Graph Augmentation

A Quantum Algorithm for Maximising Robustness

by

Finn de Ridder

Submitted in partial fulfilment of the requirements for the degree of

MSc in Computing Science (TRU/e)

at Radboud University.

Supervisor	Dr. Aleks Kissinger
External Supervisor	Dr. Ir. Thijs Veugen (TNO)
Daily Supervisor	MSc Niels Neumann (TNO)

Abstract

The *robustness* of a network is the extent to which the network is able to continue performing well when it is subject to failures or attacks [18, p. 1]. In order to *quantify* a network's robustness, all kinds of graph measures have been invented, each of which captures a different perception of robustness. This work revolves around two of them: the graph's *algebraic connectivity* and its *effective graph resistance*. The larger the graph's algebraic connectivity, the more robust the graph, while the larger the graph's effective graph resistance, the less robust the graph.

We asked ourselves two questions, both of which are easy to understand: given a connected graph, the addition of which single edge will maximally increase the graph's algebraic connectivity? And, given a connected graph, the addition of which single edge will maximally decrease its effective graph resistance? Up to now, no one has detected any structure in either problem. An exhaustive search takes time $\mathcal{O}(N^5)$, irrespective of whether robustness is defined as the graph's algebraic connectivity or minus its effective graph resistance. Kim [30] has formulated a clever kind of search that will output the answer to the first question in time $\mathcal{O}(N^3)$, but otherwise no algorithms have been proposed.

In this work, we first attempt to discover a structure in the second problem. It has been shown that the effective graph resistance of a graph relates to the eigenvalues of the graph's *Laplacian*, to the *random walk* on the graph, and to the number of *spanning trees*. We make use of the last two to slightly simplify the problem and as a result dispose of a fraction of its complexity — a lot of complexity remains.

Afterwards we present the main contribution of this work: the application of a *quantum search algorithm* by Dürr and Høyer [16] to both problems. We show that the algorithm can be used to answer both questions in time $\mathcal{O}(N^4)$, which means that it is the "fastest" known algorithm for solving the problem of minimising the effective graph resistance of the augmented graph. Finally, we also illustrate how heuristics can be used to speed up Dürr and Høyer's algorithm, in particular, we explain how by taking advantage of a very simple heuristic, the *query complexity* of the algorithm can be *halved*.

Acknowledgments

Many thanks to

- Aleks Kissinger, for bringing your expertise on quantum computation, for being very accessible and frank;
- Robert Kooij, for coming up with the idea to apply quantum computing to the graph augmentation problem. I think that two months has been more than enough to convey your enthusiasm. Also many thanks to
- Thijs Veugen, for taking over the baton from Robert, and your great precision; and
- Niels Neumann, for sharing your knowledge of *both* quantum computation and graph theory, and for showing me the way at TNO.[1]

Finally, the work climate at TNO has been exceptional and accordingly has made a significant positive contribution to this work.

I would like to thank all three supervisors, and Robert Kooij, for being confident that I — a cybersecurity student with little knowledge of but a great interest in quantum computation and graph theory — would bring this thesis to a favourable conclusion.

[1] TNO, the *Netherlands Organisation for Applied Scientific Research*, is an independent non-profit research organisation in the Netherlands.

Contents

Chapter 1

Introduction

1.1 The Problem

A *network* is a set of *vertices* or *nodes* joined together by *edges* and usually associated with a real-world phenomena. Probably the most well-known example of a network is the Internet. The Internet is a network of computers: vertices represent computers and edges the wires that connect them. Computer networks such as the Internet are just one example of the variety of phenomena that are naturally described by a network. Other widely studied examples include social networks (see Fig. 1.1), which describe the relationships between people or groups of people, and neural networks, which consist of neurons and the synapses that interconnect them.

If we abstract away from what the vertices and edges in a network stand for, networks become equivalent to *graphs*. *Graph theory*, the study of graphs, constitutes a large part of discrete mathematics.

There is a lot to say about graphs. Accordingly, a large number of properties of graphs and graph measures have been defined. This work is centred around a particular graph measure called *robustness*. We will adopt the loose definition of robustness by Ellens and Kooij [18, p. 1]:

Definition 1 (Robustness)**.** The extent to which the network is able to continue performing well when it is subject to failures or attacks.

Arguably, robustness is more of a network measure than it is a graph measure, as it relates to the *performance* of a network, something which is dependent on the phenomena the network describes. Consequently, there is a multiplicity of formulas for computing a network's robustness, each formula expressing a different interpretation of performance. A survey by Ellens and Kooij [18] provides for an overview of several graph measures that intuitively quantify a graph's robustness.

In their survey, Ellens and Kooij only consider *topological measures*, i.e. measures that capture (part of) the network's topology, the arrangement of

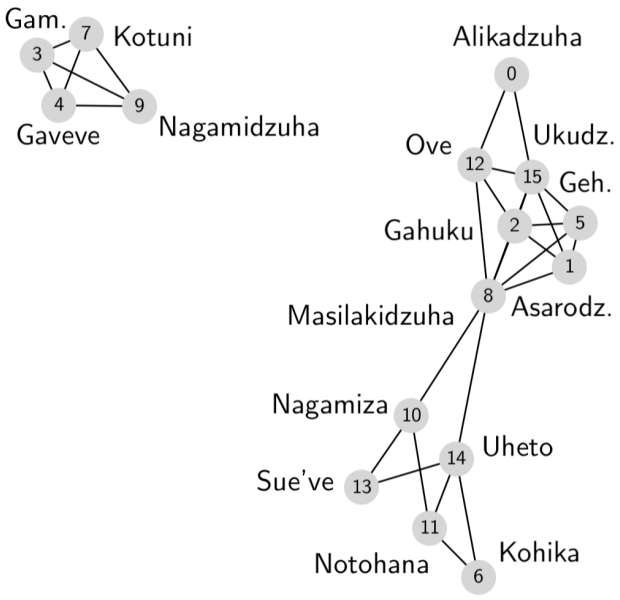


Figure 1.1: The friendly relationships between Gahuku-Gama sub-tribes in New Guinea around 1950, depicted as a network. Based on the graph in Fig. 5 of [42, p. 41] which also includes the oppositions between the sub-tribes.

vertices and edges, as opposed to measures that also relate to what happens *on* the network. To illustrate the difference, if the network is the Internet, a topological measure would be the average number of wires connected to a computer, whereas a measure such as the average amount of time between a computer sending a message and another receiving it will depend on the network's topology as well, but definitely also on how many other messages are being sent, for example, or on how messages are being routed.

In this work, we will only consider two out of the many graph measures that are believed to reflect a graph's robustness: *algebraic connectivity* and *effective graph resistance* (see Sects. 2.2 and 2.3, respectively, for their definitions). Both measures are topological measures, which makes what happens on the network irrelevant. For that reason, we will from now on use the term graph more frequently, and the term network only whenever we talk about robustness, or when an example is given.

We will now define the problem addressed by this work.

Definition 2 (Problem)**.** Given a network, the addition of which single edge will maximally increase the network's robustness? We consider two instances of the problem, respectively: one in which the network's robustness is defined as the graph's algebraic connectivity, and one in which it's defined as minus the graph's effective graph resistance. Indeed, the larger the

graph's algebraic connectivity, the more robust the graph, while the larger the graph's effective graph resistance, the less robust the graph.

We will also briefly discuss why these measures are natural indicators of robustness (see Sects. 2.2 and 2.3), but it should be emphasised that we do not intend to provide for a comparison between them and other measures. For a comparison, see for example [46].

1.2 Exhaustive Search

Our problem is to find an *algorithm* that takes as input the *adjacency matrix* $A(G)$ (see Sect. 2.1.1) of a graph G and outputs the edge e' that, if added to G, either maximally increases the graph's algebraic connectivity or maximally decreases its effective graph resistance. A natural question to ask is, what is the worst-case running time of an exhaustive search? An exhaustive search involves computing, for all graphs $G + e$ that can be obtained by adding a candidate edge e to G, the graph's algebraic connectivity or effective graph resistance and finding the maximum or minimum, respectively, among them. Therefore, the worst-case running time of an exhaustive search is dependent on

- the number of candidate or non-existing edges in G;
- the worst-case running time of computing the algebraic connectivity or effective graph resistance of $G + e$; and on
- the worst-case running time of finding the maximum or minimum among the outcomes.

In fact, we are not interested in the worst-case running time of the search, but rather in the *order of growth* of the running time as a function of the *size* of the graph G for which we would like to solve the problem. The size of a graph G is naturally dependent on N, the number of vertices in G, and on M, the number of edges.

If either of the three parts of the computation (see above) cannot be bounded above by some $cg(N, M)$, where c is a constant and $g(N, M)$ a polynomial, then neither can the running time of the computation as a whole. Consequently, the exhaustive search would be an *exponential time algorithm*. Otherwise, if each part can be bounded above by some $cg(N, M)$, then the running time of the computation as a whole can also be bounded above by a polynomial and as a result, the algorithm is a *polynomial time algorithm*.

As long as no polynomial time algorithm has been found to solve a problem, the problem is considered to have not been "well-solved" [23, p. 8]. The underlying idea is that for a lot of problems, the polynomial time algorithm

that solves it takes advantage of the problem's hidden structure, which usually is discovered only after a profound understanding of the problem has been obtained. Most exponential time algorithms, on the other hand, are simply exhaustive search in disguise and in general do not require much more than a basic understanding of the problem.

Is exhaustive search, applied to our problem, an exponential or polynomial time algorithm? Let's consider each part of the search one after the other.

Number of candidate edges in G. A complete graph, a graph in which each pair of vertices is connected by an edge, has $\binom{N}{2} = N(N-1)/2$ edges. Therefore, the number of candidate edges in G is $N(N-1)/2 - M$, which can be bounded above by $cg(N)$ where c is a positive constant and $g(N) = N^2$.

Computing algebraic connectivity or effective graph resistance. One way to compute a graph's algebraic connectivity or effective graph resistance, is by computing the eigenvalues of the symmetric N by N *Laplacian* matrix of the graph G (see Sect. 2.2 for a definition), followed by a few inexpensive operations that can be neglected. The *symmetric QR algorithm* can be used to find the eigenvalues of the Laplacian, and its running time can be bounded above by $cg(N)$ where c is again a positive constant and $g(N) = N^3$ [25, p. 463].

Finding the maximum or minimum. In order to find the maximum or minimum, we simply need to iterate over all the outcomes whilst keeping track of the largest or smallest value, respectively, encountered up till now. It is possible that there is more than one candidate edge whose addition would result in a maximum increase in algebraic connectivity or decrease in effective graph resistance, but we only need to find one to solve our problem. The running time of this part of the computation can therefore be bounded above by a positive constant c times the maximum number of candidate edges N^2.

Accordingly, the running time of an exhaustive search can be bounded above by the polynomial $c\left(N^2 \cdot N^3 + N^2\right) = c\left(N^5 + N^2\right)$ where c is once more a positive constant, which is equivalent to saying that the running time of the search is $\mathcal{O}(N^5)$.[1] Consequently, the exhaustive search, as outlined above, is a polynomial time algorithm.

However, although this means that a polynomial time algorithm that solves our problem exists, because the algorithm is a mere exhaustive search that does not take advantage of the problem's hidden structure — if it exists

[1] A function $f(n)$ is $\mathcal{O}\left(g(n)\right)$ if there exist positive constants c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.

at all — the question as to whether it is possible to formulate a polynomial time algorithm that *does* take advantage of the problem's hidden structure remains open.

1.3 Kim's Bisection Algorithm

There does exist a polynomial time algorithm by Kim [30] that solves one of the two instances of our problem: the problem of finding the edge e' that would maximally increase the graph's algebraic connectivity, if it were to be added to G; but it does also not make use of a hidden structure and can be regarded as a clever kind of exhaustive search. The *time complexity* of the Kim's bisection algorithm is $\mathcal{O}(N^3)$, which is better than the time complexity $\mathcal{O}(N^5)$ of the exhaustive search given above.

Other than Kim's algorithm, we are unaware of any algorithm that solves the first instance of our problem. For the second instance of our problem, in which the network's robustness is defined as minus the graph's effective graph resistance, no algorithm seems to exist. Indeed, not even a clever kind of exhaustive search. In their paper about effective graph resistance, Ellens et al. [19, p. 2505] propose, as a direction for future research, the "design of an algorithm for determining the edge that decreases the effective graph resistance most, without having to try all possible edges" — which is what our second problem instance is about.

1.4 Quantum Computation

There are many different *algorithm paradigms* that we could use as the starting point of our search for an algorithm to solve our problem efficiently. Well-known examples include the *divide-and-conquer*, *dynamic programming*, and *greedy* paradigms. Our starting point, however, will be none of these established paradigms. Instead, we will explore whether it is possible to use a *quantum algorithm* to solve our problem. Quantum computation is not just another algorithm paradigm, rather it is a different *model of computation*; different from the *Turing machine* model of computation, which underlies the algorithm paradigms mentioned before.

In the end, computation is a form of controlled physical evolution. *Quantum computers*, i.e. computers able to run quantum algorithms, translate their instructions, the algorithms they are asked to execute, into different physics compared to the physics exploited by conventional computers. The physics that quantum algorithms are translated into are accurately described by *quantum theory*. Accordingly, quantum computation, as a model for computation, adheres to what quantum theory dictates.

As a result, quantum algorithms have some remarkable features that classical algorithms do not, in particular *quantum parallelism* and *interfer-*

ence (see Sect. 3.2). At the same time, quantum theory imposes some serious limitations on quantum algorithms, limitations that classical algorithms do not have. The task of the quantum algorithm designer is to work out how to take advantage of the characteristic features of quantum theory, given a particular computational problem. Needless to say, some problems lend themselves better for the features of quantum computation than others.

This brings up the question, whether our problem is naturally solved by a quantum algorithm. In Chap. 4 we present a quantum algorithm based on an algorithm by Dürr and Høyer [16] for solving both problem instances, the main contribution of this work. The complexity of the algorithm is $\mathcal{O}(N^4)$: better than the exhaustive search of Sect. 1.2, worse than Kim's bisection algorithm — but Kim's algorithm only solves the first problem instance. Before we do so, however, we first introduce in Chaps. 2 and 3, respectively, the concepts from graph theory and quantum computation necessary in order to understand our application of Dürr and Høyer's algorithm in Chap. 4. Along the way, we try to dispose of as much complexity as possible, making an effort to simplify our problem and succeeding in part. Be that as it may, the problem's hidden structure will continue to be a mystery.

Chapter 2

Graph Theory

2.1 Graphs

For graphs, we adopt the definitions and notation of Godsil and Royle [24].

Definition 3 (Graph [24, p. 1])**.** A *graph* G consists of a *vertex* set $V(G)$ or V and an *edge* set $E(G)$ or E. An edge (i, j) is an unordered pair of distinct vertices i and j of G. If (i, j) is an edge, we say that i and j are *adjacent* or that i is a *neighbour* of j, and denote this by writing $i \sim j$. A vertex is *incident* with an edge if it is one of the two vertices of the edge. Also, unless stated otherwise, $N = |V|$ and $M = |E|$.

Definition 4 (Complete Graph [24, p. 2])**.** A graph is called *complete* if every pair of vertices is adjacent, and the complete graph on N vertices is denoted K_N.

We want two graphs G and H to be equal if and only if their topologies are equal, irrespective of how their vertices are labeled. In the following definition, we define when two graphs G and H are isomorphic; we will consider isomorphic graphs as if they were equal.

Definition 5 (Isomorphic Graphs [24, p. 2])**.** Two graphs G and H are *isomorphic* if there is a bijection, φ say, from $V(G)$ to $V(H)$ such that $i \sim j$ in G if and only if $\varphi(i) \sim \varphi(j)$ in H. We say that φ is an *isomorphism* from G to H. If G and H are isomorphic, then we write $G \cong H$.

We will mostly deal with *simple graphs*: graphs as defined in Definition 3, i.e. edges are *unweighted* and *undirected*, and the graph does not contain *loops* (i.e., edges that connect a vertex to itself) or a pair of vertices between which multiple edges exist. In addition to simple graphs, we will concern ourselves with *directed graphs*, as defined below.

Definition 6 (Directed Graph [24, p. 2])**.** A *directed graph* G consists of a vertex set V and an *arc* set $R(G)$ or R, where an arc or *directed edge* is an ordered pair of distinct vertices.

Both simple graphs and directed graphs can be either *connected* or *disconnected*, but we will only consider connected graphs.

Definition 7 (Connected Graph [24, p. 4])**.** If there is a path between any two vertices of a graph G, then G is *connected*. Otherwise, G is *disconnected*.

Finally, whenever we draw graphs, such as in Fig. 1.1, the labelling of vertices is arbitrary, as is the position of the vertices and edges.

2.1.1 Representations of Graphs

The two standard graph representations are the *adjacency-list representation* and the *adjacency-matrix representation*. The former, which comprises a collection of lists, one for each vertex $v \in V$, each list consisting of v's neighbours, is the most natural choice if the graph contains only a few edges, relative to the number of vertices. The amount of memory required by the adjacency-list representation is $\Theta(N + M)$.[1]

The latter, the adjacency-matrix representation, is simply an N by N matrix denoted $A(G)$ or A in which A_{ij} is 1 if i is adjacent to j and 0 otherwise. The adjacency-matrix representation, unlike the adjacency-list representation, explicitly stores information about non-existing edges and as a consequence usually requires more memory than the adjacency-list representation, namely $\Theta(N^2)$. The adjacency-matrix of an undirected graph is symmetric.

