# A Quantum Algorithm for Minimising the Effective Graph Resistance upon Edge Addition

Finn de Ridder[1(✉)], Niels Neumann[2], Thijs Veugen[2,3], and Robert Kooij[4,5]

[1] Radboud University, Nijmegen, The Netherlands
f.deridder@alumnus.utwente.nl
[2] TNO, The Hague, The Netherlands
{niels.neumann,thijs.veugen}@tno.nl
[3] CWI, Amsterdam, The Netherlands
[4] iTrust Centre for Research in Cyber Security,
Singapore University of Technology and Design, Singapore, Singapore
robert_kooij@sutd.edu.sg
[5] Faculty of Electrical Engineering, Mathematics and Computer Science,
Delft University of Technology, Delft, The Netherlands

**Abstract.** In this work, we consider the following problem: given a graph, the addition of which single edge minimises the *effective graph resistance* of the resulting (or, *augmented*) graph. A graph's effective graph resistance is inversely proportional to its *robustness*, which means the graph augmentation problem is relevant to, in particular, applications involving the robustness and augmentation of complex networks. On a classical computer, the best known algorithm for a graph with $N$ vertices has time complexity $\mathcal{O}(N^5)$. We show that it is possible to do better: Dürr and Høyer's quantum algorithm solves the problem in time $\mathcal{O}(N^4)$. We conclude with a simulation of the algorithm and solve ten small instances of the graph augmentation problem on the *Quantum Inspire* quantum computing platform.

**Keywords:** Graph augmentation · Effective graph resistance · Dürr and Høyer's algorithm · Quantum Inspire

## 1 Introduction

Our society depends on the proper functioning of several infrastructural networks, whose main function is to distribute flows of critical resources. Representative examples include electrical networks, distributing power through transmission links, and water networks, distributing water through pipe lines. The edges

Parts of this work are heavily based on the contents of De Ridder's master's thesis, in particular, Sects. 2 and 4. Readers interested in a more extensive treatment of the subject matter discussed in each of these sections are referred to the thesis available at https://www.ru.nl/publish/pages/769526/z_finn_de_ridder.pdf.


S. Feld and C. Linnhoff-Popien (Eds.): QTOP 2019, LNCS 11413, pp. 63–73, 2019.
https://doi.org/10.1007/978-3-030-14082-3_6

of these networks resist the passage of electric current and water molecules, respectively, and their resistance is governed by well-established physical laws. The physical characteristics of resistance in individual edges play a crucial role in the *robustness* of the network as a whole [1,2,13]. For this reason, the *effective graph resistance*, a graph metric also referred to as *Kirchhoff index*, is often used as a robustness indicator for complex networks (see [7]). The relationship between this metric and robustness is negative: a lower effective graph resistance indicates a more robust network.

Now suppose we can add a single edge to the network. Then the question that naturally arises is: "The addition of which (single) edge maximises the robustness of the augmented graph?" As robustness and the effective graph resistance are inversely proportional, in terms of the latter the problem becomes the following:

**Definition 1 (Graph Augmentation Problem).** *Given a graph $G$, the addition of which* single *edge $e$ minimises $R_{G+e}$, where $R_{G+e}$ denotes the effective graph resistance of $G$ augmented with $e$.*

An exhaustive search for a solution takes time $\mathcal{O}(N^5)$, where $N$ is the number of vertices in $G$ [20]. The worst-case running time of an exhaustive search depends on (i) the number of non-existing or *candidate edges* in $G$, on (ii) the worst-case running time of the best known algorithm that outputs the effective graph resistance of the graph it takes as input, and on (iii) the time it takes to find the minimum value amongst the outcomes. The number of candidate edges is $\mathcal{O}(N^2)$, and the best known algorithms for computing the effective graph resistance take time $\mathcal{O}(N^3)$ to do so, finding the minimum is also $\mathcal{O}(N^2)$ and accordingly, an exhaustive search takes time $\mathcal{O}(N^5)$.

