}(l^2k)$ for the `for`-loop on lines 9 - 11.

The condition on line 12 demands time $\mathcal{O}(1)$. If this condition is satisfied, the code below needs time $\mathcal{O}(l + l\gamma)$ for a cycle of length l, where $\mathcal{O}(l\gamma)$ is the time required for the function AddCycle. Since at most $\lfloor k/l \rfloor$ l-cycles are in W' and the `while`-loop is executed at most k times, the overall execution time is $\mathcal{O}((k/l)(lk^2 + l + l\gamma) + k^4) = \mathcal{O}(k^4 + k^3 + k\gamma + k)$.

Rule B. We proceed analysing rule B. The `for`-loop on line 2 is executed k times. The condition on line 3 can be verified in time $\mathcal{O}(1)$, if the number of in-neighbours is tracked during the execution of the algorithm. The inner `for`-loop requires at most m steps. This gives a total runtime of $\mathcal{O}(km) = \mathcal{O}(n)$.

Rule C. Similar to rule B, it demands time $\mathcal{O}(n)$.

Algorithm 3: Rule A

```
1  Function ruleA(W', T, C)
       Data: The temporary move matrix W' used by the algorithm using this function,
             the path table T and the collection of cycles C.
       Result: An updated version of W' on which rule A is applied where possible and
               the (extended) cycle collection C.
2      l := 1;
3      extended := 1 ;          // Flag:  Decomposition C extended (1) or not (0)
4      while extended == 1 do
5          extended := 0;
6          l := getMinCycleLen(W');
7          foreach cycle C in W' with |C| = l do
8              c := 0 ;                                              // Reset counter
9              foreach (i, j) ∈ C do
10                 if shortest path from P_j to P_i has length l − 1 and flow f_ji ≤ w_ij then
11                     c := c + 1 ;                                  // Increase counter
12             if c ≥ l − 1 then
13                 w'_ij := w'_ij − 1, ∀(i, j) ∈ C ;          // Remove cycle C from W'
14                 addCycle(C, C, T) ;                  // Build cycle decomposition C
15                 extended := 1 ;                                   // C is extended
```

So far, we know the computational complexity of rules A, B and C using the proposed algorithms. Since we do not know how frequently rules A - C cannot be applied, the exact time complexity of Algorithm 2 is not presented. However, since the number of 3-cycles in G' is at most $n/3$, the `while`-loop is used at most $n/3$ times, such that the algorithm runs in time $\mathcal{O}((n/3)(k^4 + k^3 + k\gamma + k))$.

5.8.2 Space Complexity

Another complexity measure is how much classical memory the algorithm requires. The single variables are only l, c and *extended*. These require space $\mathcal{O}(2\log k + 1)$. The other variables are matrices and sets, being the matrix W', the path table $\mathcal{T}$ and the cycle decomposition $\mathcal{C}$. W' needs spaced $\mathcal{O}(k^2 \log m)$; W' is a $k \times k$ matrix with weights $w'_{ij} = [W']_{ij} \leq m$ requiring $\mathcal{O}(logm)$ bits each. In the path table $\mathcal{T}$, each quadruple demands space $\mathcal{O}(3\log k + \log m)$. Each time rule B or C can be applied on some node P_i, at most m quadruples are added to $\mathcal{T}$. Since a trivial lower bound on the number of times rule B (C) can be applied successively is $k/3$, the path table $\mathcal{T}$ needs space $\mathcal{O}((n/3)(3\log k + \log m)) = \mathcal{O}(n\log k + n/3\log m)$. Lastly,

Algorithm 4: Rule B

```
1 Function ruleB(W', T)
    Data: The temporary move matrix W' used by the algorithm using this function,
          and the path table T.
    Result: Rule B applied on W'; an updated version of W', and the (updated) path
            table T.
2   foreach j ∈ [k] do
3     if Σ_i w_ij == 1 then                    // If P_j has exactly 1 in-neighbour
4       foreach l ∈ [k], i ≠ l ≠ j do              // For each out-neighbour P_l
5         w_il := w_il + w_jl;
6         w_ij := w_ij − w_jl;
7         w_jl := 0;
8         STORE(i, j, l, w_jl) ;                 // Add quadruple to path table T
```

Algorithm 5: Rule C

```
1 Function ruleC(W', T)
    Data: The temporary move matrix W' used by the algorithm using this function,
          and the path table T.
    Result: Rule B applied on W'; an updated version of W', and the (updated) path
            table T.
2   foreach j ∈ [k] do
3     if Σ_i w_ji == 1 then                    // If P_j has exactly 1 out-neighbour
4       foreach i ∈ [k], j ≠ i ≠ l do               // For each in-neighbour P_i
5         w_il := w_il + w_ij;
6         w_jl := w_jl − w_ij;
7         w_ij := 0;
8         STORE(i, j, l, w_ij) ;                 // Add quadruple to path table T
```

the cycle decomposition $\mathcal{C}$ requires space $\mathcal{O}(n \log k)$. Adding up these space requirements, the sum is $\mathcal{O}(2(n+1)\log k + (k^2 + n/3)\log m + 1)$.

Note that the time complexity of a function like ADDCYCLE can be reduced by using an index on the path table $\mathcal{T}$. However, this would increase the demanded memory space.

5.9 Implementation and Statistics of Algorithm 2

An algorithm executing rules A - C has been implemented using the programming language C, whereby -so far- only the number of cycles is counted; an optimal decomposition is not given. As stated in Section 4.3, the function Algorithm 1 has been implemented in C too. These implementations are available online at `https://github.com/jkeur/routeSwaps.git`.

By counting the number of cycles in the decomposition, the minimum number of swaps is determined by Eq. 13, which is the number of swaps used by an algorithm based on Algorithm 2. For such an algorithm, see Algorithm 6 in Section 5.12. In this way, the number of swaps can be compared to the simple algorithm given before, namely Algorithm 1 (Section 4.3).

Fig. 13 gives an indication of the performance in terms of the number of swaps, for an algorithm based on GETDECOMPOSITION, compared to Algorithm 1. The number of nodes per star m is fixed to 6. For $k \in \{5, 10, 20, 50\}$ stars, 500 random move graphs were generated and the optimal number of cycles in the decomposition was found in most of the cases. What 'most of the cases' implies, is discussed later. For a certain number of stars, the number of swaps was averaged over the 500 cases. This is shown by the blue line with circle marks in Fig. 13. Analogously, the same problem instances were solved using Algorithm 1. The average number of swaps for a certain value of k is indicated by the red line with dotted marks. In this figure, is clearly visible that an algorithm based on Algorithm 2 performs much better than ROUTESIMPLE (Algorithm 1).

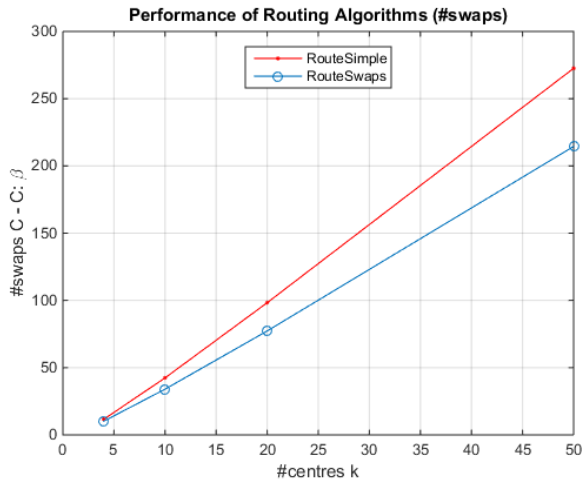


Figure 13: *A graph giving an indication of the performance in terms of the number of swaps, of an algorithm based on* GETDECOMPOSITION *(Algorithm 2), with respect to the simple algorithm* ROUTESIMPLE *(Algorithm 1). The number of nodes per star m is fixed to 6. For a certain number of stars* $k \in \{5, 10, 20, 50\}$*, 500 problem instances were generated, after which the number of swaps were counted used by both algorithms. The red line with dotted marks gives a benchmark of* ROUTESIMPLE*. The optimal number of swaps using* GETDECOMPOSITION *is indicated by the blue line with circled marks.*

Unfortunately, Algorithm 2 does not always give an optimal result for a given problem instance. One example is given in Fig. 14, where none of the rules A - C can be applied.