Although the adjacency-matrix representation has a worse *space complexity* than the adjacency-list representation, the *time complexity* of finding out whether a particular pair of vertices are adjacent is better: we only need to extract the right entry from A, whilst if the graph is represented as a collection of adjacency lists, we need to iterate over a whole adjacency list to work out whether the pair of vertices are neighbours.

We will use the adjacency-matrix representation, because it is closely related to the Laplacian matrix mentioned before, which again relates to the graph's algebraic connectivity and effective graph resistance, but also because we will deal with other matrices associated with graphs, in particular the *transition matrix* of the *random walk* on the graph and the *Moore-Penrose matrix inverse* of the *Laplacian*.

2.2 Algebraic Connectivity

In 1973, Miroslav Fiedler coined the term *algebraic connectivity* and defined it as the second smallest eigenvalue $\lambda_2(G) = a(G)$ of the Laplacian of G [21, p. 298]. The corresponding eigenvector is called the *Fiedler vector*. The

[1] A function $f(n)$ is $\Theta(g(n))$ if there exist positive constants c_1, c_2, and n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.

Laplacian $Q(G)$ or Q of a graph G is the difference between the *diagonal matrix of valencies* of G denoted $\Delta(G)$ or Δ and its adjacency matrix A. That is,

$$Q = \Delta - A. \tag{2.1}$$

The Laplacian, in addition to a graph's adjacency matrix and *incidence matrix*, captures some interesting characteristics of the graph it is associated with. In particular, if an arbitrary row and column i are deleted from the Laplacian of a graph G, the determinant of the resulting matrix equals the number of *spanning trees* of G. A proof and more formal formulation of the theorem can be found in [24, p. 282].

Algebraic connectivity, just as *edge connectivity* and *vertex connectivity* (see Sect. 2.2.1), says something about how well-connected or robust a graph is. For example, if and only if a graph's algebraic connectivity is zero, the graph is not connected [21, p. 298]. And the larger a graph's algebraic connectivity, the larger the minimum number of edges that need to be removed from the graph to create a bipartition[2] [14, p. 3]. Lastly, although there are many more examples of the relation between algebraic connectivity and robustness, if a graph's algebraic connectivity is large, the random walk on the graph will rapidly escape any subset of states [14, p. 5].

The eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_N$ of the Laplacian are all real and nonnegative [24, p. 280], and we will refer to them as the *spectrum* of the Laplacian. The smallest eigenvalue λ_1 is zero for any graph because $Q\mathbf{1} = \mathbf{0}$ where $\mathbf{0}$ and $\mathbf{1}$ denote the N-dimensional column vector of which each component is 0 and 1, respectively. Indeed, each component of $Q\mathbf{1}$ is the sum of the valency of a vertex i, i.e. the number of neighbours of i or its *degree*, and minus the number of edges from i to every other vertex, which is zero. Therefore, algebraic connectivity is sometimes also referred to as the smallest non-trivial or non-zero eigenvalue of the Laplacian.

Example 1. Consider the random, undirected, unweighted graph G in Fig. 2.1. Its Laplacian Q, diagonal matrix of valencies Δ, and adjacency matrix A are shown in Eq. 2.2. The set of vertices V is ordered $0, 1, \ldots, 5$. That is, the $(i+1)$th row or column of each of these matrices corresponds to the vertex with label i. Equation 2.3 gives the spectrum of the Laplacian.

The order of the rows and columns of the Laplacian is only relevant if we want to draw the graph and relate the entries of the Laplacian to actual vertices. The ordering does not affect the spectrum of the Laplacian, which includes the graph's algebraic connectivity.

In order to find the algebraic connectivity of G, we compute the eigenvalues of Q by solving the *characteristic equation* $\det(Q - \lambda I) = 0$ for λ. The smallest non-zero eigenvalue of Q is $\lambda_2 = 2$.

[2] A *bipartition* (V_1, V_2) of a graph G is a tuple of sets V_1 and V_2 that partition $V(G)$ such that no two vertices from the same set are adjacent [3, p. 7].

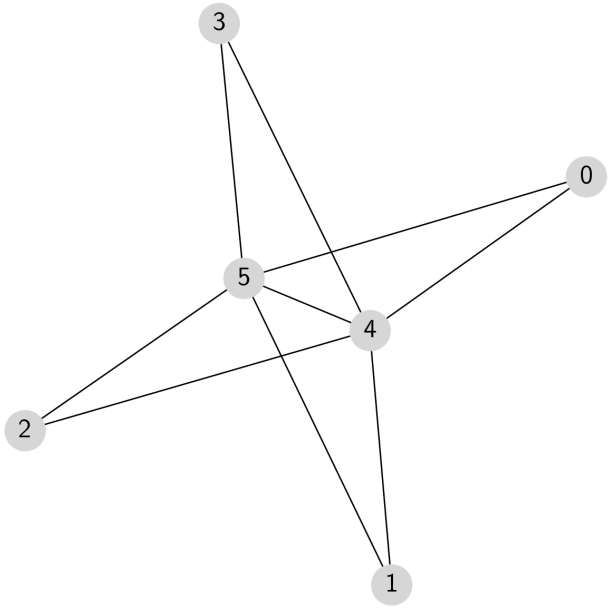


Figure 2.1: A simple graph G on $N = 6$ vertices.

$$
Q = \begin{bmatrix} 2 & 0 & 0 & 0 & -1 & -1 \\ 0 & 2 & 0 & 0 & -1 & -1 \\ 0 & 0 & 2 & 0 & -1 & -1 \\ 0 & 0 & 0 & 2 & -1 & -1 \\ -1 & -1 & -1 & -1 & 5 & -1 \\ -1 & -1 & -1 & -1 & -1 & 5 \end{bmatrix}
$$

$$
\Delta = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{bmatrix} \quad A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{2.2}
$$

The spectrum of Q is

$$
0,\ 2,\ 2,\ 2,\ 6,\ 6. \tag{2.3}
$$

2.2.1 Relation to Edge Connectivity and Vertex Connectivity

The *edge connectivity* of a connected graph G is the minimum number of edges that need to be removed from G in order to disconnect G. The graph in Fig. 2.1, for example, has edge connectivity 2 because removing edges $(0, 4)$ and $(0, 5)$ makes the graph a disconnected graph and it is not possible to disconnect the graph by removing a single edge.

Similarly, the *vertex connectivity* of a connected graph G is the minimum number of vertices that need to be removed from G in order to lose connectivity. If a vertex $i \in V$ is removed, all edges $(i, j) \in E$ that contain i are removed as well. Removing vertices 4 and 5 from the graph in Fig. 2.1 makes it disconnected, as doing so makes it impossible to travel from 0 to 1, for example. It is not possible to disconnect the graph by removing only a single vertex, and therefore the vertex connectivity of the graph is 2.

How do edge connectivity and vertex connectivity relate to algebraic connectivity? Fiedler [21, p. 303] showed that for a non-complete graph G with algebraic connectivity $a(G)$ and vertex connectivity $v(G)$, we have

$$a(G) \leq v(G). \tag{2.4}$$

It is also known that for any graph G of at least two vertices,

$$v(G) \leq e(G) \tag{2.5}$$

where $e(G)$ denotes the edge connectivity of G. A proof of Eq. 2.5 can be found in [7, p. 73].

2.3 Effective Graph Resistance

A *metric* is a function $g(x, y)$ that describes the *distance* between any two members of a given set, the *metric space*. A metric satisfies the following axioms:

$$\begin{aligned} g(x, y) &\geq 0 \\ g(x, x) &= 0 \\ g(x, y) &= g(y, x) \\ g(x, z) + g(z, y) &\geq g(x, y) \end{aligned} \tag{2.6}$$

In their 1993 paper [34], Klein and Randić share their observation that there exists no metric that depends both on

- how far removed from each other two vertices are, and is larger if two vertices are further apart; and on
- how many different paths exist that start in one of the two vertices and end in the other, and decreases as the number of paths increases.

Nevertheless, they foresee applications in chemistry, and propose a new metric "with the characteristic of multiple route distance diminishment" and call it *effective resistance* [34, p. 82].

The function Klein and Randić defined describes the distance between any two vertices $(i, j) \in V \times V$, i.e. the metric space is the set of vertices $V(G)$ of a connected graph G. The effective resistance R_{ij} between two

vertices i and j equals the electrical resistance between the two vertices, if we imagine the graph as an electrical network where each edge is replaced by a resistor of $1\,\Omega$ and a voltage is applied across i and j.

The *effective graph resistance* R_G or *Kirchhoff index* of a connected graph G on N vertices is the sum of the effective resistances between all pairs of vertices, i.e.

$$R_G = \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} R_{ij}. \tag{2.7}$$

One way to compute R_{ij}, and with that R_G, is by simply computing the electrical resistance between i and j using the fact that

- the total resistance that n resistors in series give rise to, equals

$$R_1 + R_2 + \cdots + R_n = R_{total} \tag{2.8}$$

- and that, the reciprocal of the total resistance that n resistors in parallel give rise to, equals

$$\frac{1}{R_1} + \frac{1}{R_2} + \cdots + \frac{1}{R_n} = \frac{1}{R_{total}}. \tag{2.9}$$

Observe that these two relations, respectively, lead exactly to the two properties Klein and Randić were looking for, given at the beginning of this section: a consequence of Eq. 2.8 is that longer paths will lead to a larger effective resistance compared with shorter paths, whilst because of Eq. 2.9, the smaller the effective resistance, the more paths exist between the two vertices.

Not just all paths between i and j count in Eq. 2.9, only parallel paths and parallel subpaths are included. We adopt Godsil and Royle's definition of a *path*, which runs as follows:

Definition 8 (Path [24, p. 2])**.** A *path* of length r from i to j is a sequence of $r+1$ distinct vertices, starting with i and ending with j such that consecutive vertices are adjacent.

The definition above is in accordance with the line of reasoning behind Eq. 2.9, because it invalidates paths that contain cycles. For example, consider the graphs in Fig. 2.2. According to the definition, there exists only one path between $i = 1$ and $j = 2$ in Fig. 2.2b, namely $\textcircled{1} \to \textcircled{4} \to \textcircled{2}$.[3] The sequence $\textcircled{1} \to \textcircled{4} \to \textcircled{0} \to \textcircled{3} \to \textcircled{4} \to \textcircled{2}$, on the other hand, is not a path according to the definition because it contains $\textcircled{4}$ twice. Indeed, cycles such as $\textcircled{4} \to \textcircled{0} \to \textcircled{3} \to \textcircled{4}$ in Fig. 2.2b should not be part of a path between two distinct vertices i and j, because

[3] We write $\textcircled{i} \to \textcircled{j}$ if and only if $i \sim j$.

- they necessarily start and end at the same vertex k; and therefore
- in order for a cycle starting and ending at k to be part of a path between i and j, a sequence $\textcircled{i} \to \cdots \to \textcircled{k} \to \cdots \to \textcircled{j}$ must exist; which implies that
- the sequence $\textcircled{i} \to \cdots \to \textcircled{k} \to \cdots \to \textcircled{j}$ is a path as well; and as a consequence
- if k would be removed from the graph, *both* paths would no longer be able to reach j, which makes them nonparallel in the sense of Eq. 2.9. Observe that a cycle does not increase the effective resistance R_{ij} between i and j, in spite of the fact that it does increase the length of a sequence from i to j, because the *potential difference* across k and k is 0, which means that no current flows through the cycle, and as a result the cycle $\textcircled{k} \to \cdots \to \textcircled{k}$ "adds" zero resistance to the sequence that it is part of.

Definition 9 (Parallel Path)**.** A path $\textcircled{i} \to \cdots \to \textcircled{j}$ is *parallel* to a path $\textcircled{i'} \to \cdots \to \textcircled{j'}$ if and only if

- $i = i'$ and $j = j'$; and
- there does not exist a vertex k such that $k \neq i$ and $k \neq j$, and that is part of both paths. That is, with their endpoints removed, the paths need to be *vertex-disjoint*.

There are at least three other, not at all obvious ways to compute R_G for a given graph G, all of which will be presented in the following three sections.

2.3.1 Relation to the Eigenvalues of the Laplacian

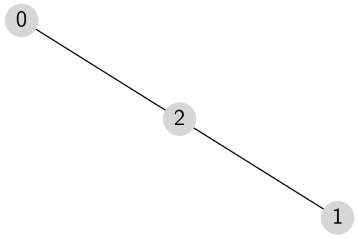
Klein and Randić have shown [34, p. 86], in the same paper that introduced effective resistance, that

$$R_G = N \operatorname{Tr}\left(Q^+\right) \tag{2.10}$$

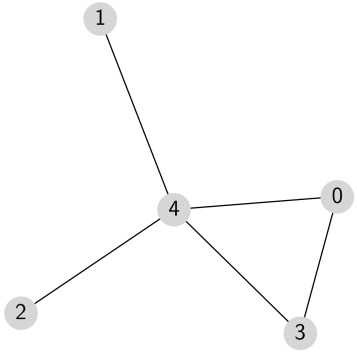
where Q^+ denotes the inverse of the Laplacian Q within the subspace orthogonal to the eigenvector $\boldsymbol{x}$ of Q, such that $Q\boldsymbol{x} = \mathbf{0}$. In [47, p. 206], Van Mieghem proves that

$$R_G = N \sum_{k=2}^{N} \frac{1}{\lambda_k} \tag{2.11}$$

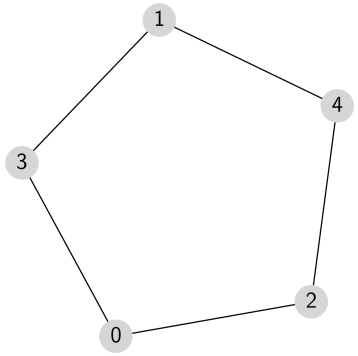
where $0 = \lambda_1 < \lambda_2 \leq ... \leq \lambda_N$ are the eigenvalues of Q (see Sect. 2.2). Equation 2.11 makes it possible to relate a graph's algebraic connectivity $a(G) = \lambda_2(G)$ to its effective graph resistance R_G. It shows that, as λ_2 increases, R_G decreases, which explains why we need to maximise the former and minimise the latter, to increase the robustness of the graph (see Definition 2).



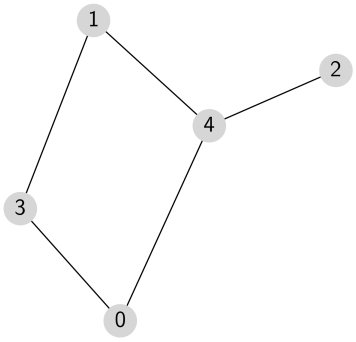
(a) One path from 0 to 1.



(b) One path from 1 to 2, the cycle $\textcircled{4} \to \textcircled{0} \to \textcircled{3} \to \textcircled{4}$ does not introduce additional paths.



(c) Two parallel paths between all pairs of distinct vertices.



(d) Two parallel subpaths from 4 to 3, if we consider R_{23}, which can be reduced to a single subpath and makes the graph equal to the graph in Fig. 2.2a.

Figure 2.2: Four small simple graphs to illustrate the notion of a parallel path, in the sense of Eq. 2.9.

2.3.2 Relation to the Random Walk on the Graph

What follows is a truly fascinating relationship between the effective resistance R_{ij} between two vertices i and j, and the random walk on G. Chandra et al. [10, p. 317] found that

$$R_{ij} = \frac{1}{2M} C_{ij} \tag{2.12}$$

where C_{ij} denotes the *commute time* between i and j, i.e. the sum of (i.) the expected length $\mathbb{E}[T_{ij}]$ of a random walk starting in i and ending in j; and (ii.) the expected length $\mathbb{E}[T_{ji}]$ of a walk in opposite direction. That is,

$$C_{ij} = \mathbb{E}[T_{ij}] + \mathbb{E}[T_{ji}]. \tag{2.13}$$

A random walk is a stochastic process. A walker starts in a vertex $X_{t=0} = i$. Each time step ($t := t + 1$), the walker moves to one of the vertices adjacent to its current position. The probability p_j of moving to a particular neighbour j is the same for all neighbours, and therefore equals $p_k = 1/d_k$ where d_k denotes the degree of k, the walker's current position.

Equation 2.12 is interesting not just because it relates random walks to electrical networks. Connections between the two have been found before [15, p. 2]. What makes Eq. 2.12 particularly interesting is that cycles, such as ④ → ⓪ → ③ → ④ in Fig. 2.2b, affect the random walk on the graph but *not* the effective resistance, as argued in the preceding section.

To illustrate, consider once more the graphs in Fig. 2.2. The effective resistance R_{12} between 1 and 2 in the graph in Fig. 2.2b is the same as the effective resistance R_{01} between 0 and 1 in the graph in Fig. 2.2a: both are $2\,\Omega$. However, due to the cycle in Fig. 2.2b, the commute time $C_{12} = 20$ in Fig. 2.2b, but $C_{01} = 8$ in Fig. 2.2a. It is striking that a factor as simple as $1/(2M)$ in Eq. 2.12 is able to sort of "correct for" the range of additional paths introduced by the cycle, each of which is a possible instance of the random walk on the graph.