There are different ways to compute the effective graph resistance of a graph. Currently, the most efficient way is to first compute the eigenvalues of the Laplacian of $G$, and make use of the fact that the effective graph resistance is equal to the sum of the multiplicative inverses of the non-zero eigenvalues. That is, Van Mieghem [16] has proven that

$$R_G = N \sum_{k=2}^{N} \frac{1}{\lambda_k}, \tag{1}$$

where $N$ is the number of vertices in $G$ and $\lambda_2 \leq \ldots \leq \lambda_N$ are the non-zero eigenvalues of its Laplacian. The *symmetric QR algorithm* (see for example [9]) can be used to find the eigenvalues of the Laplacian and takes time $\mathcal{O}(N^3)$.

In Sect. 4, we show that Dürr and Høyer's quantum algorithm can solve our problem in time $\mathcal{O}(N^4)$. Note, however, that our application of their quantum algorithm does *not* exploit graph-theoretic short cuts to outperform an exhaustive search for a solution, but instead makes use of the intricacies of quantum computation to arrive at a speed-up.

A part of Dürr and Høyer's algorithm, the *oracle*, is application dependent. To solve the graph augmentation problem, the oracle will need to compute the effective graph resistance of certain graphs, and to do so efficiently it could use

the symmetric QR algorithm—any algorithm for computing the effective graph resistance in time at most $\mathcal{O}(N^3)$ would work. Remember, however, that Dürr and Høyer's algorithm is a *quantum* algorithm, which means the oracle cannot "just" be a *classical* implementation of the symmetric QR algorithm—more details on the oracle will be given in Sect. 4.

Before we do so, we first discuss in Sect. 2 related works on the graph augmentation problem and applications of quantum algorithms to graph theory. Section 3 is a brief introduction to quantum computation and serves as a preparation for Sect. 4, which embodies our main contribution. Before we conclude in Sect. 6, we discuss in Sect. 5 our simulation of the algorithm presented in Sect. 4.

## 2 Related Work

Ghosh et al. [8] have studied the problem of assigning weights to the edges of a given graph $G$, such that $R_G$ is minimised. They also show that among all graphs on $N$ vertices, the graph for which the effective graph resistance is maximal is the path graph $P_N$, and that the complete graph $K_N$ has the smallest effective graph resistance. A theorem by Deng [4] gives the minimum effective graph resistance among all graphs $G$ on $N$ vertices that have $k$ bridges; a *bridge* of a not-necessarily-connected graph $G$ is an edge whose removal increases the number of components of $G$ [11]. Van Mieghem [16] has shown that, if $G$ is a connected graph on $N$ vertices, then

$$\frac{(N-1)^2}{\bar{d}} \leq R_G, \tag{2}$$

where $\bar{d}$ denotes the average degree of the $N$ vertices. The bound is tight, i.e., there exists a graph $G$ for which $R_G = (N-1)^2/\bar{d}$, namely the complete graph $K_N$. Finally, Wang et al. [20] have proposed a handful of heuristics for the graph augmentation problem and in [19], the problem is studied in the context of power transmission grids.

Before we briefly discuss related works on applications of quantum algorithms to graph theory, it is worthwhile to mention that, for a problem related to the graph augmentation problem, namely that of *maximising* the *algebraic connectivity* of the graph upon edge addition, a more efficient algorithm (than exhaustive search) has been found already: Kim's bisection algorithm [12] has a time complexity of $\mathcal{O}(N^3)$, which is better than an exhaustive search, which also for this problem, takes time $\mathcal{O}(N^5)$. Heuristics for the problem can be found in the work by Wang and Van Mieghem [17].

Perhaps the second best-known quantum algorithm, after Shor's algorithm, is Grover's algorithm [10]. In fact, Dürr and Høyer's algorithm is a generalisation of Grover's. Grover's algorithm searches an unsorted database, to find some particular record, and has a *query complexity* of $\mathcal{O}(\sqrt{n})$ where $n$ is the number of records in the database. A classical search, by contrast, requires exactly $n$ queries in the worst case. Grover's algorithm achieves a *quadratic speed-up*.