In such cases, one of the shortest cycles was taken (at random), after which Algorithm 2 was continued. In the given instance, this method will lead to an optimal solution. Although this is not proven, it can be examined. It is open for research how to find the best decomposition in general.

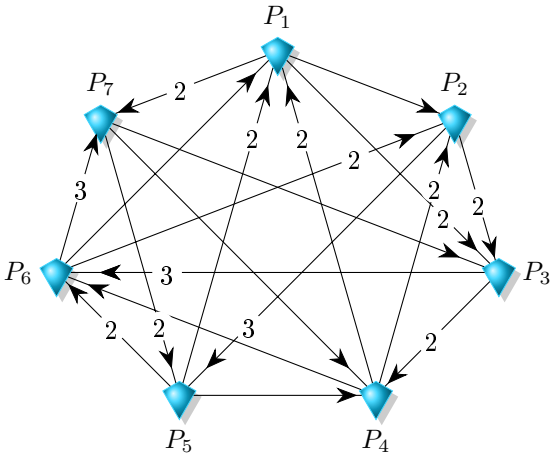


Figure 14: *This figure shows an example of a move graph G' which cannot be decomposed into cycles by using rules A - C. More research would be required to solve it.*

Optimality of Algorithm 2 has been verified using a second approach to get a cycle decomposition. This reference method was not guaranteed to give an optimal solution, however, in any case that Algorithm 2 gave a decomposition, the number of cycles was not lower than using the reference. This has been tested for thousands of random instances with $m = 6$ nodes per star and different numbers of stars $k \leq 7$.

Graphs with more than seven stars have been evaluated too. By comparing the results of Algorithm 2 and the reference algorithm, as low as 7/500 (1.4%) instances with $k = 10$ were not decomposed in an optimal way by Algorithm 2 and 22/500 (4.4%) for $k = 20$ stars. If the optimal number of cycles/swaps was not reached by Algorithm 2, then the detected deviation was only 1 swap from the reference value, which is at least closer to the optimal value. For $k = 4$ stars, the minimum number of swaps was found successfully for all instances. This is caused by the small problem size. For $k = 500$ stars, remarkably, only 2/500 (0.4%) cases were detected to be not optimal. This may be caused by the relative sparsity of in/out-neighbours. In this case, only at most 6 out of 499 nodes were in-neighbour of a particular node and at most 6/499 nodes out-neighbour. This means that there is a high chance that different cycles do not have

overlapping edges. Moreover, there is a high probability that the condition of the flow (rule A) is satisfied for some edge. Therefore, with high probability rule A can be applied, which seems to be the case. It can be concluded that rule A is worth to use, although its correctness has not been investigated/proven. All together, it can be concluded that Algorithm 2 gives nearly optimal results if not optimal.

5.10 Example Cases

To illustrate these rules, consider the example of a move graph in Fig. 15. All of the rules A - C can be applied on this instance. Thereby, it does not matter whether rule A is applied firstly, or rule B or C. This is demonstrated with some examples.

If rule A is chosen firstly, it should be applied on the unique 2-cycle $\{(3,5),(5,3)\}$. If rule B would be used, it should be applied on e.g. node P_2, which has a single in-neighbour P_1. Vice versa, node P_1 has a unique out-neighbour P_2. Therefore, rule C can be used too. Below, an example is given where an optimal cycle decomposition is built using only rule A. As stated above, rule B can be chosen before applying rule A, for which an example is given too.

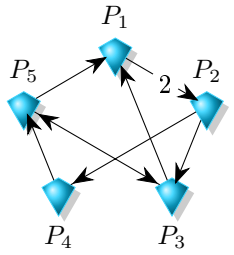


Figure 15: *An example of a move graph G' is illustrated, on which rules A - C can be applied.*

Fig. 16 shows that rule A is sufficient to get an optimal cycle decomposition for this instance (a). It is assumed that the graph is fully described by the move matrix W. As said before, the rule acts on a copy of the move matrix, which is visualized by showing the move graph corresponding to the temporary move matrix W', which is initially equal to the original move matrix W. A cycle like $\{(3,5),(5,3)\}$ is written shortly as $\{3,5\}$.

Firstly, the 2-cycle $\{3,5\}$ should be removed from W' (b); this is the only shortest cycle. At the same time, we start building the cycle decomposition $\mathcal{C}$ by adding this cycle. Then the shortest cycle is the 3-cycle $\{1,2,3\}$. All of the three edges in this cycle satisfy the condition

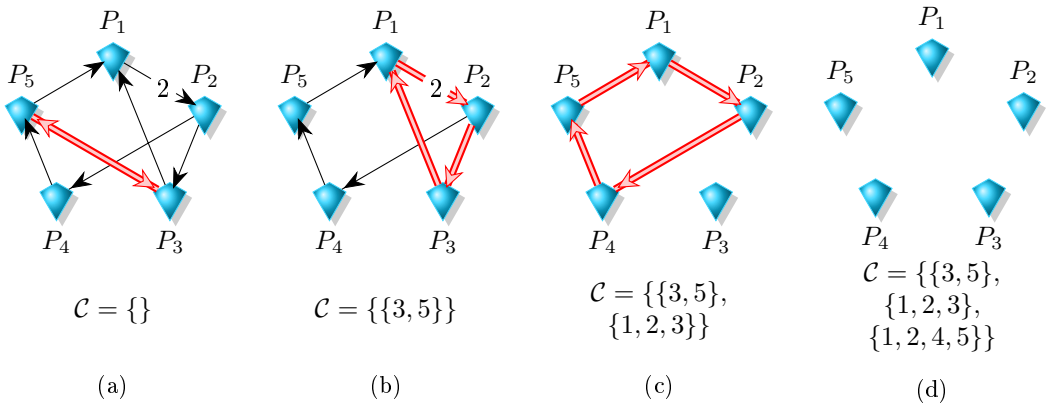


Figure 16: *(a) shows an example of a move graph G'. It is assumed that the graph is fully described by the move matrix W. Here, rule A acts on a copy of the move matrix, say W', for which the corresponding move graph is depicted in this figure for each step the algorithm takes. For a cycle $\{(3,5),(5,3)\}$, the short hand notation $\{3,5\}$ is used. In this example, rule A can be applied to the cycle $\{3,5\}$ firstly (b). Then on the cycle $\{1,2,3\}$ (c), and lastly on the remained cycle $\{1,2,4,5\}$ (d). In each step, the cycle decomposition $\mathcal{C}$ is extended accordingly.*

that there is a unique shortest path from the end of the edge to its origin. Therefore, by rule A, this 3-cycle should be removed from W' and added to $\mathcal{C}$ (c). After that a 4-cycle remains, which should be removed lastly. Then $W' = 0$ and the cycle decomposition consists of 3 cycles (d).

The next example shows how to apply rule B, see Fig. 17. Rule B is applicable to node P_2; its unique in-neighbour is node P_1. The situation will become as shown in Fig. 17 (b) and the quadruples $(1,2,3,1)$ and $(1,2,4,1)$ are stored in the path table $\mathcal{T}$. Rule B can be applied again on node P_4: the edges $(1,4)$ and $(4,5)$ will be replaced by a direct edge $(1,5)$. The quadruple $(1,4,5,1)$ will be stored. What remains are only bidirected edges between nodes P_1, P_3 and P_4.