2.3.3 Relation to the Number of Spanning Trees

It is worthwhile to mention that not only does there exist a relation between the effective resistance R_{ij} between two vertices i and j, electrical networks, eigenvalues of the Laplacian, and random walks on the graph; R_{ij} can also be expressed in terms of the number of spanning trees[4] $\tau(G)$ of G, that is

$$R_{ij} = \frac{\tau(G \cdot (i,j))}{\tau(G)} \quad \text{if} \quad i \neq j \tag{2.14}$$

where $\tau(G \cdot (i,j))$ denotes the number of spanning trees of G after vertices i and j have been merged, resulting in a vertex $k \in V(G \cdot (i,j))$ that is adjacent

[4] A *spanning tree* of a graph is a connected, spanning subgraph without cycles [24, p. 4].

to both the neighbours of i *and* the neighbours of j (in the original graph G). It is possible and absolutely fine that, as consequence of merging i and j, $G \cdot (i,j)$ now contains pairs of vertices between which multiple edges exist, or that loops are introduced. Different edges give rise to different spanning trees, and both trees count, but loops are never part of a spanning tree as follows directly from the definition of a tree (a connected graph without cycles).

Equation 2.14 was found by Bapat, and a proof can be found in [4, p. 496]. Bapat's formulation is slightly different from Eq. 2.14, however: Let $Q\{i,j\}$ denote the Laplacian of a graph G with rows i, j and columns i, j removed. Equivalently, $Q\{i\}$ denotes the submatrix resulting from the removal of the ith row and column of the Laplacian. Then,

$$R_{ij} = \frac{\tau(G \cdot (i,j))}{\tau(G)} = \frac{\det Q\{i,j\}}{\det Q\{i\}}. \tag{2.15}$$

The fact that $\tau(G) = \det Q\{i\}$, where i is an *arbitrary* vertex of G, is a well-known result known as the *matrix-tree theorem* or *Kirchhoff's matrix-tree theorem*. For a proof, see for example [24, p. 282]. A proof that $\tau(G \cdot (i,j)) = \det Q\{i,j\}$ can be found in [48, p. 25].

Observe that $\det Q\{i,j\}$ does not change if we add (i,j) to the graph G of which Q is the Laplacian. After all, what is the effect on Q of adding (i,j) to G? Four entries will change, assuming $(i,j) \in \bar{E}(G)$ where $\bar{E}(G)$ denotes the set of edges *not* in G (see Sect. 4.3.1):

$$\begin{aligned} Q_{ii} &:= Q_{ii} + 1 \quad & Q_{ij} &:= -1 \\ Q_{ji} &:= -1 & Q_{jj} &:= Q_{jj} + 1 \end{aligned} \tag{2.16}$$

However, none of the entries in Eq. 2.16 survives the removal of rows i, j and columns i, j. Therefore, let $Q^{+e} = Q(G + e)$, then $Q\{i,j\} = Q^{+(i,j)}\{i,j\}$ and as a result

$$\frac{R_{ij}^{+(i,j)}}{R_{ij}} = \frac{\det Q^{+(i,j)}\{i,j\}}{\det Q^{+(i,j)}\{i\}} \frac{\det Q\{i\}}{\det Q\{i,j\}} = \frac{\det Q\{i\}}{\det Q^{+(i,j)}\{i\}} \tag{2.17}$$

and

$$R_{ij}^{+(i,j)} = \frac{\tau(G)}{\tau(G + (i,j))} R_{ij} = \frac{\tau(G \cdot (i,j))}{\tau(G + (i,j))} \tag{2.18}$$

where also R_{ij}^{+e} denotes the effective resistance R_{ij} between $i, j \in V(G + e)$.

Equation 2.18 can also be derived directly from Eq. 2.15, because if $G := G + (i,j)$ we immediately find the denominator, and it is also not hard to see that $\tau(G + (i,j) \cdot (i,j)) = \tau(G \cdot (i,j))$ because merging i and j turns the edge (i,j) into a loop, which as explained before is never part of a spanning tree.

Equation 2.18 is interesting because it provides for an explanation of the *change* of R_{ij} as a consequence of adding (i,j) to G. It shows that the

relative increase in the number of spanning trees, caused by the addition of (i, j), is inversely proportional to the relative increase of the effective resistance between i and j.

It sounds great, but this insight alone is not enough to solve our problem (the second instance, to be precise). Ultimately, we are not interested in what changes R_{ij}. Instead, we want to know what makes R_G change. Expanding R_{G+e}, where $e \in \bar{E}(G)$, gives

$$R_{G+e} = \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} R_{ij}^{+e} = R_{01}^{+e} + R_{02}^{+e} + \dots . \tag{2.19}$$

The terms of R_{G+e} fall under three categories. Let $e = (k, l) \in \bar{E}(G)$. Then

1. there is exactly one term R_{kl}^{+e}, which as argued above equals $\tau(G \cdot e)/\tau(G + e)$; and
2. there are terms R_{ij}^{+e} for which $(i, j) \in E(G)$; and finally
3. there are also terms R_{ij}^{+e} for which $(i, j) \in \bar{E}(G) \setminus \{e\}$.

What do we know about the terms in the last two categories? The following very simple relationship will come in useful. Let $e \in E(G)$, then

$$\tau(G) = \tau(G - e) + \tau(G \cdot e). \tag{2.20}$$

Equation 2.20 can be used to compute, recursively, the number of spanning trees of G, because $G - e$ and $G \cdot e$ both contain fewer edges than G. This method for counting spanning trees is called the *deletion-contraction method*; a clear and intuitive explanation of Eq. 2.20 can be found in [11, p. 1]. A term in category two, expressed in terms of the number of spanning trees of "mutants" of G, looks as follows (see Eq. 2.15):

$$R_{ij}^{+e} = \frac{\tau(G + e \cdot (i, j))}{\tau(G + e)}. \tag{2.21}$$

Since $(i, j) \in E(G)$, we are allowed to apply Eq. 2.20. We find

$$R_{ij}^{+e} = \frac{\tau(G + e) - \tau(G + e - (i, j))}{\tau(G + e)} = 1 - \frac{\tau(G + e - (i, j))}{\tau(G + e)} \quad \text{if} \quad (i, j) \in E(G). \tag{2.22}$$

Before we can apply Eq. 2.20 to a term in category three, we need to use a little trick. Equation 2.21 also holds if $(i, j) \in \bar{E}(G) \setminus \{e\}$, but we can no longer directly apply Eq. 2.20. However, observe that $\tau(G + e \cdot (i, j)) = \tau(G + e + (i, j) \cdot (i, j))$ because evidently, adding (i, j) to $G + e$ makes no difference to the number of spanning trees because we merge i and j directly after doing so, which effectively undoes the addition. We get

$$R_{ij}^{+e} = \frac{\tau(G + e + (i, j) \cdot (i, j))}{\tau(G + e)} \quad \text{if} \quad (i, j) \in \bar{E}(G) \setminus \{e\} \tag{2.23}$$

and are now able to apply Eq. 2.20 to the nominator of Eq. 2.23, to find

$$\begin{aligned} R_{ij}^{+e} &= \frac{\tau(G+e+(i,j)) - \tau(G+e)}{\tau(G+e)} \\ &= -1 + \frac{\tau(G+e+(i,j))}{\tau(G+e)} \quad \text{if} \quad (i,j) \in \bar{E}(G) \setminus \{e\}. \end{aligned} \tag{2.24}$$

Note that Eqs. 2.22 and 2.24 are similar. Their sum R_{G+e} becomes

$$\begin{aligned} R_{G+e} &= \overbrace{\frac{\tau(G \cdot e)}{\tau(G+e)}}^{\text{1st cat.}} + \overbrace{M + \sum_{(i,j) \in E(G)} -\frac{\tau(G+e-(i,j))}{\tau(G+e)}}^{\text{2nd cat.}} \\ &+ \overbrace{M + 1 - \binom{N}{2} + \sum_{(i,j) \in \bar{E}(G) \setminus \{e\}} \frac{\tau(G+e+(i,j))}{\tau(G+e)}}^{\text{3rd cat.}} \\ &= 2M + 1 - \binom{N}{2} \\ &+ \frac{1}{\tau(G+e)} \left(\tau(G \cdot e) + \sum_{(i,j) \in \bar{E}(G) \setminus \{e\}} \tau(G+e+(i,j)) - \sum_{(i,j) \in E(G)} \tau(G+e-(i,j)) \right) \end{aligned} \tag{2.25}$$

where $N = |V(G)|$ and $M = |E(G)|$.

Equation 2.25 gives some insight into how R_G changes, if we add an edge to G. It shows that, as the number of spanning trees of $G + e$ increases, the effective graph resistance decreases. Unfortunately, however, it appears to be impossible to eliminate the two summations, and a lot of complexity remains.

One might wonder whether the edge $e' \in \bar{E}(G)$ that maximally decreases R_G, if added to G, also maximally increases the number of spanning trees of G. The answer to this question is, no, not necessarily, although often the two do indeed coincide. It is a pity that the two problems are not the same, because if they were, a solution would be readily available: Tsen et al. [22] have formulated an algorithm to find the most "vital" edge in $E(G)$: "the edge whose deletion results in a maximum decrease in the number of spanning trees." [22, p. 600]. Their algorithm can also be used to instead find the edge in $\bar{E}(G)$ whose addition maximally increases the number of spanning trees. The complexity of their algorithm is dependent on, and equal to, the complexity of matrix multiplication, which was $\mathcal{O}(N^{2.376})$ when they published their paper in 1994, and since then has only slightly improved to become $\mathcal{O}(N^{2.3728639})$ in 2014 (see [35]).

2.3.4 Convergence of the Effective Resistance

It turns out that if G is a large *random geometric graph*, then

$$R_{ij} \approx \frac{1}{d_i} + \frac{1}{d_j} \tag{2.26}$$

where d_i and d_j denote the degree of the vertices i and j, respectively. This surprising result by Luxburg et al. [38] seems to undermine the idea that R_{ij} says something about how well-connected i and j are. After all, d_i and d_j reveal the graph's *local* structure, whereas the connectivity of i and j depends on the *global* structure of the graph (i.e., on the length and amount of paths from i to j). Another way to look at it, is to say that apparently, if the graph is large, then the only relevant information about the paths from i to j, with respect to R_{ij}, is (i.) the number of directions to depart from; and (ii.) the number of directions to arrive in. Evidently, all intermediary directions to take have a negligible contribution to R_{ij}.

The proof of Eq. 2.26 is quite involved, and can be found in the supplement of [38]. The intuition behind the proof is as follows. Consider the random geometric graph G_g in Fig. 2.3. What would $R_{9,30}$ look like? Since G_g is well-connected and relatively large (i.e., it consists of many more vertices than just 9 and 30), a lot of parallel (sub)paths from 9 to 30 are expected. And as the number of parallel paths increases, the resistance encountered by electrons travelling from 9 to 30 decreases.

However, the degrees of 9 and 30 restrict the number of parallel paths between the two vertices. Vertex 9, for example, has a degree of 7, which means that at most 7 parallel paths from 9 to any other vertex can exist. The number of parallel paths and subpaths between vertices *adjacent to* 9 and vertices *adjacent to* 30, on the other hand, can be — and probably is — much larger.

Consequently, we can illustrate the connection between 9 and 30 by the graph G'_g in Fig. 2.4, in which G_g with vertices 9 and 30 removed has been reduced to a single edge that connects vertices i and j, each of which is also connected to either 9 or 30. The vertices i and j represent the aggregate of the neighbours of 9 and 30, respectively.

The resistance between i and j is negligible as a result of the abundance of parallel paths between the neighbours of 9 and 30. As a result, the effective resistance $R_{9,30}$ is roughly equal to

$$R_{9,i} + R_{i,j} + R_{j,30} = \left(d_9 \frac{1}{1}\right)^{-1} + \left(d_{30} \frac{1}{1}\right)^{-1} = \frac{1}{d_9} + \frac{1}{d_{30}} \approx R_{9,30}\,. \tag{2.27}$$

The graph in Fig. 2.3 is not large enough for the effective resistance to actually converge to Eq. 2.26. Experiments by Luxburg et al. show that, for

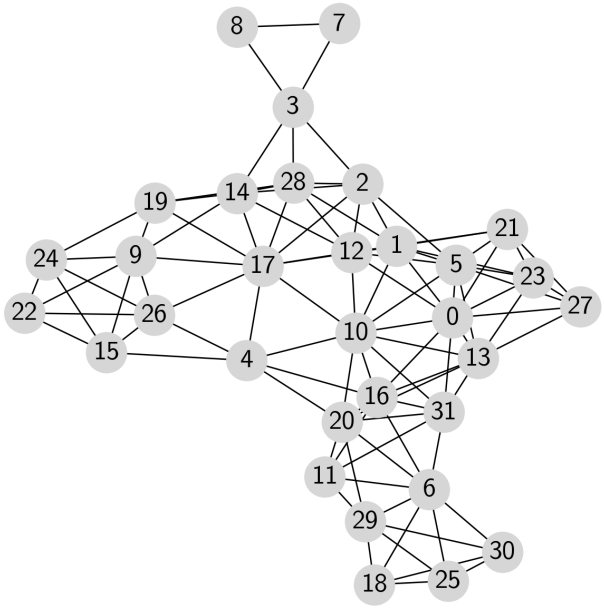


Figure 2.3: A random geometric graph G_g on $N = 32$ vertices. To generate G_g, each vertex has been placed uniformly at random in a unit square. Vertices within a Euclidean distance of each other less than or equal to $r = 0.3$ have been connected. Note that, for the sake of clarity, the drawing does not correspond to the "original" unit square that was used to generate the graph. Drawing the unit square, with the vertices in it, would have resulted in a very cluttered picture.

random geometric graphs of the same kind as the graph in Fig. 2.3,[5] the maximum relative error $\max\{|R_{ij} - 1/d_i - 1/d_j|/R_{ij}\}$ becomes about 1% only if N has order of magnitude 1000 or larger. In addition to the number of vertices N, the maximum relative error depends on the *dimension* of the space in which the vertices are placed (e.g., G_g was generated from a two-dimensional space), on the graph's *density* (i.e., the relative number of edges), and on its *clusteredness*. As one might expect, the higher the dimension, the better the graph is connected, and the smaller the maximum relative error. Similarly, a high density and a low clusteredness will result in a small error.

It may not always be possible to actually reduce a graph, or even a part of it, to a single edge and obtain a graph such as G'_g in Fig. 2.4. Indeed, although the explanation above of Eq. 2.26 is entirely dependent on the assumption that it *is* possible, Luxburg et al. do not need the assumption to

[5]The authors call these kind of random geometric graph *ε-graphs*, where $\varepsilon = r$ is the maximum Euclidean distance between a pair vertices in order for them to become neighbours.

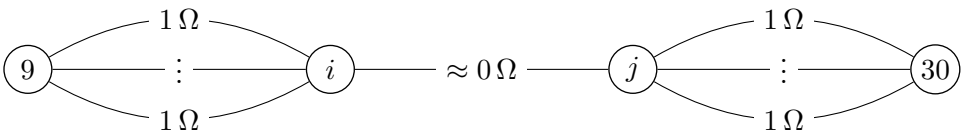


Figure 2.4: The reduced graph G'_g to illustrate Eq. 2.26.

arrive at their result. It would be interesting, however, to find out whether it is also possible to prove that Eq. 2.26 holds by proving that the assumption is justified, or at least justified under certain conditions. In the next few paragraphs we will briefly discuss what an inquiry into this problem would lead to.

First, how can a graph be reduced? That is, what kind of reductions are allowed? There are four possible reductions, all of which can be derived from Eqs. 2.8 and 2.9. Archdeacon et al. [2, p. 2] formulate them as follows:

Loop reduction. Remove a loop.

Degree-one reduction. Remove a degree-one vertex and its incident edge.

Series reduction. Remove a degree-two vertex k and its incident edges (i, k) and (k, j), and add a new edge (i, j).

Parallel reduction. Remove one of a pair of parallel edges.

Observe that both loops and degree-one vertices are examples of cycles, as explained in Sect. 2.3. According to Definition 8, a sequence of vertices that consists of a loop $\cdots \to (k) \to (k) \to \cdots$ or a degree-one vertex $\cdots \to (k) \to (i) \to (k) \to \cdots$ cannot be a path.

In addition to the four reductions above, two useful *transformations* exist: the ΔY*-transformation* and its converse the $Y\Delta$*-transformation* (left to right and right to left in Fig. 2.5, respectively). The latter actually reduces the graph, as it removes a vertex, which implies that the former does the opposite, namely add a vertex to the graph. Notice that neither transformation changes the number of edges in the graph. If no reduction is possible, one or more $\Delta Y/Y\Delta$-transformations might transform the graph into a graph that can be further reduced.

Definition 10 ($Y\Delta Y$-Reducible [2, p. 3])**.** A connected graph is $Y\Delta Y$*-reducible* if it can be reduced to a single vertex by a sequence of loop, degree-one, series, or parallel reductions and $\Delta Y/Y\Delta$-transformations.

Not all graphs are $Y\Delta Y$-reducible, a counterexample is K_6, the complete graph on $N = 6$ vertices [2, p. 3]. However, $Y\Delta Y$-reducibility is not really the graph property that we are interested in. We do not want to reduce the *whole* graph to a single vertex, we want to reduce only a part of it, to obtain a graph similar to G'_g.

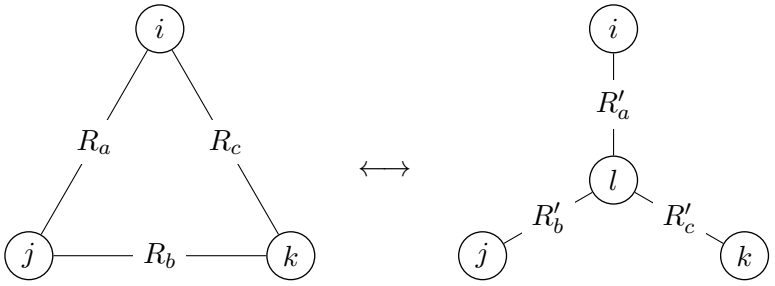


Figure 2.5: The ΔY*-transformation*, where $R'_a = (R_a R_c)/(R_a + R_b + R_c)$; $R'_b = (R_a R_b)/(R_a + R_b + R_c)$; and $R'_c = (R_b R_c)/(R_a + R_b + R_c)$.