Based on Grover's algorithm, Dürr et al. [5] formulated a quantum algorithm that determines whether a graph on $N$ vertices is connected, solving the problem in time $\mathcal{O}(N\sqrt{N})$, up to logarithmic factors. A classical computer requires time of order $N^2$, in the worst case. They also gave efficient algorithms for some other graph-theoretic problems related to strong connectivity, minimum spanning trees and shortest paths.

## 3 Quantum Computation

The following is a very short introduction to quantum computation. For a more elaborate introduction to quantum computing we refer to the standard work on the subject by Nielsen and Chuang [14].

Classical computing relies on bits to perform operations. By manipulating bits, operations are performed and results obtained. Quantum computers in principle do the same with quantum bits, or *qubits*. The state of a qubit $|\psi\rangle$ can be described by a vector

$$|\psi\rangle = \alpha_0 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \alpha_1 \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \alpha_0|0\rangle + \alpha_1|1\rangle,$$

with the last expression in the so-called *ket*-notation. The complex coefficients $\alpha_0$ and $\alpha_1$ are called *amplitudes* and are subject to $|\alpha_0|^2 + |\alpha_1|^2 = 1$, to make $|\psi\rangle$ a valid quantum state.

If both amplitudes are non-zero, $|\psi\rangle$ is said to be in a *superposition* of the states $|0\rangle$ and $|1\rangle$. The combined state of two qubits $|\psi_1\rangle$ and $|\psi_2\rangle$ is $|\psi_1\rangle \otimes |\psi_2\rangle$, where $\otimes$ denotes the *tensor product*. Often however, it is not possible to decompose a combined state into a tensor product. If such decomposition is not possible, the qubits are said to be *entangled*.

There is an important difference between classical logic gates and quantum gates: the latter need to be reversible. Accordingly, a quantum gate can be described by a unitary operator. Analogous to classical computing, problems can be solved on a quantum computer by applying the right quantum gates on the right qubits in the right order.

## 4 The Algorithm

As mentioned before, Dürr and Høyer's algorithm is based on a generalisation of the well-known quantum search algorithm by Grover [10]. The generalisation was formulated by Boyer et al. [3], two years after Grover made public his pioneering research.

Given an unsorted database $T$, Dürr and Høyer's algorithm is able to find the *index* of the smallest entry in $T$. As one might expect, $T$ will contain for each *candidate edge* $e$ of $G$ (i.e., each edge that is not in $G$ already) the effective graph resistance $R_{G+e}$ of the augmented graph $G+e$. Accordingly, the algorithm will return the index of the edge that we are looking for.

This brings up the question: "How does the algorithm acquire $T$?" After all, if $T$ would be an argument of the algorithm, there would be no need to use it: finding the minimum in an unsorted database, given the database, is trivial and takes time proportional to the number of entries in the database. The actual answer is a bit more involved, but at the same time exemplary of the difference between *quantum* algorithms and algorithms that run on conventional computers.

Conceptually, the algorithm constructs and *queries* $T$ at runtime. To that end, it is given a single argument known as the oracle: the oracle is a function $f$ which, given two valid indices $(i, i')$ of $T$, returns $i$, if $T[i] < T[i']$, and $i'$, otherwise. The algorithm queries the database through the oracle. First, it will choose uniformly at random an index $i_r$ and then "ask" the oracle whether there exists another index $i_1$ different from $i_r$ such that $T[i_1] < T[i_r]$, and if so, to return $i_1$ (otherwise the oracle will simply return $i_r$). In the unlikely event that the oracle returns $i_r$, the solution was found by a single guess and we are done. If instead a different index $i_1$ is returned, a new query is raised: "Does there exist an index $i_2$ such that $T[i_2] < T[i_1]$?" These steps are repeated until the solution is found. Bear in mind, however, that the preceding is only a simplified explanation of the algorithm, to give some intuition into its workings. The actual specification is given in the following section.