To form a decomposition, rule A should be applied now. Say we start with the cycle $\{1,3\}$. According to the first quadruple stored in table $\mathcal{T}$, node P_3 is reached from node P_1 by going via node P_2. This means that the cycle will be extended to $\{1,2,3\}$ and then removed from W', see Fig. 17 (d). The quadruple used will be removed from $\mathcal{T}$ and the forming the decomposition is taken off with the 3-cycle. Secondly, we choose the 2-cycle $\{(3,5),(5,3)\}$. None of the stored quadruples indicate that some of these nodes are reached via an intermediate node, which means that the cycle can be removed; it occurred in the original graph as it is now (e). Lastly, the 2-cycle $\{1,5\}$ is to be removed. The second of the remained quadruples states that node P_5 is reached from node P_1 via P_4 and the first quadruple that this node P_4 is reached from node

P_1 via P_2. Therefore, the cycle becomes a 4-cycle, being $\{(1, 2, 4, 5\}$. Now, this cycle should be removed from W' together with the quadruples used from $\mathcal{T}$. The cycle finalizes the formation of an optimal decomposition; $W' = 0$.

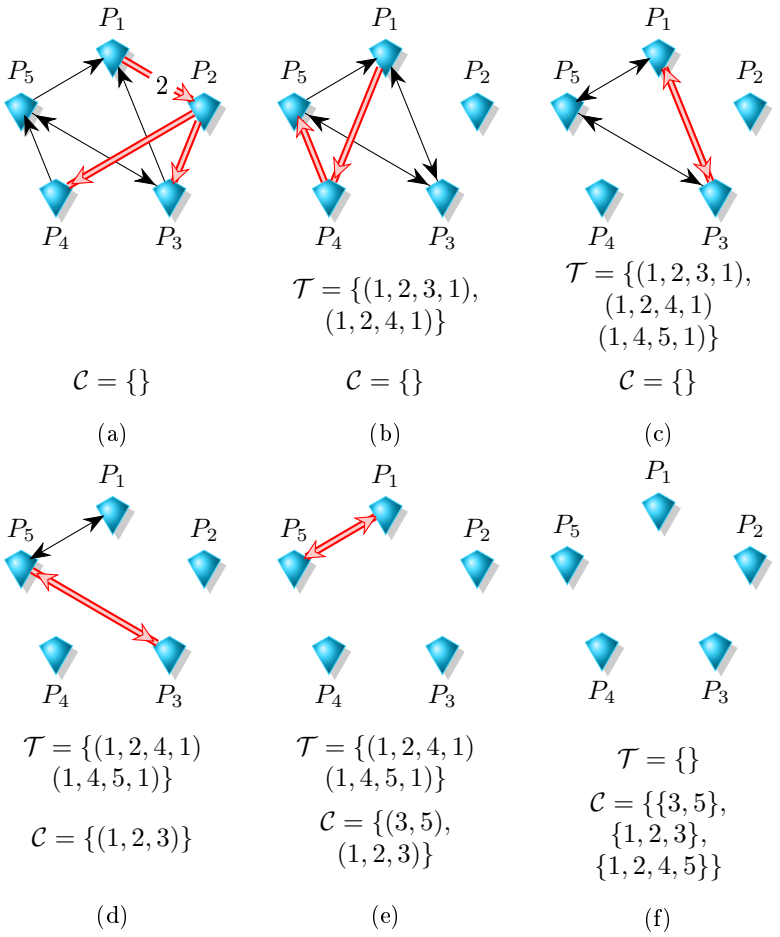


Figure 17: *(a) shows an example of a move graph G'. Rule B is used to replace the edges adjacent to node P_2; it has a single in-neighbour. The corresponding quadruples are stored in the path table $\mathcal{T}$ (b). Then rule B can be applied on node P_4 and its edges, by which one quadruple will be added to $\mathcal{T}$ (c). Since rule B cannot be applied any longer (nor rule C), rule A should be used. The 2-cycle $\{1,3\}$ is chosen firstly, which is extended to the 3-cycle $\{1,2,3\}$ by using the first quadruple. With this cycle, building the cycle decomposition begins (d). Next, the 2-cycle $\{3,5\}$ is picked, and directly added to $\mathcal{C}$; there is no quadruple indicating that an extension should be made (e). Lastly, the remained 2 quadruples indicate that the 2-cycle $\{1,5\}$ should be extended to the 4-cycle $\{1,2,3,4\}$. This cycle finalizes the formation of an optimal decomposition; $W' = 0$.*

5.11 From the Decomposition to Swaps

In the previous section, it has been demonstrated how to find an optimal cycle decomposition. In order to use such decomposition to find an optimal qubit sorting scheme, it should be known how to use it. From the decomposition, it can be derived which swaps are legal to use and which may not be used in a sorting scheme.

To see which swaps are allowed, let our attention go to a single cycle from an optimal cycle decomposition, as shown in Fig. 18. The cycle considered has weights $w_{ij} = 1$ for all edges (i, j) in the cycle. If such cycle C occurs p times ($p = \min_{i,j,(i,j)\in C} w_{ij}$) in the optimal decomposition, then the proposed theories hold for all p fully overlapping cycles.

Regarding the cycle, it can be observed in Fig. 18, that a swap (⟷) over an edge in an n-cycle reduces the cycle to an $(n-1)$-cycle and leaves one single node.

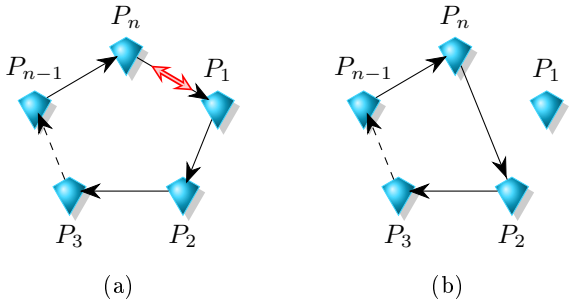


Figure 18: *A cycle containing n nodes is drawn (a). A* SWAP *operation between neighbouring nodes P_1 and P_2 would move the qubit from P_1 to its destination P_2 and the qubit from P_2 with destination P_3 is moved to P_1 (b). The length of the new cycle is $n-1$.*

In general, the following can be stated about doing a swap between any two nodes in a cycle. Recall that the distance was given in Def. 3.2.

Lemma 5.2. *A swap between nodes P_n and P_d in a single cycle having only edges with weights $w_{ij} = 1$, leaves one $(n-d)$-cycle and one d-cycle, both having only edges with weight 1, where d is the distance between the two nodes.*

Proof. This is shown in Fig. 19 (and Fig. 18). □

Doing a swap according to Lemma 5.2, the number of swaps to sort the two remained cycles is $(n-d-1)+(d-1) = n-2$. Addition of the single swap done, makes $n-1$ swaps for a n-cycle, which is exactly Cayleys distance (12) for a cyclic permutation. Therefore, the following

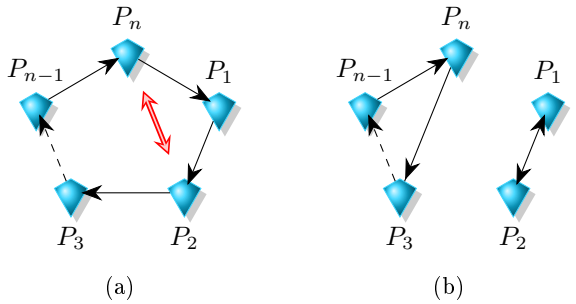


Figure 19: *A single cycle containing n nodes and edge weights 1 is shown (a). A* SWAP *operation between nodes P_1 and P_3 would move the qubit from P_1 with destination P_2 to P_3 and the qubit from P_3 with destination P_4 ends in P_1 (b). Two smaller cycles are formed; one with length $(n-d)$ and one with length d.*

Lemma 5.3 is true. This lemma is linked to what is called the *individual routability* of a cycle C [12]. However, in the move graph G', more freedom exists to swap two qubits, since swaps can be done between any pair of nodes in the move graph and not only over edges $(i,j) \in C$.