At first sight, the notion of N*-terminal reducibility* is what we are looking for. If a graph G can be reduced to a graph G' on N vertices without removing any of N designated vertices $v_t \in V_t \subset V(G)$ or *terminals* in the process, it is N*-terminal reducible*. If a graph is 4-terminal reducible, does that mean that it can be reduced to a graph such as the graph in Fig. 2.4? Unfortunately, the answer is no, because it is not possible to identify all four terminal vertices k, i, j, and l in advance.[6] In particular, the vertices i and j, each of which represents the aggregate of the neighbours of k and l, respectively, probably do not even exist before a reduction has taken place.

To conclude this brief interlude, proving that Eq. 2.26 holds for some specified class of graphs, because each graph that belongs to the class can be reduced to a graph similar to the graph in Fig. 2.4, might be an interesting exercise. Finally, it is worth mentioning that all planar graphs[7] are 2-terminal reducible (i.e., they can be reduced to a single edge (i, j) connecting the two terminals i and j) [17, p. 21]. It goes without saying that the weight of (i, j) in the reduced graph equals R_{ij}.

Finally, observe that if Eq. 2.26 holds, the problem of finding the edge that will maximally decrease the graph's effective graph resistance becomes fairly easy. After all, the effective graph resistance is the sum of the effective resistances between all unordered pairs of vertices (see Eq. 2.7). And if $R_{ij} \approx 1/d_i + 1/d_j$, the terms of R_G no longer depend on each other. Therefore, we need to simply compute, for each unordered pair of vertices (i, j)

$$\left| \frac{1}{(d_i + 1)} + \frac{1}{(d_j + 1)} - \frac{1}{d_i} - \frac{1}{d_j} \right| \tag{2.28}$$

and find the maximum. Connecting the two vertices that correspond to the maximum will maximally decrease the graph's effective graph resistance. Figure 2.6 shows that, if Eq. 2.28 were to be used as a heuristic to find the best edge to add, the best edges according to the heuristic will be those that

[6] In Fig. 2.4, $k = 9$ and $l = 30$.

[7] A graph is *planar* if it can be drawn (in a plane) without crossing edges [24, p. 12].

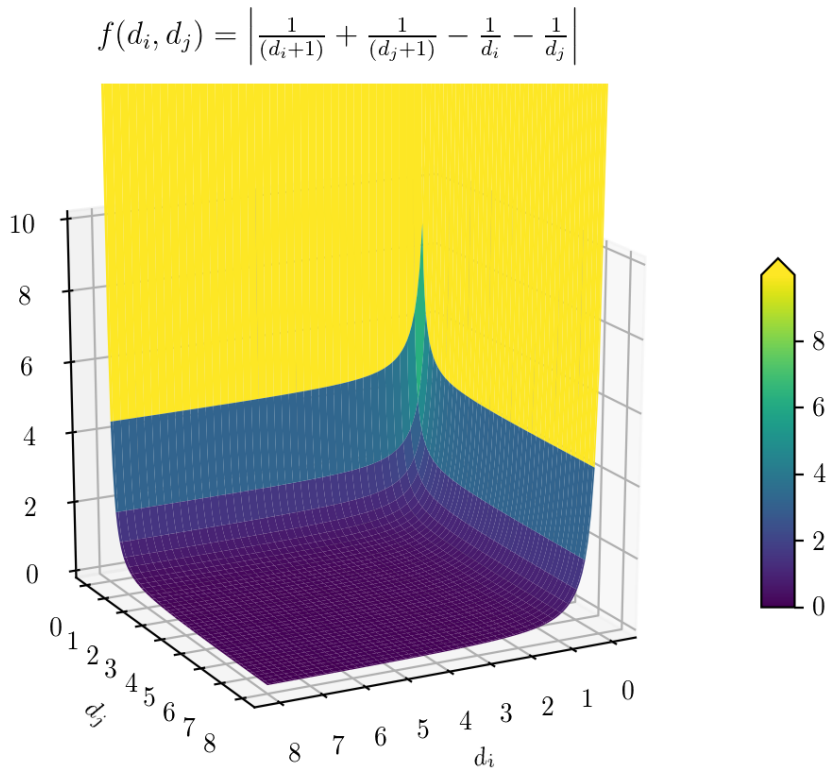


Figure 2.6: The graph of $f(d_i, d_j)$. If one of either d_i or d_j is small, $f(d_i, d_j)$ becomes large.

consist of two vertices i and j such that $d_i = d_j = 1$, if such pair of vertices exists, and otherwise a pair where either d_i or d_j is 1, and if such a pair does not exist, a pair where $d_i = d_j = 2$, etc.

Chapter 3

Quantum Computation

This section is about quantum computation and quantum information, not about quantum theory. Quantum computation is based on *quantum information*, which according to Kaye et al. is "the result of reformulating information theory in [the] quantum framework." [28, p. 38]. The quantum framework is quantum theory, and includes *quantum electrodynamics* and *quantum field theory*.

3.1 Qubits

Normal computers use bits to store information. Quantum computers use *qubits*. The state of a qubit can be described by a vector

$$\alpha_0 |0\rangle + \alpha_1 |1\rangle \tag{3.1}$$

where $|0\rangle = \begin{pmatrix} 1 & 0 \end{pmatrix}^T$ and $|1\rangle = \begin{pmatrix} 0 & 1 \end{pmatrix}^T$ make up the *computational basis* of the two-dimensional *Hilbert space* $\mathcal{H}$ in which the qubit lives. The complex coefficients α_0 and α_1 are called the *amplitudes* of the basis states $|0\rangle$ and $|1\rangle$, respectively. It turns out that the following needs to hold in order for Eq. 3.1 to be a correct abstraction of two-level quantum systems,

$$|\alpha_0|^2 + |\alpha_1|^2 = 1. \tag{3.2}$$

A complex number $re^{i\varphi}$ can be characterised by its *phase factor* or length r and the *phase* or angle φ that it makes with the x-axis. We are not interested in the actual phase of each amplitude, instead the difference between the phases of α_0 and α_1 or *relative phase* is what matters. Therefore, Eq. 3.1 can also be written as

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\varphi}\sin\left(\frac{\theta}{2}\right)|1\rangle \tag{3.3}$$

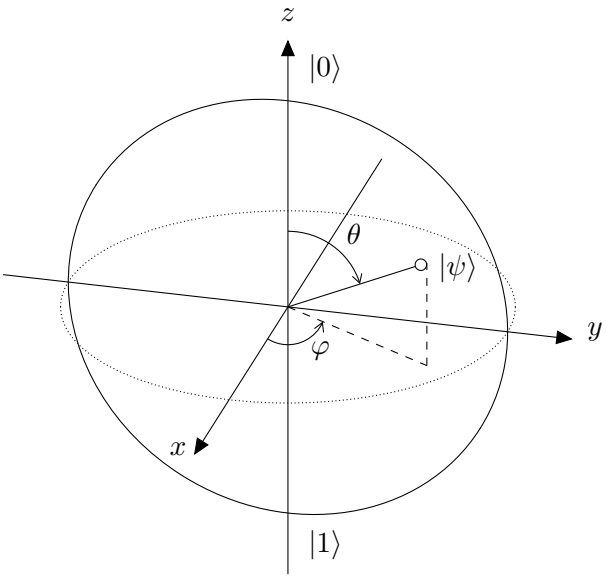


Figure 3.1: The Bloch sphere, visualising the state $|\psi\rangle$.

where indeed the sum of the absolute squares of the amplitudes equals

$$\begin{aligned} |\cos(\theta/2)|^2 + \left|e^{i\varphi}\sin(\theta/2)\right|^2 &= |\cos(\theta/2)|^2 + |\sin(\theta/2)|^2 \\ &= \cos^2(\theta/2) + \sin^2(\theta/2) \\ &= 1. \end{aligned} \tag{3.4}$$

A very useful and commonly used graphical representation of Eq. 3.3 is the *Bloch sphere*, shown in Fig. 3.1. In fact, the reason why θ in Eq. 3.3 has been multiplied by a factor $1/2$ is to make it consistent with the Bloch sphere.

Although the Bloch sphere provides for a convenient visualisation of the state of a single qubit and the effect of single-qubit gates on a qubit's state, it does not enable us to visualise the state of two or more entangled qubits — which is usually far more interesting.

Just as classical computation is about manipulating bits, quantum computation is about manipulating qubits. From quantum theory, we know that the evolution of a quantum system that does not interact with the environment, such as a qubit, is described by a *unitary* operator (for a definition, see for example [45, p. 441]). That is,

$$|\psi_2\rangle = U|\psi_1\rangle \tag{3.5}$$

where U is a unitary operator. Accordingly, *quantum gates* — the foundation of any quantum algorithm — are all nothing but unitary operators. As a consequence, any quantum gate is reversible by definition: applying the same gate twice in succession will leave the quantum system unchanged.

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* of the two vectors. Sometimes, the symbol $\otimes$ will be omitted, i.e. $|\psi_1\rangle \otimes |\psi_2\rangle$ is equivalent to $|\psi_1\rangle\,|\psi_2\rangle$ or even $|\psi_1\psi_2\rangle$. Likewise, the combined state of n qubits is $|\psi_1\rangle \otimes \cdots \otimes |\psi_n\rangle = |\psi_1 \ldots \psi_n\rangle$.

It is not always possible to decompose a combined state into a tensor product. In fact, most of the time, decomposition is not possible. If decomposition is not possible, the qubits are said to be *entangled*.

Finally, given a composite but not necessarily entangled state

$$|\psi\rangle = \sum_{i=1}^{n} \alpha_i\,|i\rangle\,, \tag{3.6}$$

if we were to have at our disposal a measuring apparatus to measure $|\psi\rangle$, measuring $|\psi\rangle$ would give

- i with probability $|\alpha_i|^2$ and if the measurement gives i,
- leave the system in $|i\rangle$. That is, $|\psi\rangle := |i\rangle$.

For a more elaborate introduction to quantum computation, the reader is kindly referred to the standard work on the subject by Nielsen and Chuang [39]. Another excellent explanation of quantum computation is given in Kaye et al.'s *An Introduction to Quantum Computing* [28], which does a surprisingly good job of reviving time and again the intuition that one loses so easily while exploring the field of quantum computation.

3.2 Grover's Quantum Search Algorithm

In 1996, Lov Grover published a paper [26] in which he presents a quantum algorithm that can search 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. Compared to a classical search (i.e., simply iterating the database until the record of interest has been found) which requires N queries in the worst case, Grover's algorithm achieves a quadratic speed-up.[1]

It is quite remarkable that Grover's algorithm has a better time complexity, because it seems that there is no way to improve a classical search. After all, any search algorithm would have to check up on each and every record in the database, which suggest that it is impossible to devise an algorithm that has a query complexity better than $\mathcal{O}(N)$? Although it is indeed *not* possible to devise a *classical* algorithm with a better query complexity, Grover's algorithm shows that it *is* possible to formulate a *quantum* algorithm that requires fewer queries in the worst case. In fact, just as iterating the database is the best one can do classically, it has been proven that it is

[1] A classical search takes time N in the worst case because it is possible that the record that we are looking for is the last record of N records that we inspect.

not possible to come up with a *quantum search algorithm* that has a better query complexity than Grover's algorithm (for a proof, see for example [39, p. 269]).

Superb explanations of Grover's algorithm already exist, e.g. in both of the two books mentioned in the preceding section, and for that reason we will not explain the algorithm in great detail here. Instead, we will try to illustrate the underlying idea.

If a classical search algorithm A queries the oracle f, it "asks" the oracle whether a particular entry $T[i]$ in the database T is the solution, or not. It sends an index i to the oracle, and the oracle returns either 1 if $T[i]$ is a solution and 0 otherwise. Index i is a *bit string*, a series of bits that encodes an integer $0 \leq i \leq n-1$ where n is the number of entries in T. Accordingly, there are n possible queries, but because A can only send *one* bit string per query, it can send no more than *one* index to the oracle, each time.

A quantum search algorithm B queries f in a similar fashion. There is one crucial difference, however: B sends a "qubit string" $|i\rangle$ to the oracle, instead of a series of classical bits. To see why this makes a difference, consider the following.

First, we will assume that there exists a classical algorithm F for computing f, which takes as input a bit string and outputs a single bit. If A wants to query the oracle, it really just runs F. B, on the other hand, cannot simply "send" $|i\rangle$ to F. Therefore, B is provided with a unitary operator U_f that implements f:

$$U_f : |i\rangle\,|b\rangle \mapsto |i\rangle\,|b \oplus f(i)\rangle\,. \tag{3.7}$$

Observe that U_f operates on two *registers* i.e., on two collections of qubits, the first encoding the input or query i, the second encoding the XOR of some b and the response $f(x)$.[2]

Let's assume that $n = 32$ is the number of entries in T. If B would want to know whether the entry at index $i = 19$ is a solution, it would first put the first register in the state $|10011\rangle$ and b for example in $|0\rangle$ to then apply U_f and find

$$|10011\rangle\,|0\rangle \xmapsto{U_f} |10011\rangle\,|f(19)\rangle\,. \tag{3.8}$$

So far, it might appear as if there is no real difference between A and B, except that f is implemented in different ways. However, what would happen if B puts the first register in a *superposition* of indices? For example, what happens if B prepares the two registers in the states $|\psi\rangle$ and $|b\rangle$, respectively, and afterwards applies U_f, where

$$|\psi\rangle = \sum_{i=1}^{n} \frac{1}{\sqrt{n}}\,|i\rangle \tag{3.9}$$

[2]In fact, the second register is not really a register, because a single qubit suffices.

is a uniform superposition of all n indices? Since U_f acts linearly on the state that it is applied to, we find that

$$\begin{aligned} U_f \left|\psi\right\rangle \left|b\right\rangle &= U_f \frac{1}{\sqrt{n}} \left(\left|00000\right\rangle + \left|00001\right\rangle + \cdots + \left|11111\right\rangle\right) \left|b\right\rangle \\ &= U_f \frac{1}{\sqrt{n}} \left(\left|00000\right\rangle \left|b\right\rangle + \left|00001\right\rangle \left|b\right\rangle + \cdots + \left|11111\right\rangle \left|b\right\rangle\right) \\ &= \frac{1}{\sqrt{n}} \left(U_f \left|00000\right\rangle \left|b\right\rangle + U_f \left|00001\right\rangle \left|b\right\rangle + \cdots + U_f \left|11111\right\rangle \left|b\right\rangle\right) \\ &= \frac{1}{\sqrt{n}} \left(\left|00000\right\rangle \left|b \oplus f(0)\right\rangle + \left|00001\right\rangle \left|b \oplus f(1)\right\rangle + \cdots + \left|11111\right\rangle \left|b \oplus f(32-1)\right\rangle\right) \end{aligned} \tag{3.10}$$

which is striking, because it shows that it is possible to evaluate f for *all* indices $i \in \mathbb{Z} : 0 \leq i \leq n-1$ at the cost of only a *single* query. Note, however, that if, for example, $b = 0$ and we measure the second register after having queried U_f, the outcome will be $f(i)$ with probability $1/n$. In fact, it is impossible to extract more than one evaluation of f from the state that we obtain after the query.

Does Eq. 3.10 reveal the difference between A and B? On the one hand, yes, contrary to U_f, it is impossible for F to evaluate f more than once in the same amount of time required to evaluate f one single time. On the other hand, no, because we can easily modify F such that it will display the same behaviour as U_f *given that we always measure the second register directly after querying*: we simply let it discard its input, evaluate f for an index i chosen uniformly at random, and return $f(i)$. Hence, in order to take advantage of the quantum parallelism exhibited by U_f, we shall not measure the second register directly after querying, but instead only just before the end of our computation; or rather, not before we know that with high probability, the outcome of the measurement will be 1 and accordingly the first register encodes i' such that $f(i') = 1$.

In a nutshell, that is the crux of Grover's algorithm: exploit quantum parallelism to evaluate concurrently f for different input, do not measure, but instead use quantum interference to amplify the amplitude $\alpha_{i'}$ of the index i' that we are looking for and by that assure that once we *do* measure, we will find i' with high probability. Accordingly, Grover's algorithm is not much more than a repeated application of the *Grover iterate*, which itself consists of the mere application of U_f and $U_{\psi^\perp}$. Indeed, the latter operator is responsible for the amplification of $\alpha_{i'}$. For more information about $U_{\psi^\perp}$, see for example [28, p. 156].

Finally, after how many applications of the Grover iterate should we stop and measure? It can be shown (see for example [28, p. 161]) that the amplitude $\alpha_{i'}$ of $\left|i'\right\rangle$, in terms of the number of applications k of the Grover iterate, equals

$$\alpha_{i'} = \sin\left((2k+1)\,\theta\right) \tag{3.11}$$

where $\sin(\theta) = 1/\sqrt{n}$ and n the number of entries in T. Therefore, $\alpha_{i'} = 1$ if $k = \pi/(4\theta) - 1/2 \approx \pi/4\sqrt{n}$. k needs to be an integer, because we can only apply the Grover iterate an integer number of times, and therefore $k = \lfloor \pi/4\sqrt{n} \rfloor$. It can be shown that after $\lfloor \pi/4\sqrt{n} \rfloor$ applications of the Grover iterate, the probability that measuring will give i' is very close to 1 (for a more precise analysis, the reader is referred once more to [28, p. 161]).