### 4.1 Specification

As before, let $T$ denote the unsorted database that we are searching through and $T[i]$ the entry at index $i$. The number of entries in $T$ equals $n$. Dürr and Høyer's algorithm is shown in Fig. 1.

The algorithm outputs $i_m$ with probability larger than $1/2$ after at least $\left\lceil 8\pi\sqrt{n} \right\rceil$ applications of the Grover iterate. Therefore, if the algorithm is run $c$ times, the probability that the minimum is among the results is at least $1 - (1/2)^c$, which converges rapidly to 1 as $c$ increases. For example, after running the algorithm $c = 8$ times, the minimum is part of the outcomes with a probability that is higher than 0.99.

### 4.2 Complexity Analysis

The loop described in step (3) should be interrupted only if, with probability at least $1/2$, $i_t$ equals $i_m$. Accordingly, the running time of the algorithm is largely dependent on how long it takes for the foregoing to hold.

We define $X$ to be the time it takes until $i_t := i_m$. By Markov's inequality,

$$P\{X < 2\mathbb{E}[X]\} > \frac{1}{2} \tag{4}$$

and therefore, if we interrupt the loop after running for time $2\mathbb{E}[X]$, we know $P\{i_t = i_m\}$ is at least $1/2$. The remainder of this section will be about computing the expectation of $X$.

**Input.** A quantum circuit $U_f$ that implements the oracle $f$.
**Output.** With probability $p > 1/2$, index $i_m$ such that $T[i_m]$ is the smallest entry in the database.

1. Define $\lambda$ to be $8/7$ (see [3]).
2. Choose threshold index $i_t \in \mathbb{Z}$, $0 \leq i_t \leq n-1$, uniformly at random.
3. Repeat the following until the cumulative sum of the number of applied Grover iterates exceeds $\lfloor 8\pi\sqrt{n} \rfloor$. Afterwards, go to (4.).
   a. Use the generalisation of Grover's algorithm by Boyer et al. [3] to search for an index $i$ such that $T[i] < T[i_t]$. That is,
      i. Initialise $s = 1$.
      ii. Initialise two $n$-qubit registers as $\sum_{i=0}^{n-1} \frac{1}{\sqrt{n}} |i\rangle |i_t\rangle$.
      iii. Choose $l \in \mathbb{Z}$, $0 \leq l < s$, uniformly at random.
      iv. Apply the Grover iterate $l$ times. The oracle $f$ is implemented by $U_f$ as follows:
      $$U_f : |i\rangle |i_t\rangle \mapsto \begin{cases} -|i\rangle |i_t\rangle & \text{if } T[i] < T[i_t] \\ |i\rangle |i_t\rangle & \text{otherwise} \end{cases}. \tag{3}$$
      v. Measure the first register. Let $i'_t$ be the outcome. If
      - $T[i'_t] < T[i_t]$ let $i_t := i'_t$ and return, i.e. go to (a.);
      - otherwise let $s := \min(\lambda s, \sqrt{n})$ and go to (ii.).
4. Return $i_t$.

**Fig. 1.** Dürr and Høyer's algorithm

In step (v) of Boyer et al.'s algorithm, $i_t$ is changed if $T[i'_t] < T[i_t]$, after which the algorithm returns. The algorithm is then run again, if the cumulative sum of the number of applied Grover iterates does not yet equal $l$. For that reason, each *possible* execution of Boyer et al.'s algorithm can be associated with a distinct $i_t$: the threshold index just before the algorithm is executed.

In the worst case, i.e. if the running time of Dürr and Høyer's algorithm is maximal, the initial threshold index $i_{t_0}$, which is chosen uniformly at random in step (2), is such that $T[i_{t_0}]$ is the largest value in $T$. In addition, every possible execution of Boyer et al.'s algorithm is realised: at the end of each execution, the threshold index $i_t$ is changed to $i'_t$ such that $T[i'_t]$ is the largest entry in the database smaller than $T[i_t]$. Necessarily, the number of times Boyer et al.'s algorithm is executed equals $n-1$.