Lemma 5.3. *Let qubits $v_{i,j}$ with destinations $\pi(v_i)$, $\pi(v_j)$, be moved from nodes $P_{i,j}$. Then the optimal number of swaps* (13) *for a given optimal cycle decomposition $\mathcal{C}$ will be reached, when a swap over edge (i,j) is done such that firstly, nodes P_i and P_j are in one and the same cycle $C \in \mathcal{C}$, and secondly, if edges $(i,\pi(v_i))$ and $(j,\pi(v_j))$ are in cycle C.*

Proof. When a swap over edge (i,j) is done in some cycle $C \in \mathcal{C}$ and if edges $(i,\pi(v_i))$ and $(j,\pi(v_j))$ are in this cycle C too, then the cycle divides according to Lemma 5.2. By doing so, the total number of required swaps does not change, such that the optimal number of swaps (13) will be reached. □

So far, a single cycle is considered. Two cycles can be considered together too, as shown in [12, p. 2-3]: Two disjoint cycles can be *mutually routable* in 2 steps, if and only if they are of same length. In our cycle decomposition, two cycles can overlap each other. In certain situations, overlapping cycles can be routed in 2 steps too, see Fig. 20. However, the conditions under which this is possible, are not investigated and is open for research. From this section, it can be concluded how to swap qubits such that the minimum number of swaps is used, by investigation of an optimal cycle decomposition.

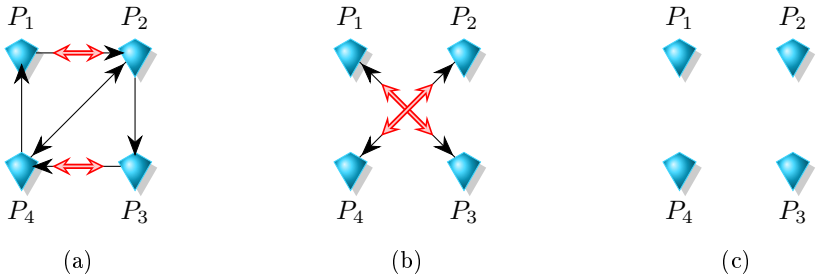


Figure 20: *The 2-cycle* $\{2,4\}$ *and 4-cycle* $\{1,2,3,4\}$ *are partly overlapping (a). A* SWAP *operation over edges* $(1,2)$ *and* $(3,4)$ *results in the situation in (b), only if the qubit used from node* P_2 *has destination* P_3 *and the qubit from* P_4 *has destination* P_1*. Two additional swaps are required to sort the remaining qubits (c). This show that there are cases where 2 partly overlapping cycles can be routed in 2 steps.*

5.12 An Algorithm Solving Problem 2 Minimizing the Number of Swaps

So far, it is described how to create a cycle decomposition with the maximum number of cycles (in many cases). This means that a routing algorithm based on the function GETDECOMPOSITION (Algorithm 2) can give a routing scheme which is optimal in the number of swaps (13). It has been investigated how to use an optimal decomposition to derive which swaps are legal. With this, an algorithm can be designed.

5.12.1 A Routing Algorithm using Swaps

A simple algorithm satisfying the condition in Lemma 5.3 is Algorithm 6, which minimizes the number of swaps. In Section 5.9, the performance of this algorithm is compared to Algorithm 1.

Algorithm 6: RouteSwaps

Data: Move matrix W.
Result: A list L containing the swaps done.

```
C := GetDecomposition(W);
foreach (i, j) ∈ C do
    swap(i, j);
    add2list(i, j);
    deleteEdge(C, (i, j));
    deleteEdge(C, (j, π(v_j)));
    addEdge(C, (i, π(v_j)));
```

In this algorithm, it is assumed that in practice, the qubit swaps is executed by some background process. With a command like SWAP(i, j), the process knows which qubit to move from node P_i to P_j and vice versa. Thereby, the process uses the cycle decomposition $\mathcal{C}$. Upon the command, the process should retrieve the correct qubit from its current position and send it from subgraph P_i to subgraph P_j containing its destination. Once the qubit is there, the background process should place the qubit in the correct node within that subgraph.

5.12.2 Time Complexity of Algorithm 6

The time complexity of Algorithm 6 can be derived as follows. The function GETDECOMPOSITION requires time $\mathcal{O}((n/3)(k^4+k^3+k\gamma+k))$ (see Section 5.8.1). Since the maximal number of swaps to do is β (Eq. 13), the `for`-loop is executed $\mathcal{O}(n)$ times. The commands on lines 3 - 5 and 7 require time $\mathcal{O}(3)$. The increase of the execution time due to calling the function DELETEEDGE is $\mathcal{O}(2n \log n)$. So the total runtime of Algorithm 6 is $\mathcal{O}((n/3)(k^4+k^3+k\gamma+k)+3n+2n \log n)$.

5.12.3 Space Complexity of Algorithm 6

Now the required classical memory space is analysed. Algorithm 6 starts calling GETDECOMPOSITION, which needs space $\mathcal{O}(2(n+1)\log k + (k^2+n/3)\log m + 1)$. Then a list L is built, which demands space $\mathcal{O}(n)$. This sums up to $\mathcal{O}(2(n+1)\log k + (k^2+n/3)\log m + 1 + n)$.

5.13 Solving Problem 2 Minimizing the Total Number of Swaps

Given Problem 1 with $a, b > 0$ and a move graph $G' = (P, E')$, the total number of swaps can be optimized by considering the second term of the edge weights $w_{ij}, 1$ -indicating the number of qubits ending in the centre of P_j, which is always at most 1 in the FCS graph. If the second term is 1 for edge (i, j), then it should be ensured that all qubits having a destination in P_j have arrived there, except one qubit coming from node P_i. When this is the case, the one qubit to be moved outwards from node P_j should be moved to node P_i and finally, the qubit from node P_i having the centre of P_j as destination should be teleported. This is possible by doing the teleportation procedure in two steps (see Fig. 21). Firstly, entanglement is created between both centres (a). Then, the entangled qubit in processor P_1 is swapped with the spare qubit

(b). Then qubit 2 is teleported to a leaf in P_1, after which entanglement is created between the centres for the second time. (c) Lastly, qubit 1 can be teleported to the centre of processor P_2 (d).

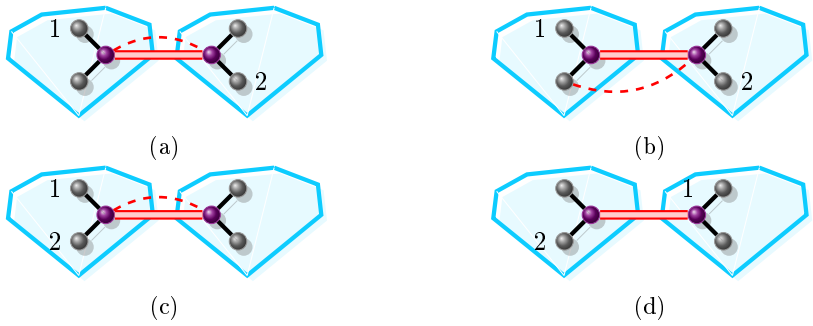


Figure 21: *A* SWAP *operation between distant qubits can be performed in two time steps. This might be relevant in the case that a qubit from the one node P_1 should end in the centre of P_2. Firstly, entanglement is created between both centres (a). Secondly, the entangled qubit in P_1 is swapped with one of the spare qubits in the processor (b). Then qubit 2 is teleported to a leaf in P_1, after which entanglement is created between the centres for the second time. (c) Lastly, qubit 1 can be teleported to the centre of processor P_2 (d).*

6 Routing Qubits via Swaps with the Minimum Number of Steps

It has been set out what the minimum number of swaps (13) is and how to create an algorithm reaching this number, using an optimal decomposition of the move graph G'. Minimizing the number of steps has more practical significance than optimizing the number of swaps, since the number of steps determine the execution time of a quantum circuit. The less steps used, the faster quantum circuits can be executed. Recall that only the swaps between stars are counted ($a = 0$).

6.1 The Optimal Number of Steps

On general graphs, finding a sequence of swaps such that the number of steps is minimized is an NP-complete problem [12]. At least some upper bound on the number of steps can be given for this problem using the move graph. In order to do so, the routing number of the move graph G' is considered. "The *routing number* $rt(G)$ of a connected graph G is the minimum integer r so that every permutation of vertices can be routed in r steps by swapping the ends of disjoint edges." [15]

As stated in e.g. [1] and [16], the routing number of a complete graph K_n is

$$rt(K_n) = 2, \quad n \geq 3 \tag{14}$$

Therefore, the following is conjectured to hold for the routing number of the move graph.