The question of when to stop computing and start measuring will play an important role in Chap. 4, where we will show that a quantum algorithm by Dürr and Høyer [16], based on Grover's, can be used to find with high probability the index i' corresponding to one of the edges e' that maximally increases the robustness of the augmented graph — in less time than required by the (classical) exhaustive search of Sect. 1.2.

Chapter 4

A Quantum Algorithm for Robustness Maximisation

In Sect. 1.2 we explained that solving either problem by searching exhaustively for the best edge to add will take time $\mathcal{O}(N^5)$ in the worst case, where N is the number of vertices in the graph. That being so, both problems can be solved in polynomial time. But we also explained that, in spite of this fact, no clever algorithm that makes use of the problem's hidden structure exists — so far no one has stumbled upon such structure.

For that reason, it seems worthwhile to investigate whether it is possible to simply use Grover's quantum search algorithm to speed up the exhaustive search outlined in Sect. 1.2. After all, if it turns out to be possible, a quadratic speed-up might be obtained, which means a quantum search would not only be faster than a classical search but it would also outperform Kim's bisection algorithm (see Sect. 1.3).

In this chapter, we report on the findings of our investigation: Grover's algorithm cannot be used directly to solve either problem, but a different algorithm based on Grover's by Dürr and Høyer [16] *can* be used to solve both problems *and* outperforms the exhaustive search of Sect. 1.2. The complexity of the algorithm that we will present in Sect. 4.4 is $\mathcal{O}(N^4)$, which is not better than the complexity $\mathcal{O}(N^3)$ of Kim's bisection algorithm, but it does mean that there exists no algorithm (asymptotically) faster than ours that maximises the effective graph resistance of the augmented graph.

4.1 Search Problems

A search algorithm, such as Grover's, can be used to solve a *search problem*, although as one might expect, in order for the search algorithm to be useful it needs to be tailored to the specifics of the search problem first. In particular, the search algorithm needs to be "told" what a solution looks like.

What is a search problem? We adopt the following very general definition of a search problem by Garey and Johnson:

Definition 11 (Search Problem by Garey and Johnson [23, p. 110])**.** A *search problem* Π consists of a set D_Π of finite objects called *instances* and, for each instance $I \in D_\Pi$, a set $S_\Pi[I]$ of finite objects called *solutions* for I. An algorithm is said to *solve* a search problem Π if, given as input any instance $I \in D_\Pi$, it returns the answer "no" whenever $S_\Pi[I]$ is empty and otherwise returns some solution s belonging to $S_\Pi[I]$.

Just as any *decision problem* can be reformulated as a search problem, optimisation problems can be viewed as search problems as well, where the set $S_\Pi[I]$ simply consists of those solutions to the problem that are minimal or maximal in some respect. Both of our problems are optimisation problems and for that reason it might be possible to reformulate them into a search problem that can be solved by a search algorithm (i.e., an algorithm that solves the search problem) such as Grover's.

Let's consider a particular instance $I \in D_\Pi$ of a search problem Π. If x is a possible solution to I (i.e., x is in the *search space* corresponding to I), the algorithm that solves the search problem must be able to tell whether $x \in S_\Pi[I]$ or not. Accordingly, we need to provide the search algorithm with a function f or access to an *oracle* which, given a possible solution x, returns either $f(x) = 1$ if $x \in S_\Pi[I]$ and $f(x) = 0$ otherwise. The search algorithm only determines *how* the search space of I is gone through. That is, what queries will be "send to" the oracle and in which order.

Kaye et al. define the search problem in a less general but more applicable form:

Definition 12 (Search Problem by Kaye et al. [28, p. 153])**.** Given a black box U_f for computing an unknown function $f : \{0,1\}^n \to \{0,1\}$, find an input $x \in \{0,1\}^n$ such that $f(x) = 1$.

It is possible and allowed for f to be known, but usually it is hard to invert. That is, computing $f(x)$ is easy whilst computing $f^{-1}(y)$ is very difficult. If computing $f^{-1}(y)$ were easy, there would be no problem to solve, but if it is not easy and the problem's hidden structure is not known or does not exist, then simply searching for a solution is the best we can do.

4.2 Solution Identification

Grover's algorithm solves the search problem as defined in Definition 12. Hence, if we are able to put together the black box U_f that "identifies" edges that would, if added to the graph, either maximally increase the graph's algebraic connectivity or maximally decrease its effective graph resistance, a speed-up *might* be obtained. Although a quantum search requires fewer

queries than a classical search in the worst case, non-query parts of the algorithm might decelerate as a consequence of replacing the classical search by a quantum search, which is why it does not necessarily result in a speed-up.

Is it possible to distinguish "good" edges from "bad" edges? Considering that a candidate edge e is good only if it is better than or as good as other candidate edges (e.g., in order for e to be a good edge, the algebraic connectivity $a(G+e)$ of the graph $G+e$ must be larger than or equal to the algebraic connectivity of any other graph that can be obtained by adding a single non-existing edge to G), it might not be possible to assess the worth of a single edge without having looked at all other edges first. Consequently, a naive oracle would, if queried whether a particular edge e is good or bad, search exhaustively for all good edges, and report back whether e is among them or not. Needless to say, such an oracle is useless because responding to a single query takes as much time as solving the actual problem of finding good edges. As explained before, Grover's algorithm flourishes if solutions to the problem are easy to identify, but hard to *find*.

To return to the question posed at the beginning of the preceding paragraph, can we tell whether a possible solution $x = e$ of an instance $I = G$ is a solution or not — without the possibility to analyse any other possible solution? If the answer to this question is affirmative, it probably takes either of the following two forms (without loss of generality, we only consider the first instance of our problem):

- Yes, because if the vertices $i, j \in V(G)$ incident with e' share a particular property P, adding e' to G will result in a maximal increase of the graph's algebraic connectivity $a(G)$. It is not known whether such P exists, and it is probably hard to find. Or,
- Yes, because given G, we can compute the maximal possible increase of the graph's algebraic connectivity $a(G + e') - a(G)$ where e' is a solution, and therefore also $a(G + e')$ (because $a(G)$ is known) which can be used to easily identify whether e is a solution or not: we simply need to evaluate whether $a(G + e) \stackrel{?}{=} a(G + e')$.

In the following section, we try to find the maximal possible increase of the graph's algebraic connectivity and the maximal possible decrease of its effective graph resistance, respectively. We show that, for the path graph and the almost-complete graph, *sharp bounds* exist. In general, however, that is for any graph G, no sharp bounds have been found.

4.3 Sharp Bounds

We want to find a *sharp upper bound (supremum)* for the algebraic connectivity of a graph and a *sharp lower bound (infimum)* for its effective graph

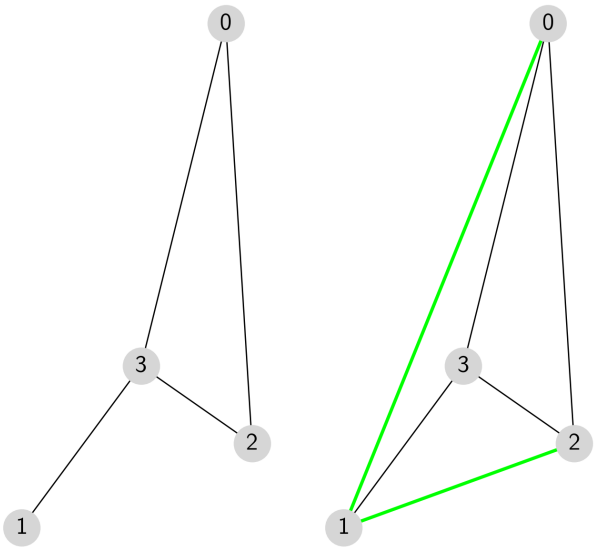


Figure 4.1: Two graphs G (left-hand side) and G' (right-hand side). The green edges $(0, 1)$ and $(1, 2)$ in $E(G')$ maximally increase $a(G)$; the increase is the same for both edges, namely 1. The spectrum of G is $0, 1, 3, 4$ and the spectrum of G extended with either $(0, 1)$ or $(1, 2)$ is $0, 2, 4, 4$. Hence, G is an example of a graph that *cannot* be extended with a non-existing edge e such that $\lambda_2(G + e) = \lambda_3(G)$.

resistance. A *sharp*, *tight*, or *attainable bound* is a bound able to be attained by adding an edge to the graph: for example, if the sharp lower bound for the effective graph resistance of a graph G is x, there exists a candidate edge e' such that $R_{G+e'} = x$ and there does not exist a different candidate whose addition would result in an effective graph resistance *smaller than* x.

The following well-known theorem is relevant to both bounds (see for example [20, p. 134]):

Theorem 1. *Let G be a graph on N vertices. The eigenvalues of $Q(G)$* interlace *those of $Q(G + e)$ where e is an edge or a loop. That is,*

$$\lambda_1(G) \leq \lambda_1(G+e) \leq \lambda_2(G) \leq \lambda_2(G+e) \leq \cdots \leq \lambda_N(G) \leq \lambda_N(G+e) \quad (4.1)$$

Unfortunately, the bounds of Eq. 4.1 are not sharp: It is not always possible to find an edge e such that $a(G + e) = \lambda_3(G)$ (see for example Fig. 4.1). In addition, Eq. 4.1 cannot be used to formulate an infimum for the effective graph resistance (based on Eq. 2.11) because not only does Eq. 4.1 not contain an upper bound for $\lambda_N(G + e)$, at least one of the inequalities must be strict, as follows from Eq. 4.5 (see below). In the remainder of this section, we will treat the supremum for a graph's algebraic connectivity and the infimum for its effective graph resistance separately from each other, starting with the former.

4.3.1 A Supremum for the Algebraic Connectivity

What is the largest possible increase of a graph's algebraic connectivity? That is, for a given graph G, what is the sharp upper bound of the set

$$\mathcal{A}_G = \left\{ a(G+e) : e \in \bar{E}(G) \right\} \tag{4.2}$$

where

$$\bar{E}(G) = \{(i,j) : i < j \text{ and } i,j \in V(G) \text{ and } (i,j) \notin E(G)\} \tag{4.3}$$

is the set of candidate edges. Except for some highly particular graphs G, a sharp upper bound of $\mathcal{A}_G$ has not been found in literature.

We do know that for any square n by n matrix A, the sum of its n diagonal entries (i.e., the *trace* of A) equals the sum of its n eigenvalues. That is,

$$a_{11} + a_{22} + ... + a_{nn} = \lambda_1 + \lambda_2 + ... + \lambda_n \tag{4.4}$$

and therefore

$$\sum_{i=1}^{n} (\lambda_i(G+e) - \lambda_i(G)) = 2. \tag{4.5}$$

After all, the diagonal of the Laplacian consists of the degrees of the vertices in the graph, in some particular order, and therefore, if we add an edge to the graph, two of them increase by 1, and accordingly their sum increases by 2.

Let $N(i)$ be the set of vertices adjacent to vertex i. The following theorem by So [44, p. 197] reveals when, given a graph G that we expand by adding an edge e to it, only a single eigenvalue of $Q(G)$ increases:

Theorem 2. *Suppose i and j are fixed but arbitrary nonadjacent vertices in G. Let $e = (i,j)$. Then $N(i) = N(j)$ if and only if the spectrum of $Q(G+e)$ overlaps the spectrum of $Q(G)$ in $n-1$ places.*

As a consequence, if $N(i) = N(j)$, then due to Eq. 4.5 the eigenvalue $\lambda_i(G+e)$ that does not equal $\lambda_i(G)$ must have increased by 2.

The question then arises, when does the algebraic connectivity $\lambda_2(G)$ increase by 2? Part of the answer is given by Barik and Pati [5, p. 219]:

Theorem 3. *Let G be a connected graph on N vertices and let $e \in \bar{E}(G)$. Let $a(G)$ have multiplicity one. Then the spectral integral variation occurs in one place by adding the edge e where the changed eigenvalue is $a(G)$ if and only if $G = K_N - e$.*

A spectral integral variation means the spectrum of the Laplacian changes such that one or more eigenvalues change by some integer. In fact, due to Eq. 4.5 there are only two possibilities: after edge addition, a single eigenvalue λ_i has increased by 2 or two eigenvalues λ_i and λ_j with $i \neq j$ have

both increased by 1. Characterising those graphs for which the latter holds is a question posed by So [44, p. 197] as well, to which the answer was found by Kirkland [32, p. 80] five years later.

Theorem 3 gives a sharp bound of $\mathcal{A}_G$ where $G = K_N - e$, namely $a(G) + 2$. However, if $G = K_N - e$ the problem of finding the best edge to add is trivial anyway as there is only a single possibility. Below, in the next subsection, we give one additional sharp bound of $\mathcal{A}_G$ if G is the path graph P_N on N vertices, but also in this case, it is trivial to find the edge that maximally increases $a(G)$. The problem of finding a sharp bound of $\mathcal{A}_G$ for *any* graph G, without making an exhaustive search, remains open.

A closely related problem that has been solved is, given M, for all incomplete graphs G where $|E(G)| = M$, what is the largest possible value of $a(G)$? That is, for some natural number M, what is the sharp upper bound of the set

$$\mathcal{B}_M = \{a(G) : |E(G)| = M\}. \tag{4.6}$$

Belhaiza et al. [6, p. 13] show that for all $G \neq K_N$ and $M \geq 2$, the following upper bound of $\mathcal{B}_M$ is sharp:

$$a(G) \leq \left\lfloor -1 + \sqrt{1 + 2M} \right\rfloor. \tag{4.7}$$

At first glance, it might seem as if we can translate the bound of Eq. 4.7 to a sharp bound of $\mathcal{A}_G$. That is, perhaps the following bound is sharp as well:

$$a(G + e) \leq \left\lfloor -1 + \sqrt{1 + 2(M + 1)} \right\rfloor \tag{4.8}$$

where M is still the number of edges of G. However, M does not fully determine the graph: Equation 4.8 holds for *all* graphs G' such that $|E(G')| = M + 1$. Consequently, it is possible that we cannot "obtain" a graph $G + e$ such that $a(G+e) = \left\lfloor -1 + \sqrt{1 + 2(M + 1)} \right\rfloor$ by adding a single edge to the graph G that we would like to expand.

More bounds

Fiedler, the inventor of algebraic connectivity, gives an explicit formula for the algebraic connectivity of a number of standard graphs in the same paper in which he defines the algebraic connectivity of a graph. Fiedler's formulas are given in Table 4.1.

It can be shown that the following formula describes the spectrum of the Laplacian $Q(P_N)$ of the path graph P_N on N vertices (for proofs of Eqs. 4.9 and 4.10, see for example [9, p. 8]):

$$\lambda_k(P_N) = 2\left(1 - \cos\left(\frac{\pi(k-1)}{N}\right)\right) \tag{4.9}$$

Table 4.1: Explicit formulas for the algebraic connectivity $a(G)$, edge connectivity $e(G)$, and vertex connectivity $v(G)$ of a graph G on N vertices [21, p. 304].

G	$a(G)$	$e(G)$	$v(G)$
path	$2\left(1-\cos\left(\pi/N\right)\right)$	1	1
cycle	$2\left(1-\cos\left(2\pi/N\right)\right)$	2	2
star	1	1	1
complete	N	$N-1$	$N-1$
k-cube	2	k	k

where $k = 1, 2, \ldots, N$. Similarly, the spectrum of $Q(C_N)$ where C_N is the cycle graph on N vertices is accurately described by

$$\lambda_k(C_N) = 2\left(1-\cos\left(\frac{2\pi(k-1)}{N}\right)\right). \tag{4.10}$$

Observe that Eqs. 4.9 and 4.10 are consistent with Table 4.1: if $k = 2$, we find that $\lambda_2 = a(G)$ as specified in Table 4.1. Also observe that

$$\lambda_2(C_N) = \lambda_3(P_N). \tag{4.11}$$

Considering that it is always possible to obtain C_N by adding the edge e' to P_N that connects the two endpoints of the path graph (i.e., the only two vertices in $V(P_N)$ that have valency 1), that is, $P_N + e' = C_N$, we have

$$\lambda_2(C_N) = \lambda_2(P_N + e') = \lambda_3(P_N). \tag{4.12}$$

In addition, a corollary of Theorem 1 is that if

$$\lambda_2(G + e') = \lambda_3(G), \tag{4.13}$$

e' is among the best edges to add with respect to the graph's algebraic connectivity. Therefore, the supremum of $\mathcal{A}_{P_N}$ is $\lambda_3(P_N)$. Of course, it would be silly to formulate an oracle U_f based on this bound, and use Grover's algorithm to search for an edge $e \in \bar{E}_{P_N}$ that maximally increases the algebraic connectivity of P_N.

Finally, it is worthwhile to mention the works of Kirkland [31, 33], who found a tight bound for the algebraic connectivity of a graph, in terms of the number of vertices N and cutpoints k. A *cutpoint* of a connected graph is a vertex that, if removed, turns the graph into a disconnected graph. He proves that, the algebraic connectivity $a(G)$ of a graph G on N vertices with k cutpoints, where $2 \leq k \leq N/2$, is maximal if and only if

- each cutpoint is adjacent to every non-pendant[1] vertex of G; and

[1] A *pendant* vertex is a vertex that has valency 1 (and a non-pendant vertex has a valency larger than 1).

- there are exactly two components at each cutpoint of G, one component of which is a pendant vertex.

The proof can be found in [31, p. 102]. In [33] he presents a similar theorem, which deals with the case where $N/2 < k$. Neither theorem can be used directly to base the supremum of $\mathcal{A}_G$ on because, as more or less also applies to the bound by Belhaiza et al.: not all graphs on N vertices with k cutpoints can be transformed into a graph that meets both of the requirements above, just by adding a single edge. For example, if a graph contains no pendant vertices, it becomes impossible to meet the second requirement if all we are allowed to do is add one edge.