Fortunately for us, it is unlikely that all $n-1$ possible executions are required. Most will be skipped, automatically, simply because the result of Boyer et al.'s algorithm is an index $i'_t$ such that $T[i'_t]$ is not only smaller than $T[i_t]$, but also smaller than a handful of different entries each of which is also smaller than $T[i_t]$, but that were just unlucky and have not been chosen. More than that, it is impossible that, in the future, they are chosen.

The expectation of $X$ is the sum of the expected running times of each possible execution of Boyer et al.'s. That is, let $Y_k$ be a random variable defined as the running time of the $k$th possible execution. We have,

$$\mathbb{E}[X] = \mathbb{E}[Y_1] + \mathbb{E}[Y_2] + \cdots + \mathbb{E}[Y_{n-1}], \tag{5}$$

where $\mathbb{E}[Y_k]$ is $p_k L_k$: the product of the probability $p_k$ of execution $k$ taking place and the running time or length $L_k$ of execution $k$, given it takes place, respectively. Observe that the $k$th possible execution occurs directly after, and at no other point in time, threshold index $i_t$ becomes $x$ such that $T[x]$ is the $k$th largest entry in $T$. As a result, $p_k$ is equal to the probability that $i_t := x$.

In the same paper in which they present their algorithm, Dürr and Høyer prove by induction that $p_k = 1/r$ where $1 \leq r \leq n$ is the *ranking* of $T[x]$, which is 1 if the entry is minimal, and $n$ if it is maximal (see Lemma 1 in [6]). Also, note that $r = n - k + 1$. As a result, for example,

$$\mathbb{E}[Y_{n-1}] = \frac{1}{2} L_{n-1}, \tag{6}$$

because the $(n-1)$th execution, the last of all possible executions, occurs if and only if during an earlier execution $i_t := y$, such that the ranking $r$ of $T[y]$ is 2, which by Dürr and Høyer's lemma happens with probability $p = 1/r = 1/2$. We find that

$$\mathbb{E}[X] = \frac{1}{n} L_1 + \frac{1}{n-1} L_2 + \cdots + \frac{1}{2} L_{n-1}, \tag{7}$$

and are left with the assignment to find $L_k$.

The length $L_k$ of a single execution is the sum of the lengths of steps (ii) and (iv); steps (i), (iii), and (v) take a negligible amount of time and are, accordingly, neglected in the complexity analysis. The duration of (ii) is the same for all iterations: by convention, it is $\log_2 n$ [6]. More difficult is the determination of the duration of (iv), because it depends on $k$.

Boyer et al. show that the expected number of applications of the Grover iterate, until their algorithm finds the minimum, is at most $8m_0$, where

$$m_0 = \frac{n}{2\sqrt{(n-t)t}}. \tag{8}$$

In (8), $t$ is the number of solutions, which is known for each possible execution, and allows us to find $L_k$:

$$L_k = 8m_0 B + \log_2 n = \frac{4Bn}{\sqrt{(n-k)k}} + \log_2 n, \tag{9}$$

where $B$ is the complexity of a single query, i.e. the complexity of $U_f$. If $n$ is sufficiently large and $\sqrt{n+1} \approx \sqrt{n}$, it is possible to show that, consequently

$$\mathbb{E}[X] \leq 4\pi B\sqrt{n} + \ln n \log_2 n. \tag{10}$$

After time at least twice the upper bound of (10), i.e. after time $8\pi B\sqrt{n} + 1.39\log_2^2 n$ we should stop executing Boyer et al.'s algorithm and move on to step (4). That is, after at least $\left\lceil 8\pi\sqrt{n} \right\rceil$ applications of the Grover iterate, the probability that $i_t = i_m$ is at least 1/2. Accordingly, Dürr and Høyer's algorithm

runs in time $\mathcal{O}(B\sqrt{n})$ where $B$ is the time complexity of querying the oracle and $n$ the size of $T$. The size of $T$ will be equal to the number of edges not in $G$ and therefore always smaller than $N^2$.