Conjecture 6.1. *The routing number of the move graph $G'_{k,m}$ having k nodes, where each node represents m nodes from a star in the original graph G, is equal to*

$$rt(G_{k,m}) = 2m, \quad k \geq 3 \tag{15}$$

where only full swaps between stars are counted.

The move graph $G'_{k,m}$ is a special kind of a complete K_k graph, where each node has m qubits. Therefore, the move graph can be regarded as if it has m complete K_k graphs. As any

permutation in a K_k graph can be routed in at most 2 steps, the routing number $rt(G_{k,m})$ of the move graph would be $2m$.

In our case, the number of steps can be bounded further by considering a given permutation π.

Conjecture 6.2. *Given some permutation $\pi : \boldsymbol{x} \to \boldsymbol{y}$ in a move graph G', an upper bound on the routing time satisfies*

$$rt(G, \pi) \leq 2 \max_{i \in [k]} \mu_i^+ \tag{16}$$

where μ_i^+ is the number of qubits to move outwards from node P_i.

This follows from the observation that the move graph G' can be regarded as a graph having $\max_{i \in [k]} \mu_i^+$ complete K_k graphs (μ_i^+ is the number of qubits to move outwards from node P_i).

A trivial lower bound on the required number of steps $d = rt(G, \pi)$ for a given permutation $\pi : \mathbf{x} \to \mathbf{y}$ is as follows.

Lemma 6.1. *The routing time $rt(G, \pi)$ for a given permutation π satisfies*

$$rt(G, \pi) \geq \left\lceil \frac{\beta}{\lfloor k/2 \rfloor} \right\rceil \tag{17}$$

Proof. Per step, no more than $\lfloor k/2 \rfloor$ swaps can be done. Since β swaps are required, (17) follows. □

Another known, trivial bound is

$$rt(G, \pi) \geq \max_{i \in [k]} \mu_i^+ \geq \max_{i \in [k]} \max(|\mathcal{N}_i^+|, |\mathcal{N}_i^-|) \tag{18}$$

where $|\mathcal{N}_i^+|$ is the number of outgoing edges from node P_i and $|\mathcal{N}_i^+|$ the number of incoming edges. The given equations are bounds, so they do not show how to reach the optimum.

6.2 Solving Problem 2 Minimizing the Number of Steps

Algorithm 6 minimizes the number of swaps, which does not imply that the number of steps is minimized. Although finding a sequence of swaps minimizing the number of steps could be

hard, some guidelines can be given to find an optimal routing scheme for the FCS graph G, by considering the move graph G'.

Since the number of steps is dependent on the maximum number of qubits to move from/to a certain node, it is good to make sure in the current step, that most of the nodes P_j satisfying $\mu_j^+ = \max_{i\in[k]} \mu_i^+$ are used in a swap over edge (l, j) for some P_l, preferably satisfying $\mu_l^+ < \max_{i\in[k]} \mu_i^+$ -thus not over edge (j, l). When these nodes P_j are used in the current step and if there are nodes unused while they can be used, then advantageously, a swap should be done over an edge (l, j) analogous to the way described above. Realise that the value for $\max_{i\in[k]} \mu_i^+$ would be lower for the remained nodes.

If there are multiple edges (l, j) to choose for some node P_j under the aforementioned conditions, then preferably, an edge is chosen with node P_l having the lowest value for μ_l^+ among the nodes P_l to choose from.

7 Routing Qubits via Moves in Fully Connected Star Graphs

As stated in Section 5, Problem 1 (Section 4.2) can be restricted to have either unique or non-unique qubits. The first case is handled in Section 5. Here the latter case is discussed.

In this case, several qubits in the FCS graph are modelled the same, which we call *spare qubits*. The state of these qubits may be destroyed. Spare qubits are used for communication purposes between PUs via teleportation. Besides full swaps, the spare qubits allow, half swaps between PUs, called moves. A *move* over an edge (i, j) will bring a qubit from node P_i to P_j. So the same Problem 1 is addressed, be it in different context than in Section 5:

Problem 3. This problem is equivalent to Problem 1, where the costs for swaps within PUs are zero ($a = 0$) and the qubits v_i carried by node i in the FCS graph $G = (V, E)$ are unique and $v_i > 0$, except for some spare qubits v_j modelled as $v_j = 0$.

Problem 1 is repeated below, where $b/2$ are the costs of a move.

$$\min \quad a \sum_{(i,j)\in E\setminus E_c} s(l,i,j) + b/2 \sum_{(i,j)\in E_c} s(l,i,j)[\ \mathrm{step}(v_i) + \ \mathrm{step}(v_j)] + cd \tag{19}$$

s.t.

$$\mathbf{y} = \left(\prod_{l=1}^{d} S_l\right)\mathbf{x} \qquad \mathbf{x}, \mathbf{y} \in \mathbb{Z}^n_{\geq 0}$$

$$S_l = \prod_{(i,j)\in E} s(l,i,j)\Pi(i,j) \qquad \forall l \in [d]$$

$$\sum_{j\in V} s(l,i,j) \leq 1 \qquad \forall l \in [d], \forall i \in [n]$$

$$s(l,i,j) + s(l+1,i,j) \leq 1 \qquad \forall l \in [d-1], \forall (i,j) \in E$$

$$s(l,i,j) \in \{0,1\} \qquad \forall l \in [d], \forall (i,j) \in E$$

7.1 Qubit Moves in Practice

The case that the qubits are non-unique might be compared to a collection of interconnected NV-centres (processors), each having at least one spare qubit. Often, this spare qubit might be

the NV-centre, which is used for communication. Besides one spare qubit for communication, each star may have one additional spare qubit in a leaf. With this, swaps as shown in Fig. 4 can be performed between any pair of stars. As mentioned before, half swaps can be used too, called *moves*.

How a qubit move can be done in practice, is shown in Fig. 22. There, a qubit move is illustrated from star P_1 to P_2. Star P_1 should have at least one spare qubit and the other star two, in order to be able to teleport a qubit from P_2 to another processor. This follows from the teleportation procedure (Section 3.1). Firstly, a spare qubit in star P_1 is entangled with a spare qubit in star P_2. Having the entangled qubit in star P_1 near the data qubit to move, the data qubit can be teleported to star P_2. In the end, star P_1 has one spare qubit more and star P_2 one less than in the initial state.

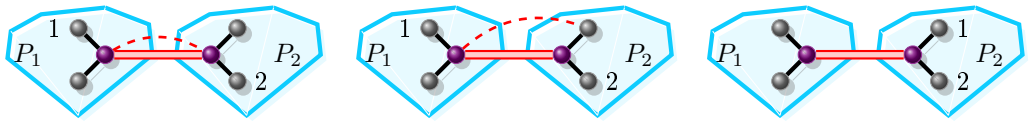


Figure 22: *Having at least one spare qubit in star P_1 and at least 2 in star P_2, a move with a qubit from star P_1 to a distant star P_2 can be done in several steps. Firstly, entanglement is created between both centres (a). Secondly, the entangled qubit in star P_2 is swapped with a spare qubit if the qubit should end in a leaf, otherwise the swap is not necessary (b). Finally, the qubit can be teleported to the star P_2 (c).*

Knowing how to move a qubit, this can be applied to the case that data qubits should be moved around in a cycle. Fig. 23 shows an example. Firstly, entanglement is created between the stars analogously to Fig. 4. Then each qubit is teleported to its destination. This implies the following.