To conclude this section, there exist many bounds for the algebraic connectivity of a graph (for a survey, see [12]), but almost all of them are not sharp. The only two that are tight, namely the bounds by Belhaiza et al. and Kirkland, are expressed in terms of graph invariants that do not fully determine the graph, and as a consequence are not readily applicable to the question that we asked ourselves in this section, namely what is the supremum of $\mathcal{A}_G$? If G is $K_N - e$ or P_N, the answer is $a(K_N - e) + 2$ or $\lambda_3(P_N)$, respectively, but otherwise the answer is not known.

4.3.2 An Infimum for the Effective Graph Resistance

We want to find the sharp lower bound of the set

$$\mathcal{R}_G = \left\{ R_{G+e} : e \in \bar{E}(G) \right\} \tag{4.14}$$

where G is given. As before, there is an abundance of literature on bounds for the effective graph resistance, but all of these bounds are in terms of graph invariants such as the number of vertices N, the average degree $\bar{d}$, or even the graph's algebraic connectivity $a(G)$ (for the last mentioned, see [19, p. 2494]).

For example, the following theorem by Deng [13, p. 177] gives the minimum effective graph resistance among all graphs G on N vertices that have k bridges. A *bridge* $e \in E(G)$ of a not-necessarily-connected graph G is an edge whose removal increases the number of components of G (i.e., the edge-equivalent of Kirkland's cutpoints) [27, p. 26].

Theorem 4. *If G is a connected graph on N vertices with k bridges, then*

$$N(k+1) + 1 - \frac{2N}{N-k} \leq R_G. \tag{4.15}$$

Deng also identifies the graphs G for which $R_G = N(k+1) + 1 - 2N/(N-k)$: the complete graphs K_{N-k} in which each vertex $i \in K_{N-k}$, in addition to being adjacent to all other vertices in $V(K_{N-k})$, is also connected to a pendant vertex $j \notin V(K_{N-k})$. The resulting graphs contain k

Table 4.2: Explicit formulas for the effective graph resistance R_G, algebraic connectivity $a(G)$, edge connectivity $e(G)$, and vertex connectivity $v(G)$ of a graph G on N vertices.

G	R_G	$a(G)$ [21, p. 304]	$e(G)$ [21, p. 304]	$v(G)$ [21, p. 304]
path	$(N^3 - N)/6$ [19, p. 2498]	$2\,(1 - \cos(\pi/N))$	1	1
cycle	$(N^3 - N)/12$ [37, p. 218]	$2\,(1 - \cos(2\pi/N))$	2	2
star	$(N-1)^2$ [19, p. 2498]	1	1	1
complete	$N - 1$ [19, p. 2498]	N	$N - 1$	$N - 1$
k-cube	See Theorem 9 in [36, p. 3].	2	k	k

of such pendant vertices (and accordingly k bridges) and therefore $N = 2k$ vertices in total.

A more general bound for the effective graph resistance is given by Van Mieghem [47, p. 207]:

Theorem 5. *If G is a connected graph on N vertices, then*

$$\frac{(N-1)^2}{\bar{d}} \leq R_G \tag{4.16}$$

where $\bar{d}$ is the average of the degrees $d_0, d_1, \ldots, d_{N-1}$ of the vertices $0, 1, \ldots, N-1 \in V(G)$, respectively.

The lower bound of Eq. 4.16 is tight, because if $G = K_N$, $\bar{d} = N - 1$ and therefore $(N-1) \leq R_{K_N}$. It is not hard to show that $R_{K_N} = N - 1$ (see for example [19, p. 2498] or our own proof in Sect. 5.1) and as a result, the bound of Eq. 4.16 is sharp.

Unfortunately, however, neither the bound of Eq. 4.15 nor the bound of Eq. 4.16 says something about the maximum possible decrease of the graph's effective graph resistance, resulting from the addition of a single edge. A theorem similar to Theorem 3, for example, but which applies to the effective graph resistance, does not seem to exist. What does exist, on the other hand, are explicit formulas for the effective graph resistance of certain standard graphs. The formulas can be found in Table 4.2, which builds on Table 4.1.

4.4 Dürr and Høyer's Algorithm Applied

Shortly after Grover published his innovative quantum search algorithm in 1996, Dürr and Høyer [16] published a quantum algorithm for finding the minimum (or maximum) in an unsorted database, which uses Grover's algorithm[2] as a subroutine. Their algorithm does not require the minimum to

[2]Not exactly Grover's algorithm, rather a generalisation of it by Boyer et al. [8, p. 498] which does not require the number of solutions to be known.

be known, which takes away the need for the supremum and infimum sought for in the two preceding sections. The algorithm looks as follows.

Dürr and Høyer's Algorithm [16]

Parameters. Let T denote the unsorted database that we are searching through, and $T[i]$ the entry at index i. Moreover, the number of entries in T equals n, which has order of magnitude 1 or larger.

Output. With probability $p > 1/2$, index i_m such that $T[i_m]$ is the smallest entry in the database.

1. Define λ to be $8/7$ (see [8, p. 498]).
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 iterations exceeds $\lfloor 8\pi\sqrt{n} \rfloor$.[3] Afterwards, go to (4.).
   a. Use the generalisation of Grover's algorithm by Boyer et al. [8, p. 498] 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 $\sum_{j=0}^{n-1} \frac{1}{\sqrt{n}} |i\rangle |i_t\rangle$.
      iii. Choose $l \in \mathbb{Z} : 0 \leq l < s$ uniformly at random.
      iv. Apply the Grover iterate (see Sect. 3.2) l times. The oracle f is implemented by U_f as follows:

$$U_f : |i\rangle |i_t\rangle \mapsto \begin{cases} -|i\rangle |i_t\rangle & \text{if} \quad T[i] < T[i_t] \\ |i\rangle |i_t\rangle & \text{otherwise} \end{cases} . \quad (4.17)$$

      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;
      - otherwise let $s := \min(\lambda s, \sqrt{n})$ and go to (ii.).
4. Return i_t.

[3] Actually, in [16, p. 1] a slightly smaller bound for the cumulative sum of Grover iterations is given, namely $22.5\sqrt{n} < 25\sqrt{n} = \lfloor 8\pi\sqrt{n} \rfloor$, which is probably a little mistake by the authors and should have been $20\sqrt{n}$. Why is our bound $25\sqrt{n}$ instead of $20\sqrt{n}$? As will be discussed below in more detail, the latter bound is only valid if during step (2.) not just one but instead a handful of indices is chosen uniformly at random and of which the best becomes the initial threshold index i_{t_0}.

Dürr and Høyer's 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 grows rapidly as c increases. For example, after running the algorithm $c = 8$ times, the minimum is part of the outcomes with a probability that is certainly larger than 0.99.

Does this mean that, if we would tailor the algorithm by Dürr and Høyer to the specifics of either instance, we would arrive at two algorithms, each of which solves a different instance, but both of which have a complexity of $\mathcal{O}(N)$? (Recall from Sect. 1.2 that the size of our search space n is strictly smaller than N^2.) The answer, unfortunately, is in the negative: the algorithm's running time is composed of more than just its *query complexity* of $\left\lceil 8\pi\sqrt{n} \right\rceil$. In general, the complexity of a black-box algorithm, U_f being our black box, equals the query complexity T times the complexity B of a single query, plus the total complexity A of all non-query operations.

In what follows, we will elaborate on where the requisite number of $\left\lceil 8\pi\sqrt{n} \right\rceil$ Grover iterations comes from. We will show why $\left\lceil 8\pi\sqrt{n} \right\rceil$ is the query complexity T of the algorithm, $1.39 \log_2^2 n$ the complexity A of all non-query operations, and that the complexity B of a single query — irrespective of the instance being solved — is $\mathcal{O}(N^3)$. As a result, the time complexity of the algorithm, after the details have been put in, is $\mathcal{O}(N^4)$.

4.4.1 Running Time in the Black-Box Model

Clearly, the loop that step (3.) is 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.

The following result is well-known (for a proof, see for example [43, p. 71]):

Theorem 6 (Markov's Inequality [43, p. 71])**.** *If X is a random variable that takes only non-negative values, then for any value $a > 0$*

$$P\{X \geq a\} \leq \frac{\mathbb{E}[X]}{a} \tag{4.18}$$

where $P\{X \geq a\}$ denotes the probability that X is larger than or equal to a.

We define X to be the time it takes until $i_t := i_m$. Theorem 6 shows that

$$P\{X \geq 2\mathbb{E}[X]\} \leq \frac{\mathbb{E}[X]}{2\mathbb{E}[X]} = \frac{1}{2} \tag{4.19}$$

and

$$1 - P\{X \geq 2\mathbb{E}[X]\} > 1 - \frac{1}{2} = \frac{1}{2} \tag{4.20}$$

$$P\{X < 2\mathbb{E}[X]\} > \frac{1}{2}. \tag{4.21}$$

Therefore, if we interrupt the loop after it has run for time $2\mathbb{E}[X]$, we know that the probability that $i_t = i_m$ is at least $1/2$. The remainder of this section will be about computing the expectation of X.

The point of subroutine (3a.) is to find an index i'_t such that $T[i'_t] < T[i_t]$. The larger the difference between $T[i_t]$ and $T[i'_t]$, the sooner we are done. Or, the smaller the number of entries smaller than $T[i'_t]$, the sooner we are done. If there is as little as one entry smaller than $T[i'_t]$, we know that with high probability only one more execution of Boyer et al.'s algorithm will be necessary before we can move on to step (4.). The question arises, what is — at most — the expected value of the time it takes until there are no entries smaller than $T[i'_t]$?

In the worst case, no two entries in T have the same value, our random initial threshold index i_{t_0} is the index of the largest entry in the database, and subroutine (3a.) always results in a new threshold index i'_t such that $T[i'_t]$ is the largest entry smaller than $T[i_t]$. As a consequence, only after $n-1$ executions of Boyer et al.'s algorithm i_t will be i_m.

Fortunately for us, it is unlikely that all $n-1$ possible executions are required. Most will be skipped, automatically, simply because the result of (3a.) 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. What's more, it is impossible that, in the future, they are chosen.

One way to compute $\mathbb{E}[X]$ is by first observing that

$$X = Y_1 + Y_2 + \cdots + Y_{n-1} \tag{4.22}$$

where Y_k is a random variable defined as the length of the kth possible execution of Boyer et al.'s algorithm. The value of $T[i_t]$ at the start of the kth possible execution is the kth largest of all entries in T. For example, the value of $T[i_t]$ at the start of the first execution (i.e., $k = 1$) of the $n-1$ possible executions is the value of the largest entry in T.

Since, in general

$$\mathbb{E}[Z_1 + Z_2 + \cdots + Z_n] = \mathbb{E}[Z_1] + \mathbb{E}[Z_2] + \cdots + \mathbb{E}[Z_n] \tag{4.23}$$

where $Z_1, Z_2, \ldots, Z_n$ are arbitrary random variables, we can evaluate $\mathbb{E}[X]$ if we can work out $\mathbb{E}[Y_k]$. What is $\mathbb{E}[Y_k]$? Evidently, the kth execution will either take place and have length $L_k > 0$ or it will not take place and have zero length. That is,

$$\mathbb{E}[Y_k] = pL_k \tag{4.24}$$

where p denotes the probability that the kth execution takes place and L_k the length of the execution. Observe that the kth execution occurs directly after, and at no other point in time, threshold index i_t becomes x such that $T[x]$ is the kth largest entry in T. As a result, p 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 $p = 1/r$ 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 [16, p. 1]). As a result, for example,

$$\mathbb{E}[Y_{n-1}] = \frac{1}{2}L_{n-1} \tag{4.25}$$

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,

$$\mathbb{E}[X] = \mathbb{E}[Y_1 + Y_2 + \cdots + Y_{n-1}] = \frac{1}{n}L_1 + \frac{1}{n-1}L_2 + \cdots + \frac{1}{2}L_{n-1} \tag{4.26}$$

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 same for all iterations: by convention, it is $\log_2 n$ [16, p. 1]. More difficult is the determination of the duration of (iv.), because it depends on k.

Boyer et al. show that the running time of their algorithm i.e., of subroutine (3a.), is $\mathcal{O}(\sqrt{n/t})$ where n is again the size of T and t the number of solutions [8, p. 499]. Recall that an index i is a solution if $T[i] < T[i_t]$. Like Dürr and Høyer, they also assume *unit time* oracle queries.

Unfortunately for us, in order to arrive at their result, Boyer et al. make an assumption that cannot be made if the algorithm is used as in, for example, Dürr and Høyer's algorithm. They assume that $t < 3n/4$. For all iterations $k \geq n - \lceil 3n/4 \rceil$ the assumption is correct, but otherwise if $k < n - \lceil 3n/4 \rceil$ it definitely cannot be made. The threshold $T[i_t]$ of each of these iterations is so large that the number of solutions t is guaranteed to be at least $3n/4$.[4]

[4]In their complexity analysis, Boyer et al. explain that "The case $t > 3n/4$ can be disposed of in constant expected time by classical sampling." [8, p. 499]. However, how does one know whether $t > 3n/4$ or not? After all, the algorithm is an algorithm "for finding a solution when the number t of solutions is unknown." [8, p. 498]. The way out of this difficulty is to, before going to step (3.), pick uniformly at random $d > 1$ indices and let the initial threshold index i_{t_0} be the index i among the d random indices such that $T[i]$ is minimal. If, for example, $d = 4$, the probability that $T[i_{t_0}]$ belongs to the smallest 75% of T equals $1 - (1/4)^4$ which is larger than 0.99. Since picking four random indices and comparing the corresponding entries takes constant time, Boyer et al.'s assumption that $t < 3n/4$ is in fact justified.

The assumption is necessary to show that

$$m_0 = \frac{n}{2\sqrt{(n-t)t}} \leq \sqrt{\frac{n}{t}} \tag{4.27}$$

but not needed to show that the expected number of applications of the Grover iterate is at most $8m_0$ [8, p. 499]. For each of all $n-1$ possible executions, the number of solutions t is known: There are $n-1$ entries $T[i]$ smaller than the threshold $T[i_t]$ of the first execution, $n-2$ entries $T[i]$ smaller than threshold $T[i_t]$ of the second execution, etc. until we reach execution $n-1$ and there is only one entry smaller than the threshold $T[i_t]$, which is $T[i_m]$. In other words, $t = n-k$ and we have found L_k:

$$\begin{aligned} L_k &= 8m_0 B + \log_2 n = \frac{4Bn}{\sqrt{(n-t)t}} + \log_2 n \\ &= \frac{4Bn}{\sqrt{(n-k)k}} + \log_2 n \end{aligned} \tag{4.28}$$

where B is the complexity of a single query. As a result, the expected value $\mathbb{E}[X]$ of X is

$$\mathbb{E}[X] = \sum_{k=1}^{n-1} \frac{1}{n-(k-1)} \left(\frac{4Bn}{\sqrt{(n-k)k}} + \log_2 n \right) \tag{4.29}$$

$$= B \sum_{k=1}^{n-1} \frac{4n}{(n-k+1)\sqrt{(n-k)k}} + \log_2 n \sum_{k=1}^{n-1} \frac{1}{n-k+1} \tag{4.30}$$

$$= B \sum_{k=1}^{n-1} \frac{4n}{(n-k+1)\sqrt{(n-k)k}} + (H_n - 1)\log_2 n \tag{4.31}$$

where H_n denotes the nth *harmonic number*. The remaining summation can be *overestimated* by a definite integral on the interval $[1, n]$. We need to increase the upper limit by 1 because as illustrated in Figs. 4.2–4.4, the terms of the summation, as a function of k and for some constant n, are virtually nondecreasing on the interval $1 \leq k \leq n-1$. Accordingly, in order for the integral to actually overestimate the summation, the latter should be thought of as a *right* Riemann sum. That is,

$$\sum_{k=1}^{n-1} \frac{4n}{(n-k+1)\sqrt{(n-k)k}} \leq \int_{k=1}^{n} \frac{4n}{(n-k+1)\sqrt{(n-k)k}} \tag{4.32}$$

and

$$\int_{k=1}^{n} \frac{4n}{(n-k+1)\sqrt{(n-k)k}} = \frac{2n\left(2\arcsin\left(\frac{n^2-2}{n^2}\right)+\pi\right)}{\sqrt{n+1}} \leq \frac{4\pi n}{\sqrt{n+1}}. \tag{4.33}$$

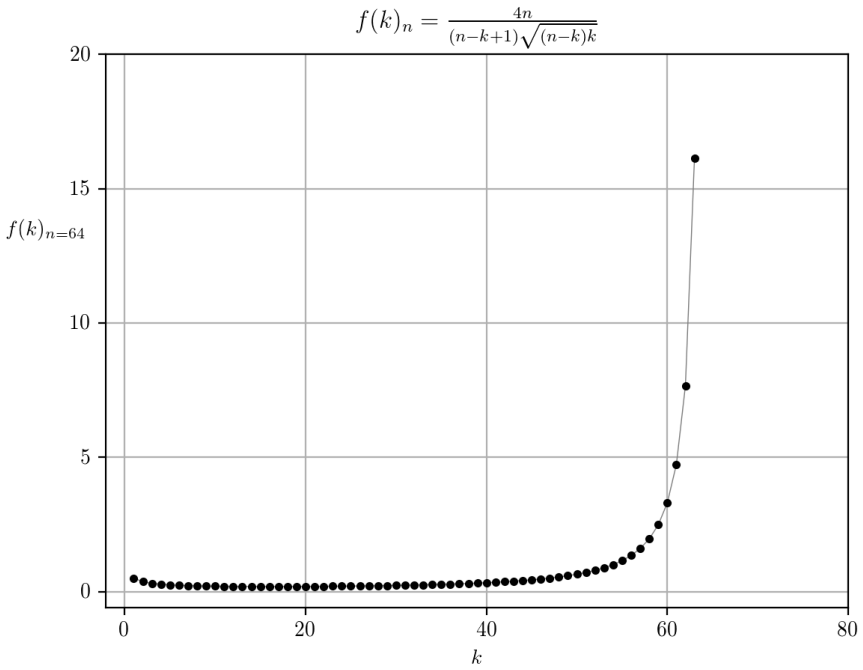


Figure 4.2: The graph of $f(k)_{n=64}$.