What is $B$, the complexity of $U_f$? If we query the oracle as to whether $T[i] < T[i_t]$, it must first compute $T[i] = R_{G+e_i}$. As mentioned before, the *symmetric QR algorithm* can be used to compute the eigenvalues of the Laplacian $Q$ of $G + e_i$, which can then be used to compute $R_{G+e_i}$. Since any classical algorithm can be implemented efficiently on a quantum computer using only *Toffoli gates* [14], the time complexity $B$ of the oracle is equal to the complexity of the symmetric QR algorithm, which is $\mathcal{O}(N^3)$. Hence, the complexity of Dürr and Høyer's algorithm, applied to the problem of minimising the effective graph resistance of the augmented graph, becomes $\mathcal{O}(N^4)$.

## 5 Simulation

The algorithm presented in Sect. 4 has been implemented on the quantum computing platform *Quantum Inspire* [15], developed by QuTech. QuTech is an advanced research centre in the Netherlands where the Technical University of Delft and TNO collaborate together with industrial partners to build a quantum computer and a quantum internet. The platform allows for simulations (of quantum computers) of up to 31 qubits.

Programming using Quantum Inspire is possible both via a web interface and via a software development kit (SDK). The SDK makes it possible to construct hybrid quantum/classical algorithms and run quantum instructions from the command line.

An (arbitrary) labelling of the candidate edges of the graph is needed to implement the algorithm. The binary representation of these labels relates directly to the states of the different qubits in the implementation. For example, state $|011\rangle$ relates to label 3. With high probability, the algorithm outputs the candidate edge whose addition minimises the effective graph resistance of the augmented graph. The algorithm is iterative and hence has multiple passes and multiple rounds of measurements.

An example of an implementation of the algorithm for only a single Grover iteration is shown in Fig. 2, other iterations are similar. Note that the circuit is subdivided in three parts. The first part, *state preparation*, creates an equal superposition over all quantum states in the first register and sets the quantum state to $|i_t\rangle$ in the second register.

In the second part, a phase-oracle is used to flip the amplitude of specific quantum states. That is, the amplitude of quantum states of edges better than $i_t$ are flipped. A Toffoli-gate combined with two Hadamard-gates flips the amplitude of the $|111\rangle$-state. Using $X$-gates before and after the controls of the Toffoli-gate, amplitudes of other states are flipped, i.e. with $X$-gates the target has to be in the $|0\rangle$-state for the amplitude to be flipped, without $X$-gates in the $|1\rangle$-state. In the example in Fig. 2, the oracle flips the amplitudes of the states $|110\rangle$, $|010\rangle$, and $|101\rangle$. The $CNOT$-gate with adjoining $X$- and Hadamard-gates is for the

first two states, flipping the amplitude of $|\cdot 10\rangle$. The amplitude of the $|101\rangle$-state is flipped using the Toffoli-gate with adjoining $X$- and Hadamard-gates. In this example, the edges labelled 2, 5, and 6 result in a lower effective graph resistance than edge $i_t = 3$.

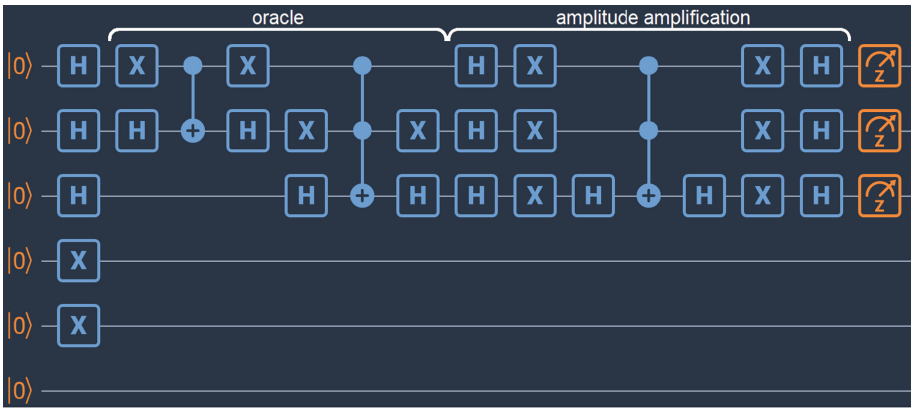


**Fig. 2.** The circuit implementation of the algorithm for $i_t = 3$. First, we have the state preparation part, afterwards a phase-oracle is applied using Hadamard-, $X$-, $CNOT$- and Toffoli-gates. Finally, amplitude amplification is applied and the first register is measured.