Lemma 7.1. *Each cycle in move graph G', where each node involved has at least 2 spare qubits in both the initial and final state, can be solved within one step.*

Proof. Entanglement can be created as illustrated in Fig. 23, after which the qubits can be teleported. □

Another implication when moves are used, is that multiple qubits can be moved to the same processor in one and the same step. Fig. 24 illustrates the movement of multiple data qubits to the same processor. For each data qubit to move to processor P_2, one entangled pair should be

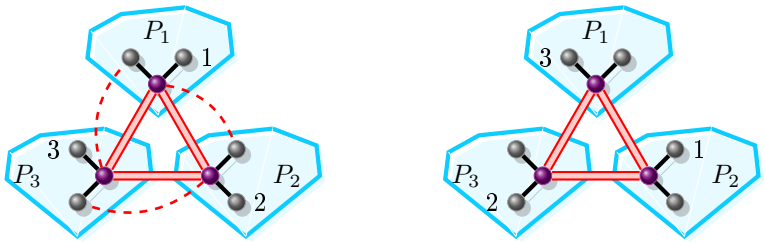


Figure 23: *In this example, the data qubits 1, 2 and 3 have to be moved around in a cyclic way, clockwise. Data qubit 1 should be moved to star P_2, qubit 2 to P_3 and qubit 3 to P_1. Each star has at least 2 spare qubits. To move qubits in a cyclic way like this, multiple steps are required. Firstly, to send qubit 1 to star P_2, entanglement is created between the centre of P_1 and a leaf in P_2. Similarly, this should be done for the other qubits (a). Secondly, the qubit are teleported to their destinations (b).*

created which is shared by the processor having the data qubit and by P_2. This implies that in one step, at most f_i qubits can be teleported to a processor P_i having f_i spare qubits. In the example in Fig. 24, the maximum number of qubits to teleport to P_2 is 3, since it has 3 spare qubits.

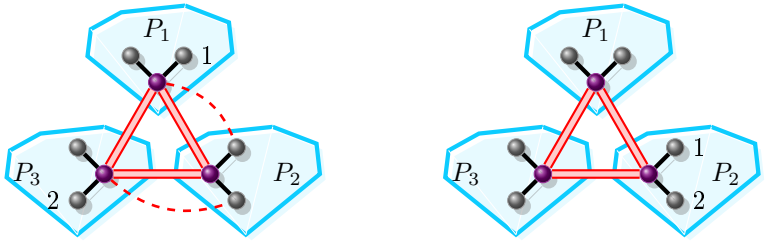


Figure 24: *Here, both data qubits 1 and 2 have to be moved to processor P_2. Processors P_1 and P_2 have at least 1 spare qubit and P_2 at least 2. To move the qubits to their destination, firstly, entanglement is created between the centre of P_1 and a leaf containing a spare qubit in P_2. Besides, entanglement should be created between the centre of processor P_3 and another leaf having a spare qubit in P_2 (a). After that, the qubits should be teleported to their destinations (b).*

Now, assume that after teleportation, the state of all qubits in P_2 in Fig. 24 should be preserved and at least one of these qubits has to be moved outwards in a second step. This is only possible if there is a spare qubit present in node P_2. Therefore, in this case, the maximum number of qubits δ_2 which could be moved to P_2 in step 1, would be 2 instead of 3.

7.2 The Optimal Number of Moves Between Centres

Knowing how qubit moves are executed in practice, the minimum number of moves can easily be derived. Say that the qubits which are not spare qubits are *data qubits*.

Theorem 7.1. *Given an instance of Problem 3, where in total μ data qubits should be moved around, this instance can be solved by using M moves, being*

$$M = \mu \tag{20}$$

Proof. Having μ qubits to move around, μ entangled pairs should be created, such that the qubits can be routed over direct paths within $M = \mu$ moves. □

7.3 The Optimal Number of Steps: 2 Spare Qubits per Star

As seen in the previous section, one could minimize the number of moves easily. A more interesting question is how to minimize the number of steps. Now, it is assumed that each node contains 2 spare qubits in both the initial and final state. This allows to move the data qubits in a cyclic way in one step, as shown in Fig. 23. This notion can be used to find the optimal number of steps.

Conjecture 7.1. *Given Problem 3 with move graph G' containing nodes P_i, $i \in [k]$, where each node has 2 spare qubits in both the initial and final state, there exists a cycle decomposition $\mathcal{C}$ containing a (sub)set of disjoint cycles, which covers at least all nodes P_i satisfying $\mu_i^+ = \max_{j\in[k]} \mu_j^+$, where μ_j^+ is the number of qubits to move outwards from node P_j.*

Graph G' can be seen as a collection of $\max_{j\in[k]} \mu_j^+$ complete graphs, each having edges (i, j) with weight $w_{ij} \in \{0, 1\}$. Therefore, there must exist a set of cycles covering all nodes P_i satisfying $\mu_i^+ = \max_{j\in[k]} \mu_j^+$. Assuming Conjecture 7.1, the minimum number of steps follows.

Theorem 7.2. *Assuming Conjecture 7.1, Problem 3 with move graph G' containing nodes $P_i, i \in [n]$, where each node has two spare qubits in both the initial and final state, can be solved within d_M steps, being*

$$d_M(W') = \max_{i\in[k]} \mu_i^+ \tag{21}$$

Proof. Let the number of nodes P_j satisfying $\mu_j^+ = \max_{i\in[k]} \mu_i^+$ be p. By assumption of Conjecture 7.1, there exists a set of disjoint cycles $\mathcal{C}_1$ covering the p nodes. It has been shown that a permutation of disjoint cycles can be sorted in 1 step. By doing that, the cycles $\mathcal{C}_1$ will be removed from the move graph G' and the weights w_{ij} are decreased by one for all edges (i, j) in the cycles. This implies that μ_i^+ is decreased by one too, for each node P_i involved in the cycles in $\mathcal{C}_1$, including the p nodes.

The procedure above can be applied repeatedly, until all weights are zero such that no node has a qubit in a wrong position. Then it follows the procedure is applied $\max_{i\in[k]} \mu_i^+$ times. □

7.4 A Function Giving a Node Cover

In order to reach both the optimal number of moves (20) and steps (21) (Problem 3), given that each PU has at least 2 spare qubits, a proper node cover should be found in each time step, according to Conjecture 7.1. Some tools are proposed below. These rules act on a copy W' of move matrix W. A candidate procedure is given in Algorithm 7.

Rule 1: Add 2-cycles to $\mathcal{C}$ containing a node with a single neighbour. Add the 2-cycle $\{(i, j), (j, i)\}$ to C_t satisfying $w_{ij}, w_{ji}, > 0$ if node P_i has exactly one neighbour P_j which is not used in $\mathcal{C}_t$ so far.

Rule 2: Add edges to $\mathcal{C}$ with a node having a single in-neighbour. Add edge $(i, j), w_{ij} > 0$ to $\mathcal{C}_t$ if node P_j has exactly one in-neighbour P_i given that $(i, q) \notin \mathcal{C}_t, \forall q \in [k]$.

Rule 3: Add edges to $\mathcal{C}$ with a node having a single out-neighbour. Add edge $(i, j), w_{ij} > 0$ to $\mathcal{C}_t$ if node P_i has exactly one out-neighbour P_j given that $(p, j) \notin \mathcal{C}_t, \forall p \in [k]$.

Rule 4: Add an edge to $\mathcal{C}$ beginning at a node used in $\mathcal{C}_t$. Add edge $(i, j), w_{ij} > 0$ such that $(p, i) \in C_t, (i, q) \notin C_t, \forall q \in [k]$ and $|\{P_j || \mathcal{N}_j^+| = \max_{p\in[k]} |\mathcal{N}_p^+|\}| = 1$.

Rule 5: Add an edge to $\mathcal{C}$ ending at a node used in $\mathcal{C}_t$. Use edge $(i, j), w_{ij} > 0$ such that $(j, q) \in C_t, (p, j) \notin C_t, \forall p \in [k]$ and $|\{P_i || \mathcal{N}_i^+| = \max_{q\in[k]} |\mathcal{N}_q^+|\}| = 1$.