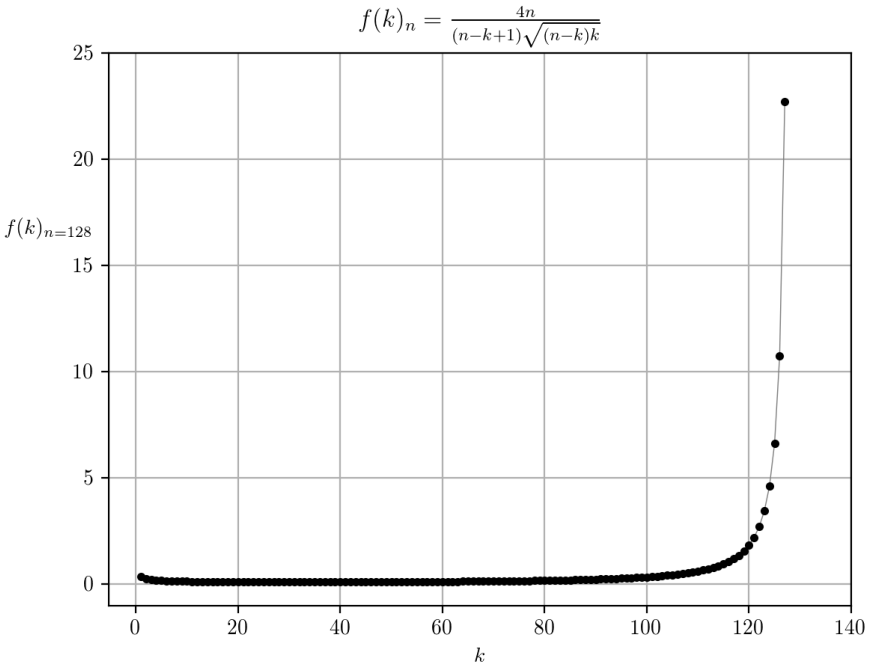


Figure 4.3: The graph of $f(k)_{n=128}$.

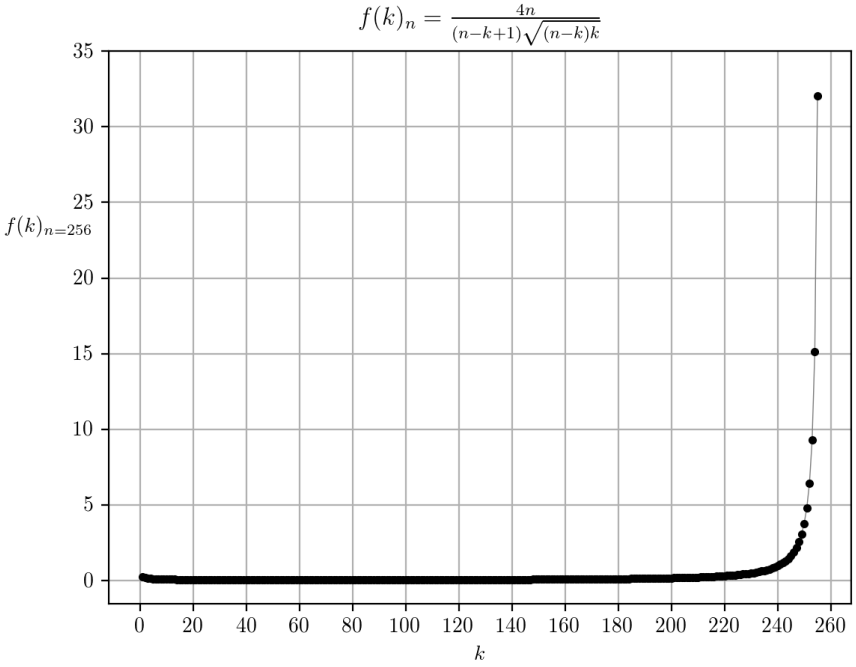


Figure 4.4: The graph of $f(k)_{n=256}$.

Combining Eq. 4.33 with the fact that $H_n - 1 < \ln n$ (see for example [41, p. 2]), we find that, if n is sufficiently large and $\sqrt{n+1} \approx \sqrt{n}$,

$$\mathbb{E}[X] \leq 4\pi B\sqrt{n} + \ln n \log_2 n \tag{4.34}$$

$$\leq 4\pi B\sqrt{n} + \frac{1}{\log_2 e} \log_2^2 n. \tag{4.35}$$

As a result, after time twice the upper bound of Eq. 4.35, 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 $1/2$. Consequently, Dürr and Høyer's algorithm does indeed run 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 $|\bar{E}(G)|$ and therefore always smaller than N^2 (see Sect. 1.2).

4.4.2 Complexity of the Oracle

How long does it take for the oracle U_f to respond to a single query? Let's consider each problem instance separately. First, the problem of maximising a graph's algebraic connectivity $a(G)$, afterwards the problem of minimising its effective graph resistance R_G.

Computing the Algebraic Connectivity

Let $a(G+e_i)$ be the algebraic connectivity of $G+e_i$, where e_i is the ith edge of a totally ordered $\bar{E}(G)$, ordered according to some arbitrary total order. The oracle is given $a(G+e_{i_t})$ for some index i_t and asked to compare it to $a(G+e_i)$ for any i and change the phase of $|i\rangle$ accordingly. It seems that, in order to do so, it cannot but compute $a(G+e_i) = \lambda_2(G+e_i)$ each time we ask for a comparison.

The most efficient way to compute λ_2 of a given Laplacian Q is by means of the symmetric QR algorithm mentioned before (see Sect. 1.2). This classical algorithm, like any other classical algorithm, can be implemented efficiently on a quantum computer using only *Toffoli gates* [39, p. 29]. The time complexity B of the oracle is equal to the complexity of the symmetric QR algorithm, which is $\mathcal{O}(N^3)$.

Computing the Effective Graph Resistance

This time, the oracle needs to compute R_{G+e_i} before it can respond to our query. We have seen in Sect. 2.3 that there are no less than four different ways by which the effective graph resistance R_G of a graph G can be computed. Or rather, by which the effective resistance R_{ij} between two vertices i and j can be computed. The most efficient way, however, is to either compute the Moore-Penrose matrix inverse of the Laplacian to obtain Q^+ and use Eq. 2.10 to compute R_G *or* to compute the spectrum of Q and then Eq. 2.11 to find R_G. Again, using the symmetric QR algorithm to compute the eigenvalues of Q, the complexity of the oracle is $\mathcal{O}(N^3)$.

4.4.3 Integrating Heuristics

Consider, once more, Dürr and Høyer's algorithm. In Sect. 4.4.1 we explained that the running time of the algorithm is heavily dependent on the ranking r of the threshold $T[i_t]$. The threshold of execution $k = 1$ is chosen uniformly at random, and from then on determined by the outcome of (v.). What if we do not choose the first threshold randomly, but instead make an educated guess at the index i of a very small, if not minimal, entry $T[i]$?

Before we discuss how such an educated guess can be made, i.e. how relatively good edges can be identified, we will determine the potential gain of time. To that end, we will quantify the performance P of a heuristic $h : G \mapsto h(G)$ by the maximum number of entries in T smaller than or equal to $T[H]$ where H is a random variable defined as the index that the heuristic returns after we feed it G. The index H is one of P indices and we will assume that it has a uniform distribution. The smaller P, the better h. For example, if $P = n$ for some heuristic h, then h performs as good as a heuristic that just returns an index chosen uniformly at random. And if $P = 1$, the heuristic always returns i_m.

Equation 4.26 shows that there is a very simple relationship between the performance P of a heuristic h and the minimum speed-up that using h will result in. If, for example, $P = n/2$ for some heuristic h, we are effectively solving the same problem but without using h, except that our database T no longer contains n entries, but $P = n/2$ instead. As a result, the complexity of Dürr and Høyer's algorithm, in terms of the performance P of the heuristic h that is used to choose the initial threshold index i_{t_0} and thereby replaces step (2.), becomes $\mathcal{O}(B\sqrt{P})$.

An Example

In [49], Wang and Van Mieghem evaluate two different heuristics for finding an edge that maximally increases the algebraic connectivity of the augmented graph. The first heuristic h_1 is very simple: pick a random vertex i from the set of vertices that have minimal degree, and connect it to a random vertex j that is not in i's neighbourhood already. In a different work by Wang et al. [50], of which Van Mieghem is indeed co-author, the same heuristic is analysed but this time to solve the second instance of our problem, where robustness is defined as minus the graph's effective graph resistance.

In both works, the performance of h_1 is measured by the (relative) increase of algebraic connectivity and the effective graph resistance, respectively. We, on the other hand, are interested in P: the maximum number of entries in T smaller than or equal to $T[H]$. To that end, Fig. 4.5 shows the rate of occurrence of each possible ranking r of the index $H = i_{h_1}$ "recommended" by h_1. The frequencies in Fig. 4.5 have been approximated as follows: for each of 1003 random, connected, non-isomorphic graphs of 16 vertices and 42 edges, a sorted table T_s of size $|\bar{E}(G)| = 78$ consisting of the algebraic connectivities in $\mathcal{A}_G$, indexed by the edges in $\bar{E}(G)$, has been constructed. The ranking r of i_{h_1} is simply the ranking of the edge e_{h_1}, the edge corresponding to i_{h_1}, in T_s. If $r = 1$, e_{h_1} is the best edge to add and accordingly $i_{h_1} = i_m$ is the index of the *largest* entry in T_s. Note that, although the algebraic connectivities of two different entries in T_s may be equal, their rankings will be different, which introduces a small error in the histogram in Fig. 4.5. For example, it is possible that the algebraic connectivities in T_s are *all* equal, but because of the arbitrariness intrinsic to the labelling of vertices, e_{h_1} will probably not have ranking 1 — while it actually has. As a consequence, the histogram in Fig. 4.5 paints a slightly *worse* — not better — picture of the distribution of the ranking of H.

Fig. 4.6 shows the same kind of histogram as in Fig. 4.5, but this time robustness is defined as minus the graph's effective graph resistance. Although a non-trivial upper bound for the performance P of h_1 is probably hard to find, Figs. 4.5 and 4.6 suggest that a speed-up can be obtained by including h_1 in Dürr and Høyer's algorithm (if it is used to solve our problem)

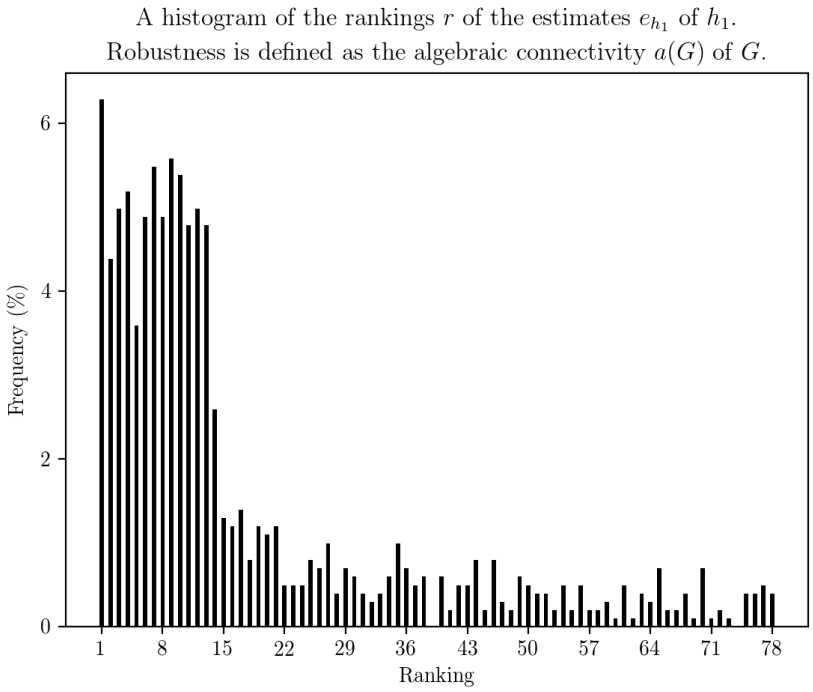


Figure 4.5: A histogram of the rankings r of the estimates e_{h_1} of h_1. Robustness is defined as the algebraic connectivity $a(G)$ of G. The number of random graphs in the experiment was 1003, the number of edges in each of these graphs 42 and accordingly the number of candidate edges 78, which explains why the rankings range from 1 to 78. Observe that h_1 guesses i_m correctly $\sim$6% of the time and that $T[i_{h_1}]$ usually ($\sim$70% of the time) belongs to the largest 25% of T.

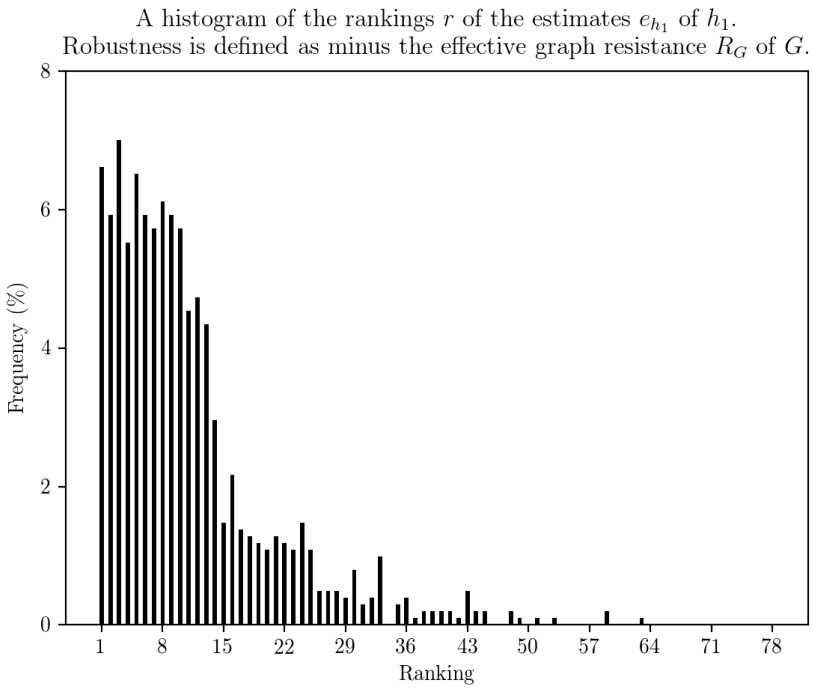


Figure 4.6: A histogram of the rankings r of the estimates e_{h_1} of h_1. Robustness is defined as the minus the effective graph resistance R_G of G. The number of random graphs in the experiment was 1013, the number of edges in each of these graphs 42 and accordingly the number of candidate edges 78, which explains why the rankings range from 1 to 78. Observe that h_1 guesses i_m correctly $\sim$6% of the time and that $T[i_{h_1}]$ usually ($\sim$80% of the time) belongs to the smallest 25% of T.

compared to *vanilla* Dürr and Høyer.

We should not forget that running the algorithm H_1 that implements h_1 will take time as well. Worse still, the time complexity of H_1 is dependent on the number of vertices N of the graph. The most expensive part of H_1, in terms of time, will be finding a random vertex that has minimal degree, because in the worst case, we will have to iterate over all N rows in the adjacency matrix A (the input to our algorithm), each time computing their N-term sum to find the degree of the vertex, whilst keeping track of the smallest degree encountered so far. If, however, we come across a vertex that has degree 1, we can stop searching immediately.

In any case, H_1 will take time $\mathcal{O}(N^2)$ (for a bit more detail, see [50, p. 6]). Given that N is reasonably large, as long as the time complexity of the algorithm for computing the heuristic is better than the complexity $\mathcal{O}(N^4)$ of Dürr and Høyer's algorithm applied to our problem, using the heuristic will be beneficial. For example, if $P \approx n/4$ — a reasonable estimate of h_1's performance if robustness is defined as minus the graph's effective graph resistance (see Fig. 4.6) — the running time of Dürr and Høyer's becomes

$$8\pi N^3\sqrt{\frac{N^2}{4}} + \mathcal{O}(N^2) + 1.39\log_2^2 N^2 = 4\pi N^4 + \mathcal{O}(N^2) + 1.39\log_2^2 N^2 \quad (4.36)$$

and the halving of the constant in front of the fourth-order term outweighs the additional polynomial of degree 2.

In the works by Wang and Van Mieghem [49] and Wang et al. [50], more involved and computationally expensive heuristics are proposed for both problems, respectively. The same kind of histograms as in Figs. 4.5 and 4.6, but illustrating the performance of those heuristics, can be made without difficulty. In general, one should expect that, the more involved the heuristic, the worse its time complexity but also the better its performance.