The last part of the algorithm is *amplitude amplification*: amplifying the amplitude of "good" states, while suppressing that of "bad" states. Note that the shown implementation is a non-optimised circuit-implementation, as for instance two Hadamard gates after each other do not change the quantum state.

We have tried the algorithm on ten small graphs, using only eight of the available 31 qubits. The ten graphs, labelled $G1, \ldots, G10$ as in Fig. 3, each consist of seven vertices and ten edges in total and are obtained from [18]. In the experiment we focused solely on the accuracy of the algorithm and not on the observed running times—after all, we are only simulating execution.

For each of the ten graphs we evaluated the algorithm a hundred times to test its accuracy and in each of the hundred runs, for each of the ten graphs, the best edge to add was found. We also found that, at least for these graphs, the upper bound on the number of iterations is a loose bound. For these graphs the upper bound would be $\left\lceil 8\pi\sqrt{11} \right\rceil = 84$, while logging information learned that for all graphs and for all hundred runs, after at most ten iterations the best edge was found already.

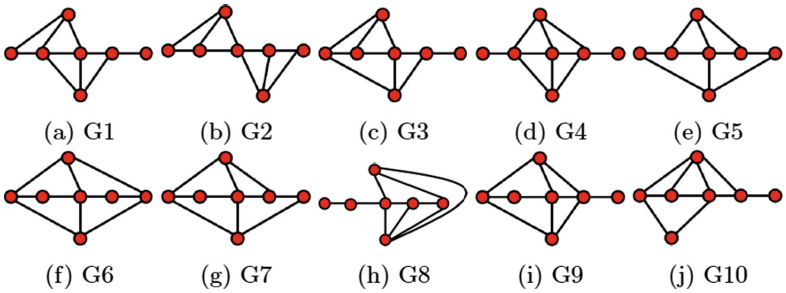


**Fig. 3.** The ten graphs $G1, \ldots, G10$ used to test the algorithm.

## 6 Conclusion

We have shown that the time complexity of Dürr and Høyer's algorithm for minimising the effective graph resistance is better than the time complexity of an exhaustive search. This means that there exists a quantum circuit that solves the optimisation problem and whose *depth*, compared to the depth of a classical circuit implementing an exhaustive search, increases significantly less fast as the number of vertices $N$ increases. As a result, it is likely that, at least for large instances, our algorithm running on a gate-based quantum computer will outperform an exhaustive search on a classical computer.

Our simulation of the algorithm shows that indeed with high probability, the best edge to add to the graph is found. In fact, in all runs, the solution was found. Our implementation is easily extended to also support larger graphs by increasing the number of qubits, adding extra controls to the controlled-NOT and Toffoli-gates, and by adding $X$-gates to the oracle.

Finally, it is very likely that there exist many other optimisation problems, for which also no clever algorithm has (yet) been formulated, and whose time complexity can be improved upon as well. Simply by taking advantage of the quadratic *reduction* of the time spent searching, provided by Dürr and Høyer's algorithm.