Rule 6: Add randomly an unused edge to $\mathcal{C}$. Given that none of the rules 1 - 5 can be applied, randomly choose an edge $(i, j), w_{i,j} > 0$, under the following conditions. Firstly, no outgoing edge from node P_i, nor another incoming edge in node P_j may be used so far. Secondly, if some edge (p, q) is used and no outgoing edge from node P_q, then an edge $(q, j), w_{qj} > 0$ should be chosen.

Algorithm 7: Get a Node Cover

```
1  Function getNodeCover
       Data: Move matrix W.
       Result: A set C of disjoint cycles covering at least the nodes with the maximum
               number of qubits to move outwards.
2      P := {P_j | μ_j^+ = max_{i∈[k]} μ_i^+};
3      G_P := setInducedGraph(W, P) ;                    // Set induced subgraph G'[P]
4      while not all nodes P_j ∈ P are used in C_t or C_t is no collection of cycles do
5          rule1(W, G_P, C);
6          while rule 2 or rule 3 can be applied do
7              rule2(W, G_P, C);
8              rule3(W, G_P, C);
9          if rule 4 or rule 5 can be applied then
10             rule4(W, G_P, C);
11             rule5(W, G_P, C);
12             continue;
13         rule6(W, G_P, C);
```

7.5 Complexity of Algorithm 7

It is important that the function proposed in the previous section can be executed efficiently on a classical computer. Therefore, both the time and space complexity of Algorithm 7 are addressed.

7.5.1 Time Complexity

Regarding the time complexity of Algorithm 7, it starts by setting the nodes to be covered (line 2), which costs time $\mathcal{O}(k)$. Setting the induced subgraph needs time $\mathcal{O}(k^2)$. Now, the rules are examined. Rule 1 costs time $\mathcal{O}(k^2)$. Rules 2 and 3 require time $\mathcal{O}(k^2)$. They are used at most k times successively. Time $\mathcal{O}(n)$ is demanded for rules 4 and 5. Rule 6 needs time $\mathcal{O}(n)$ too. The `while`-loop is executed at most k times. For simplicity, it is assumed that each rule requires time $\mathcal{O}(k^2)$ and that all rules together are executed $\mathcal{O}(k)$ times. Then this algorithm has a total runtime in the order $\mathcal{O}(k^3 + k^2 + k)$.

As in the previous algorithm, speed-ups are possible if more information is stored. However, we preferred the simplicity of the presented algorithms over their execution time.

7.5.2 Space Complexity

The required working memory depends mainly on the set of nodes to be covered $\mathcal{P}$, the move matrix W and the induced subgraph $G_P = G'[\mathcal{P}]$. The first requires space $\mathcal{O}(k)$ and the latter two graphs require space $\mathcal{O}(k^2)$. This give a total space of order $\mathcal{O}(2k^2 + k)$.

7.6 An Algorithm Solving Problem 3: 2 Spare Qubits per Star

Being able to get a node cover according to Conjecture 7.1, an algorithm can be designed minimizing the number of steps required. An sample algorithm solving Problem 3 is Algorithm 8. In each step t, qubits are moved over edges $\mathcal{C}_t \subseteq E'$, and the vector $\mathbf{v}$, representing the qubit distribution, is updated accordingly. The set $\mathcal{C}_t$ is a collection of disjoint cycles.

Algorithm 8: routeMoves

Data: The initial placement $\mathbf{x}$ and desired placement $\mathbf{y}$ of the qubits in the FCS graph $G = (V, E)$ and the number of stars k.
Result: A list L containing an optimal sequence of moves.

```
1 W := buildMoveMatrix(x, y, k);
2 t := 1 ;                                  // Start in the first time step
3 v := x ;                                  // Use a copy of the initial state
4 while W ≠ 0 do
5 |   C_t := getNodeCover(W);
6 |   useCycles(C_t, v, k);
7 |   t := t + 1;
```

Conjecture 7.2. *Given an FCS graph $G = (V, E)$ with initial qubit placement $\boldsymbol{x}$ and desired placement $\boldsymbol{y}$, Algorithm 8 solves Problem 3 such that both the number of swaps as well as the number of steps are minimized.*

For the cases (about 20) solved by hand, the conjecture was satisfied. Although not all rules have been implemented so far, no case has been imagined in which rules 1 - 6 would not lead to a proper node cover, by which an optimal solution can be obtained.

Assuming Conjecture 7.2, it can be concluded that given a working function getNodeCover, the qubits in a FCS graph can be routed efficiently using Algorithm 8. The routing scheme generated uses the minimum number of moves (Eq. 20) as well as the optimal number of

steps (Eq. 21), being only the maximum number of qubits to move outwards from a particular node, which is at most m. Both the time and space requirements are polynomial in k for k stars (see the previous section). Even the required number of spare qubits in the quantum circuit is as low as 2 per processing unit.

7.7 Complexity Algorithm 8

Knowing how to plan qubit moves with the minimum number of steps, the running time of Algorithm 8 remains to be investigated. Besides, the classical space requirements are examined.

7.7.1 Time Complexity

Algorithm routeMoves starts with creating the move matrix W, given the current and desired qubit placements in the FCS graph. This costs time $\mathcal{O}(n^2)$. Setting the single variable t costs time $\mathcal{O}(1)$. A time in the order $\mathcal{O}(n)$ is demanded to create a copy of the initial qubit distribution $\mathbf{x}$ (line 3). Then a node cover is created, requiring time $\mathcal{O}(k^3 + k^2 + k)$, see Section 7.5. The function useCycles needs time $\mathcal{O}(n)$; there are at most k edges in $\mathcal{C}_t$, but since the qubit distribution $\mathbf{v}$ is updated, time $\mathcal{O}(n)$ is required. The last statement needs time $\mathcal{O}(1)$. The `while`-loop is executed exactly d_M times (Eq. 21), which is at most m. Therefore, Algorithm 8 runs in time $\mathcal{O}(n^2 + n + 1 + m(k^3 + k^2 + k + n + 1))$.

7.7.2 Space Complexity

The working memory required for this algorithm needs to store the variables t and k, vectors $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{v}$ and the move matrix W. This costs space in the order $\mathcal{O}(k^2 + 3n + 2)$.

7.7.3 Example of the Application of Algorithm 8

As example to apply Algorithm 8, consider the move graph in Fig. 10. Let $d_{M,t}(G')$ be the number of steps to do after time step t. Then initially, the maximum number of qubits to move outwards from a node, is equal to $d_{M,0}(G') = 3$, which is the case for only node P_2. This implies by Theorem 7.2 that 3 steps are required. Furthermore, Conjecture 7.1 states that there is a disjoint set of cycles in G' covering all nodes having 3 qubits to move outwards. Let's take the cycle $\{(2, 4), (4, 2)\}$ firstly. By sorting some qubits via this cycle in step 1, the value for $d_{M,1}(G')$

becomes 2; both nodes P_2 and P_3 are part of 2 cycles. Now, we sort 3 qubits via the cycle $\{(1,2),(2,3),(3,1)\}$, covering at least the nodes P_2 and P_3. Being 2 time steps further, $d_{M,2}(G')$ is 1 for all nodes in the move graph G'. Lastly, the remained cycle $\{(1,2),(2,3),(3,4),(4,1)\}$ will be used to sort the four qubits which are not in place, which leaves $W = 0$; the qubits are sorted and the process ends.

7.8 The Optimal Number of Steps: At Least 1 Spare Qubit per Star

Suppose that each processor does not necessarily have 2 spare qubits. Then the number of steps does not always satisfy Eq. 21; if a processor P_i has only 1 spare qubit and if at least one data qubit in P_i has to be moved outwards, then no other qubit may be moved to P_i, otherwise no data qubit can be moved outwards.