Chapter 5

Conclusion

In this work, we asked ourselves the question, given a network, the addition of which single edge will maximally increase the network's robustness? We defined robustness as the algebraic connectivity of the graph and minus its effective graph resistance, respectively. In Chap. 1, we explained that searching exhaustively for the best edge to add takes time $\mathcal{O}(N^5)$; that Kim's bisection algorithm solves the first instance in time $\mathcal{O}(N^3)$, but that no algorithm exists that makes use of the hidden structure that either problem may follow. Perhaps no such structure exists, and the best we can do is to "simply" search for the edge that maximally increases the robustness of the graph.

Although using search, as a means to compute something, is simple compared to the difficulty of revealing a problem's hidden structure, there are all kinds of different ways to search: clever and naive searches, efficient and inefficient ones. In particular, Grover's algorithm *defined* quantum search and gave rise to other quantum search algorithms, each of which if fundamentally different from the more familiar and above all *classical* search algorithms.

In Chap. 3 we explained Grover's algorithm, so as to apply one of its spin-offs, a quantum search algorithm by Dürr and Høyer, in Chap. 4. We showed that the running time of Dürr and Høyer's algorithm, applied to either instance, solves the problem in time $\mathcal{O}(N^4)$. As a result, it is the "fastest" algorithm for finding an edge that maximally decreases the effective graph resistance of the augmented graph. Kim's bisection algorithm, also a clever kind of search, albeit classical, is indeed still better for finding an edge that maximally increases algebraic connectivity.

Finally, also in Chap. 4, we explained how heuristics can be used to speed up Dürr and Høyer's algorithm. Not just any search algorithm can be accelerated by integrating a heuristic, but Dürr and Høyer's can be accelerated and the gain is significant — it seems possible to roughly *halve* the algorithm's running time.

5.1 Future Work

Some obvious directions for future research include the following:

- Apply Dürr and Høyer's algorithm as applied in this work to other optimisation problems, in particular to optimisation problems for which good heuristics exist.
- Devise and measure the performance of new heuristics for solving the graph augmentation problem. A heuristic based on spanning trees, for example, does not seem to have been found while we showed in Sect. 2.3.3 that there is a relationship between the number of spanning trees of the augmented graph and its effective graph resistance.
- Search for a sharp bound for the algebraic connectivity and effective graph resistance of the augmented graph.
- Come up with a more efficient way to compute the algebraic connectivity or effective graph resistance of a given graph. It might be possible, for example, to compute efficiently and recursively the effective graph resistance of some graphs by reducing ("folding up") them by means of the four reductions presented in Sect. 2.3.4.
- Devise a clever *classical* search algorithm for minimising the effective graph resistance of the augmented graph, motivated by Kim's result.
- And last but not least, yet beyond all doubt the most challenging: reveal the problem's hidden structure or prove that there is none.

A slightly less obvious direction for future research is to elaborate further on the relation between the random walk on the graph and the effective resistance. Even less obvious is to try to relate the *quantum random walk* on the graph to the effective (graph) resistance. We have had a try at the latter during the past six months that were spent on this work, and we conclude this text by briefly speculating about what the union of effective (graph) resistance and quantum random walks has to offer.

Recall Eq. 2.12,

$$R_{ij} = \frac{1}{2M} C_{ij} \tag{2.12}$$

where $C_{ij} = \mathbb{E}[T_{ij}] + \mathbb{E}[T_{ji}]$ denotes the commute time between i and j. M is the number of edges $|E|$. Intuitively, the smaller C_{ij}, the poorer the connection between i and j. That is, if C_{ij} is small, the random walk on the graph starting in i has difficulty in reaching j and vice versa.

Think of the random walk on G as a finite time-reversible Markov chain $(X_n)_{n\in\mathbb{N}}$. If we only consider connected non-bipartite graphs, $(X_n)_{n\in\mathbb{N}}$ will

be *positive recurrent*, *irreducible*, and *aperiodic* (for definitions, see for example [40, p. 140]). Consequently, we have

$$\pi_i = \frac{1}{\mu_i(i)} \tag{5.1}$$

where i is a member of the state space $\mathbb{S}$ of $(X_n)_{n\in\mathbb{N}}$, π the limiting and at the same time stationary distribution of the Markov chain, and $\mu_j(i) \geq 1$ the expected time before a random walk on G starting in i hits j (see for example [40, p. 133]). It can be shown that

$$\mu_i(i) = 1 + \sum_{j\in\mathbb{S}\setminus\{i\}} P_{ij}\,\mathbb{E}[T_{ji}] \tag{5.2}$$

where P is the transition matrix of $(X_n)_{n\in\mathbb{N}}$ (see for example [40, p. 104]). Hence,

$$\frac{1}{\pi_i} = 1 + \sum_{j\in\mathbb{S}\setminus\{i\}} P_{ij}\,\mathbb{E}[T_{ji}] \tag{5.3}$$

and since $P_{ij} = 0$ if i is not a neighbour of j and $1/d_i$ otherwise, we have

$$\frac{1}{\pi_i} = 1 + \frac{1}{d_i} \sum_{j\in N(i)} \mathbb{E}[T_{ji}] \tag{5.4}$$

where $N(i)$ is the set of vertices adjacent to vertex i. Eq. 5.4 shows that $1/\pi_i - 1$ is the average of the expected lengths of the random walks starting in i's neighbours and ending in i.

It is well-known that

$$\pi_i = \frac{d_i}{2M} \tag{5.5}$$

and accordingly, combining Eq. 5.4 and Eq. 5.5 gives

$$2M - d_i = \sum_{j\in N(i)} \mathbb{E}[T_{ji}]. \tag{5.6}$$

Equation 5.6 can be used to simplify Eq. 2.7. That is,

$$R_G = \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} R_{ij} = \frac{1}{2M} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \mathbb{E}[T_{ij}] \tag{5.7}$$

$$= \frac{1}{2M} \left(\sum_{(i,j)\in E(G)} (\mathbb{E}[T_{ij}] + \mathbb{E}[T_{ji}]) + \sum_{(i,j)\in \bar{E}(G)} (\mathbb{E}[T_{ij}] + \mathbb{E}[T_{ji}]) \right) \tag{5.8}$$

$$= \frac{1}{2M} \left(\sum_{i\in V(G)} (2M - d_i) + \sum_{(i,j)\in \bar{E}(G)} (\mathbb{E}[T_{ij}] + \mathbb{E}[T_{ji}]) \right) \tag{5.9}$$

$$= \frac{1}{2M} \left(2NM - 2M + \sum_{(i,j)\in \bar{E}(G)} (\mathbb{E}[T_{ij}] + \mathbb{E}[T_{ji}]) \right) \tag{5.10}$$

$$= N - 1 + \frac{1}{2M} \sum_{(i,j)\in \bar{E}(G)} (\mathbb{E}[T_{ij}] + \mathbb{E}[T_{ji}]) \tag{5.11}$$

$$= N - 1 + \sum_{(i,j)\in \bar{E}(G)} R_{ij}. \tag{5.12}$$

Equation 5.12 establishes evidence that if $\bar{E}(G) = \varnothing$, which implies $G = K_N$, then $R_G = R_{K_N} = N - 1$ as was also shown by Ellens et al. (see Table 4.2) albeit in a different way.

The equation also explains why adding the edge (i, j) such that R_{ij} is maximal will probably serve as a good heuristic: if we add (i, j) to G, the large R_{ij} term will be swallowed up completely in the $N - 1$ term in front of the summation. The other effective resistances will also decrease, but to what extent remains obscure. That is also why such a heuristic will be a heuristic and not an algorithm for solving the problem: sometimes the addition of an edge (i, j) such that R_{ij} is *not* maximal will *nevertheless* bring about the largest decrease of all terms in the summation $\sum_{(i,j)\in\bar{E}(G)} R_{ij}$ and accordingly minimise the effective graph resistance of the augmented graph.

Returning to the usefulness of random walks and quantum random walks in solving the second instance of our problem, one will find that if π_i is small, connecting i to any other vertex it is not already connected to will often — but not always — maximally decrease the effective graph resistance of the augmented graph. A similar pattern is observed if we look at the long-run behaviour of the quantum random walk on the graph. Although different kinds of quantum random walks have been defined (see [51] and [29] for the definitions of two types of discrete quantum random walks on general graphs, respectively), usually if not always the components of some state $|\psi\rangle$ are mapped on to V or E and as a result, just as we can classically, a probability can be attached to each vertex or edge. Contrary to random walks, however, quantum random walks in general do not have a limiting or stationary distribution, but the average of the different distributions over

time *does* converge, and for that reason Aharonov et al. [1, p. 53] defined the limiting distribution of a quantum random walk as the average distribution over time t as t grows from 0 to infinity.

In any case, Eq. 5.5 shows that if one were to add edges based on π_i as described in the beginning of the preceding paragraph, one is effectively applying heuristic h_1 (see Sect. 4.4.3). Quantum random walks devised to emulate as closely as possible classical random walks will probably display the same behaviour — that is, low probabilities being indicative of vertices incident with relatively good candidate edges — for the same reason. To conclude, although tempting, the long-run behaviour of both classical and quantum random walks does not seem to be of immediate use. ■

Bibliography

[1] Aharonov, D., Ambainis, A., Kempe, J., and Vazirani, U. Quantum walks on graphs. ACM Press, pp. 50–59.

[2] Archdeacon, D., Colbourn, C. J., Gitler, I., and Provan, J. S. Four-terminal reducibility and projective-planar wye-delta-wye-reducible graphs. *Journal of Graph Theory 33* (2000), 83–93.

[3] Asratian, A. S., Denley, T. M. J., and Haggkvist, R. *Bipartite Graphs and their Applications*. Cambridge University Press, Cambridge, 1998.

[4] Bapat, R. B., Gutmana, I., and Xiao, W. A Simple Method for Computing Resistance Distance. *Zeitschrift für Naturforschung A 58*, 9-10 (2014), 494–498.

[5] Barik, S., and Pati, S. On algebraic connectivity and spectral integral variations of graphs. *Linear Algebra and its Applications 397* (Mar. 2005), 209–222.

[6] Belhaiza, S., de Abreu, N. M. M., Hansen, P., and Oliveira, C. S. Variable Neighborhood Search for Extremal Graphs. XI. Bounds on Algebraic Connectivity. In *Graph Theory and Combinatorial Optimization*, D. Avis, A. Hertz, and O. Marcotte, Eds. Springer US, Boston, MA, 2005, pp. 1–16.

[7] Bollobás, B. *Modern Graph Theory*, vol. 184 of *Graduate Texts in Mathematics*. Springer New York, New York, NY, 1998.

[8] Boyer, M., Brassard, G., Høyer, P., and Tapp, A. Tight Bounds on Quantum Searching. *Fortschritte der Physik 46*, 4-5 (June 1998), 493–505.

[9] Brouwer, A. E., and Haemers, W. H. Graph Spectrum. In *Spectra of Graphs*. Springer New York, New York, NY, 2012, pp. 1–20.

[10] Chandra, A. K., Raghavan, P., Ruzzo, W. L., Smolensky, R., and Tiwari, P. The electrical resistance of a graph captures its com-

mute and cover times. *Computational Complexity 6*, 4 (Dec. 1996), 312–340.

[11] Daoud, S. N. The deletion-contraction method for counting the number of spanning trees of graphs. *The European Physical Journal Plus 130*, 10 (Oct. 2015).

[12] de Abreu, N. M. M. Old and new results on algebraic connectivity of graphs. *Linear Algebra and its Applications 423*, 1 (May 2007), 53–73.

[13] Deng, H. On the minimum Kirchhoff index of graphs with a given number of cut-edges. *MATCH Communications in Mathematical and in Computer Chemistry 63* (2010), 171–180.

[14] Donetti, L., Neri, F., and Muñoz, M. A. Optimal network topologies: expanders, cages, Ramanujan graphs, entangled networks and all that. *Journal of Statistical Mechanics: Theory and Experiment 2006*, 08 (Aug. 2006).

[15] Doyle, P. G., and Snell, J. L. Random walks and electric networks. 118.

[16] Dürr, C., and Høyer, P. A Quantum Algorithm for Finding the Minimum. *arXiv:quant-ph/9607014* (July 1996). arXiv: quant-ph/9607014.

[17] El-Mallah, E. S., and Colbourn, C. J. On two dual classes of planar graphs. *Discrete Mathematics 80*, 1 (Feb. 1990), 21–40.

[18] Ellens, W., and Kooij, R. E. Graph measures and network robustness. *arXiv:1311.5064 [physics]* (Nov. 2013). arXiv: 1311.5064.

[19] Ellens, W., Spieksma, F., Van Mieghem, P., Jamakovic, A., and Kooij, R. Effective graph resistance. *Linear Algebra and its Applications 435*, 10 (Nov. 2011), 2491–2506.

[20] Fan, Y. On Spectral Integral Variations of Graphs. *Linear and Multilinear Algebra 50*, 2 (Jan. 2002), 133–142.

[21] Fiedler, M. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal 23*, 2 (Jan. 1973), 298–305.

[22] Fu-Shang P. Tsen, Ting-Yi Sung, Men-Yang Lin, Lih-Hsing Hsu, and Myrvold, W. Finding the most vital edges with respect to the number of spanning trees. *IEEE Transactions on Reliability 43*, 4 (Dec. 1994), 600–603.

[23] Garey, M. R., and Johnson, D. S. *Computers and intractability: a guide to the theory of NP-completeness*. A series of books in the mathematical sciences. Freeman, New York, 2009.

[24] Godsil, C., and Royle, G. *Algebraic Graph Theory*, vol. 207 of *Graduate Texts in Mathematics*. Springer New York, New York, NY, 2001.

[25] Golub, G. H., and Van Loan, C. F. *Matrix computations*, 4th ed. Johns Hopkins studies in the mathematical sciences. The Johns Hopkins University Press, Baltimore, 2013.

[26] Grover, L. K. A fast quantum mechanical algorithm for database search. ACM Press, pp. 212–219.

[27] Harary, F. *Graph theory*. Perseus Books, Cambridge, Mass, 2001.

[28] Kaye, P., Laflamme, R., and Mosca, M. *An introduction to quantum computing*. Oxford University Press, 2007.

[29] Kendon, V. Quantum walks on general graphs. *International Journal of Quantum Information 04*, 05 (Oct. 2006), 791–805.

[30] Kim, Y. Bisection Algorithm of Increasing Algebraic Connectivity by Adding an Edge. *IEEE Transactions on Automatic Control 55*, 1 (Jan. 2010), 170–174.

[31] Kirkland, S. A bound on the algebraic connectivity of a graph in terms of the number of cutpoints. *Linear and Multilinear Algebra 47*, 1 (Mar. 2000), 93–103.

[32] Kirkland, S. A characterization of spectral integral variation in two places for Laplacian matrices. *Linear and Multilinear Algebra 52*, 2 (Mar. 2004), 79–98.

[33] Kirkland, S. J. An upper bound on algebraic connectivity of graphs with many cutpoints. *Electronic Journal of Linear Algebra 8*, 1 (Jan. 2001).

[34] Klein, D. J., and Randić, M. Resistance distance. *Journal of Mathematical Chemistry 12*, 1 (Dec. 1993), 81–95.

[35] Le Gall, F. Powers of tensors and fast matrix multiplication. ACM Press, pp. 296–303.

[36] Liu, J., Cao, J., Pan, X.-F., and Elaiw, A. The Kirchhoff Index of Hypercubes and Related Complex Networks. *Discrete Dynamics in Nature and Society 2013* (2013), 1–7.

[37] Lukovits, I., Nikolić, S., and Trinajstić, N. Resistance distance in regular graphs. *International Journal of Quantum Chemistry 71*, 3 (1999), 217–225.

[38] Luxburg, U. V., Radl, A., and Hein, M. Getting lost in space: Large sample analysis of the resistance distance. In *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 2622–2630.

[39] Nielsen, M. A., and Chuang, I. L. *Quantum computation and quantum information*, 10th anniversary ed. Cambridge University Press, 2010.

[40] Privault, N. *Understanding Markov chains: examples and applications*. Springer undergraduate mathematics series. Springer, Singapore, 2013.

[41] Qi, F., and Guo, B.-N. Sharp bounds for harmonic numbers. *Applied Mathematics and Computation 218*, 3 (Oct. 2011), 991–995. arXiv: 1002.3856.

[42] Read, K. E. Cultures of the Central Highlands, New Guinea. *Southwestern Journal of Anthropology 10*, 1 (Apr. 1954), 1–43.

[43] Ross, S. M. *Introduction to probability models*, 11th ed. Elsevier, Amsterdam; Boston, 2014.

[44] So, W. Rank one perturbation and its application to the laplacian spectrum of a graph. *Linear and Multilinear Algebra 46*, 3 (Aug. 1999), 193–198.

[45] Strang, G. *Introduction to linear algebra*, 5th ed. Wellesley-Cambridge Press, 2016.

[46] Van der Meer, E. Comparing measures of network robustness, July 2012.

[47] Van Mieghem, P. *Graph spectra for complex networks*. Cambridge University Press, Cambridge, UK; New York, 2011.

[48] Vos, V. S. S. Methods for determining the effective resistance. Master's thesis, Universiteit Leiden, Dec. 2016.

[49] Wang, H., and 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 (ICST, Brussels, Belgium, Belgium, 2008), BIONETICS '08, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 22:1–22:8.

[50] Wang, X., Pournaras, E., Kooij, R. E., and Van Mieghem, P. Improving robustness of complex networks via the effective graph resistance. *The European Physical Journal B 87*, 9 (Sept. 2014).

[51] Watrous, J. Quantum Simulations of Classical Random Walks and Undirected Graph Connectivity. *Journal of Computer and System Sciences 62*, 2 (Mar. 2001), 376–391.