## References

1. Abbas, W., Egerstedt, M.: Robust graph topologies for networked systems. In: IFAC Proceedings, vol. 45, no. 26, pp. 85–90, September 2012. https://doi.org/10.3182/20120914-2-US-4030.00052, https://linkinghub.elsevier.com/retrieve/pii/S147466701534814X
2. Asztalos, A., Sreenivasan, S., Szymanski, B., Korniss, G.: Distributed flow optimization and cascading effects in weighted complex networks. Eur. Phys. J. B **85**(8) (2012). https://doi.org/10.1140/epjb/e2012-30122-3, http://www.springerlink.com/index/10.1140/epjb/e2012-30122-3
3. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte der Physik **46**(4–5), 493–505 (1998)
4. Deng, H.: On the minimum Kirchhoff index of graphs with a given number of cut-edges. MATCH Commun. Math. Comput. Chem. **63**, 171–180 (2010)

5. Dürr, C., Heiligman, M., Høyer, P., Mhalla, M.: Quantum query complexity of some graph problems. SIAM J. Comput. **35**(6), 1310–1328 (2006)
6. Dürr, C., Høyer, P.: A quantum algorithm for finding the minimum. arXiv:quant-ph/9607014, July 1996. http://arxiv.org/abs/quant-ph/9607014
7. Ellens, W., Spieksma, F., Van Mieghem, P., Jamakovic, A., Kooij, R.: Effective graph resistance. Linear Algebra Appl. **435**(10), 2491–2506 (2011). https://doi.org/10.1016/j.laa.2011.02.024, http://linkinghub.elsevier.com/retrieve/pii/S0024379511001443
8. Ghosh, A., Boyd, S., Saberi, A.: Minimizing effective resistance of a graph. SIAM Rev. **50**(1), 37–66 (2008). https://doi.org/10.1137/050645452, http://epubs.siam.org/doi/10.1137/050645452
9. Golub, G.H., Van Loan, C.F.: Matrix Computations. Johns Hopkins Studies in the Mathematical Sciences, 4th edn. The Johns Hopkins University Press, Baltimore (2013)
10. Grover, L.K.: A fast quantum mechanical algorithm for database search, pp. 212–219. ACM Press (1996). https://doi.org/10.1145/237814.237866, http://portal.acm.org/citation.cfm?doid=237814.237866
11. Harary, F.: Graph Theory. Perseus Books, Cambridge (2001)
12. Kim, Y.: Bisection algorithm of increasing algebraic connectivity by adding an edge. IEEE Trans. Autom. Control **55**(1), 170–174 (2010). https://doi.org/10.1109/TAC.2009.2033763, http://ieeexplore.ieee.org/document/5340588/
13. Koç, Y., Warnier, M., Mieghem, P.V., Kooij, R.E., Brazier, F.M.: The impact of the topology on cascading failures in a power grid model. Physica A: Stat. Mech. Appl. **402**, 169–179 (2014). https://doi.org/10.1016/j.physa.2014.01.056, https://linkinghub.elsevier.com/retrieve/pii/S0378437114000776
14. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information, 10th Anniversary edn. Cambridge University Press, Cambridge (2010)
15. QuTech: Quantum Inspire (2018). https://www.quantum-inspire.com/. Accessed 15 Nov 2018
16. Van Mieghem, P.: Graph Spectra for Complex Networks. Cambridge University Press, Cambridge (2011)
17. Wang, H., Van Mieghem, P.: Algebraic connectivity optimization via link addition. In: Proceedings of the 3rd International Conference on Bio-Inspired Models of Network, Information and Computing Sytems, BIONETICS 2008, pp. 22:1–22:8, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels (2008). http://dl.acm.org/citation.cfm?id=1512504.1512532
18. Wang, X., Feng, L., Kooij, R.E., Marzo, J.L.: Inconsistencies among spectral robustness metrics. In: Proceedings of QSHINE 2018–14th EAI International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (2018)
19. Wang, X., Koç, Y., Kooij, R.E., Van Mieghem, P.: A network approach for power grid robustness against cascading failures. In: 2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM), pp. 208–214. IEEE, Munich, October 2015. https://doi.org/10.1109/RNDM.2015.7325231, http://ieeexplore.ieee.org/document/7325231/
20. Wang, X., Pournaras, E., Kooij, R.E., Van Mieghem, P.: Improving robustness of complex networks via the effective graph resistance. Eur. Phys. J. B **87**(9) (2014). https://doi.org/10.1140/epjb/e2014-50276-0, http://link.springer.com/10.1140/epjb/e2014-50276-0