Proposition 7.1. *Problem 3 can be solved within one step using moves, if and only if the following requirements are met. Firstly, each processor P_i has at most 1 data qubit to move outwards ($\mu_i^+ \leq 1$). Secondly, each processor which has 1 data qubit to move out and gets 1 spare qubit instead, has at least 2 spare qubits. Thirdly, a processor which receives μ_i^- data qubits has at least $\mu_i^- + 1$ spare qubits.*

Proof. Let the number of nodes containing a data qubit to move be t. Then these t nodes form a regular graph having exactly one incoming and one outgoing edge. Therefore, the cycle decomposition of graph G' is a set of disjoint cycles. As disjoint cycles can be solved in parallel, this problem can be solved within one step (Lemma 7.1). □

7.8.1 An Algorithm Solving Problem 3: At Least 1 Spare Qubit per Star

Solving Problem 3 with at least 1 spare qubit per processing unit is much harder than when there are 2 spare qubits available in each PU. Although no algorithm has been found to give a routing scheme which is optimal in the number of steps, some solving tools are listed below.

For $\mu_i^- = \sum_j w_{ji}$ data qubits to move to processor P_i and having μ_i^+ data qubits to move to other nodes, let processor P_i have f_i spare qubits in the current state.

Rule I. Use as most edges (i,j) as possible, satisfying $f_i = 1$ and $f_j > 1$.

Rule II. Use as most edges (i, j) as possible, satisfying $\mu_i^+ > 0$.

7.9 Application of Algorithm 8 to Other Graphs

In general, Algorithm 8 can be applied to other graphs (quantum processor architectures) than the FCS graph as long as the following conditions are satisfied.

1. The graph considered should be a connected graph.
2. The graph should be divisible into k smaller subgraphs $P_1, \ldots, P_k$.
3. A qubit in subgraph P_i ($i \in [k]$) may only be moved to another subgraph P_j ($i \in [k]$) if edge (i, j) exists in the move graph.
4. Given a transition of the qubits $\pi : \mathbf{x} \to \mathbf{y}$, for each subgraph P_i, the number of incoming qubits should be the same as the number of outgoing qubits ($\mu_i^+ = \mu_i^-, \forall i \in [k]$).
5. It is assumed that in practice, a qubit move is executed by some background process. With a command like MOVE(i, j), the process knows which qubit to move from node P_i to P_j. Upon the command, the process should retrieve the correct qubit from its current position and send it from subgraph P_i to subgraph P_j containing its destination. Once the qubit is there, the background process should place the qubit in the correct node within that subgraph.

Note that requirement 4. implies that each subgraph should have at least one incoming and one outgoing edge (without restriction on their weights). Besides, the number of moves between the subgraphs are minimized by this algorithm, however, the number of steps will only be minimized in the case that at most one qubit can be moved outwards from some node in a particular time step.

8 Conclusions & Open Questions

The problem of routing qubit movements in a fully connected star graph G with qubits is discussed. In other words, the planning of qubit movements in a special quantum architecture is examined. The graph is abstracted to a move graph, by which our attention was directed to the swaps between centres. Several special cases of the problem were considered. Firstly, the qubits are moved around I. via swaps or II. via moves (half swaps). Furthermore, we can minimize 1. the number of swaps and 2. the number of time steps.

In case I.1, the move graph G' should be decomposed into as many cycles as possible. Using such a decomposition in planning the qubit swaps, results in the minimization of the number of full swaps. An algorithm is proposed which finds a (nearly) optimal decomposition. This has been implemented. From the results shown, it can be concluded that this algorithm performs better than the simple routing algorithm that was set out, based on the existing method Selection sort. Moreover, the total number of swaps can be optimized by planning the execution of swaps with qubits having a centre destination, after other swaps.

The problem in case I.2 is NP-complete on general graphs. Several solving rules are proposed by which many instances can be solved by an efficient algorithm which produces a routing scheme that is optimized in the number of steps. However, several specific cases will not be solved.

In case II where each star has at least 2 spare qubits, the optimal number of moves can be achieved easily; the order of moving the qubits does not matter. The number of steps required is conjectured, being equivalent to the maximum number of qubits which should be moved outwards from one of the stars/processing units. An efficient routing algorithm is conjectured to solve this problem. However, more thorough research would be required to validate its correctness and universality. Nevertheless, the theoretical basics are given. It is known what to achieve, though the question remains how the requirements can be met by preferably, an efficient algorithm.

Algorithm 8 is directly applicable to other graphs than fully connected star graphs, under certain conditions. The basics of this algorithm might even be used for a wider range of graphs if some changes/extensions are made. Currently, the question what and how to do that is an

open research question.

So by abstracting our problem and by examining the resulting move graph, foundations have been laid to route qubit movements efficiently in fully connected star graphs (a special quantum processor architecture). Whether only full swaps or moves are allowed, in both cases, the number of operations can be minimized efficiently using the presented algorithms. Moreover, when each processing unit has only 2 spare qubits, even the optimal number of steps can be minimized efficiently. On top of that, the proposed algorithm can be applied to other quantum processor architectures. Together with the fact that the fully connected star graph is a model for a quantum processor based on promising diamond impurities, the potential of application of this work is high.

References

[1] N. Alon, F. R. K. Chung, and R. L. Graham, "Routing permutations on graphs via matchings," pp. 513–530, 1994.

[2] R. Courtland, "China's 2,000-km quantum link is almost complete [news]," *IEEE Spectrum*, vol. 53, no. 11, pp. 11–12, November 2016.

[3] E. Samuel Reich, "A solid case for Majorana fermions," *Nature*, vol. 483, p. 132, Mar. 2012.

[4] H. Bernien, B. Hensen, W. Pfaff, G. Koolstra, M. S. Blok, L. Robledo, T. H. Taminiau, M. Markham, D. J. Twitchen, L. Childress, and R. Hanson, "Heralded entanglement between solid-state qubits separated by 3 meters," Nature 497, 86-90 (2013), 2012, arXiv:1212.6136v1.

[5] G. Vadai, R. Mingesz, and Z. Gingl, "Generalized Kirchhoff-Law-Johnson-Noise (KLJN) secure key exchange system using arbitrary resistors," 2015, arXiv:1506.00950v1.

[6] M. Fujiwara, A. Waseda, R. Nojima, S. Moriai, W. Ogata, and M. Sasaki, "Unbreakable distributed storage with quantum key distribution network and password-authenticated secret sharing," 2016, arXiv:1607.00468v1.

[7] J. Bermejo-Vega, D. Hangleiter, M. Schwarz, R. Raussendorf, and J. Eisert, "Architectures for quantum simulation showing quantum supremacy," 2017, arXiv:1703.00466v1.

[8] M. J. Bremner, A. Montanaro, and D. J. Shepherd, "Achieving quantum supremacy with sparse and noisy commuting quantum computations," 2016, arXiv:1610.01808v4.

[9] D. Maslov, S. M. Falconer, and M. Mosca, "Quantum Circuit Placement," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27(4):752-763, April 2008, 2007, arXiv:quant-ph/0703256v2.

[10] D. Bhattacharjee and A. Chattopadhyay, "Depth-Optimal Quantum Circuit Placement for Arbitrary Topologies," 2017, arXiv:1703.08540v1.

[11] C. Busch, M. Magdon-Ismail, M. Mavronicolas, and P. Spirakis, *Direct Routing: Algorithms and Complexity*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 134–145. [Online]. Available: https://doi.org/10.1007/978-3-540-30140-0_14

[12] I. Banerjee and D. Richards, "Routing and Sorting Via Matchings On Graphs," 2016, arXiv:1604.04978v2.

[13] L. K. Grover, "A fast quantum mechanical algorithm for database search," 1996, arXiv:quant-ph/9605043v3.

[14] P. Diaconis, *Chapter 6: Metrics on Groups, and Their Statistical Uses*, ser. Lecture Notes–Monograph Series. Hayward, CA: Institute of Mathematical Statistics, 1988, vol. Volume 11, pp. 102–130. [Online]. Available: http://dx.doi.org/10.1214/lnms/1215467415

[15] W.-T. Li, L. Lu, and Y. Yang, "Routing numbers of cycles, complete bipartite graphs, and hypercubes," *SIAM J. Discret. Math.*, vol. 24, no. 4, pp. 1482–1494, Nov. 2010. [Online]. Available: http://dx.doi.org/10.1137/090776317

[16] F. R. K. Chung, *Spectral Graph Theory*, 1997.