Quantum testing algorithms for graph properties

Author:
Rick van de Hoef

Supervisors:
Prof. dr. R. de Wolf
Prof. dr. H.L. Bodlaender

Department of Information and Computing Sciences
Utrecht University
Netherlands
November 2016

Contents

Chapter 1

Introduction

1.1 Property testing

In the past 20 years, the amount of data that is being processed has exploded. Companies such as Google and Amazon keep track of a lot of user data, and have enormous data centers. For example, Google's data centers are estimated to store multiple exabytes (1 exabyte = 1 million terabytes) of data. Also, experiments in CERN produce so much data, that it was preferable to use magnetic tape, one of the first computer storage media, to reduce costs, amongst other reasons.

Our interest lies with solving decision problems which have enormous amounts of data as input. Processing these amounts of data and looking for a certain property in it is often not feasible in reasonable time: even linear-time algorithms are too slow when the input is huge. To solve this, we look for algorithms that are significantly faster than linear-time algorithms. It is, however, often impossible to get deterministic, sub-linear-time algorithms for these problems, while guaranteeing correctness. Sub-linear-time algorithms do not have time to read the entire input, which might be necessary for some problems in order to guarantee correctness.

Two assumptions arise from this setting. First, since sub-linear-time algorithms cannot process the entire input, they have to choose which part of the input they inspect. This choice introduces a random factor into the algorithms, which has to be allowed. Therefore, the correctness requirement for these sub-linear-time algorithms has to be relaxed; the fast algorithms have to produce the correct answer with high probability, in contrast to always being correct. Second, because these fast algorithms do not process the entire input, it is difficult, or sometimes even impossible, to distinguish between certain boundary cases of input. To this end, the possible inputs are

limited; only yes-instances of the decision problem, and *clear* no-instances are allowed as input, where the meaning of 'clear' is left unspecified for now. In other words, we may assume that our input is not a boundary case. In practice it is of course impossible to check whether the input satisfies this condition before processing it. The sub-linear-time algorithms have to allow boundary cases as their input as well, but they are not as good at giving the correct answer in those cases. We will touch upon this subject after giving the formal definition later on. Using these two assumptions, it is possible to create sub-linear-time algorithms for some problems which we shall refer to as property testing algorithms or property testers, for decision problems.

Aside from when the input is large, property testers also have their use in other situations. For instance, property testers can be used as pre-processing algorithms for decision problems, to quickly sieve the clear no-instances from the possible yes-instances. Due to the nondeterministic nature of property testers, it might be the case that some inputs are processed incorrectly. Depending on the property tester, however, it may only reject no-instances, such that none of the yes-instances is wrongly rejected. Another situation in which property testers may do well is when the input is noisy, i.e. contains random errors. In this situation, one would not necessarily be interested in boundary cases, since it could be that a random error occurred in the input, switching that input from a yes- to a no-instance, or vice versa. A property tester is able to give the correct answer for a decision-problem with high probability, except for the boundary-cases, which in the case of noisy inputs are of little interest anyway.

1.2 Quantum computing

Our goal in this thesis is to use quantum algorithms as property testers. In this section, we introduce some basic concepts from quantum mechanics, as well as give a quick overview of the history of quantum computing.

1.2.1 Quantum mechanics

In 1900 to 1925, a group of physicists developed a new physical theory which described the behaviour of small particles: quantum mechanics. This theory, a set of mathematical rules for describing that behaviour, allows a number of phenomena which are not (often) observed in our everyday lives, and prove to be counterintuitive to humans.

For instance, the concept of *superposition* arises, which says that a particle can be in multiple basis states "at the same time". It is impossible to observe

this behaviour, however, since such a particle will collapse into one of the basis states when it is observed. Because this concept cannot be observed directly it is difficult to imagine it. The famous thought-experiment which tries to illustrate superposition for large objects is that of Schrödinger's cat, in which the cat is in a state of being alive and dead at the same time. The experiment is as follows: in an isolated box, you put a living cat, a particle which is in superposition of decaying and not-decaying, and a mechanism which releases poison and kills the cat when triggered by the decaying particle. The idea is that since the particle is in a state of both decaying and not-decaying, the release of the poison, and therefore the life of the cat, is in a similar state. However, it is impossible to observe such a thing, because when it is observed the superposition of a particle will always collapse to one of its basis states. In the cat's case, it will be either dead or alive upon opening the box.

Let $|1\rangle, |2\rangle, \ldots, |N\rangle$ be the N basis states in which a physical system can be. A quantum state of that system is then a superposition of those N basis states: $\sum_{i=1}^{N} \alpha_i |i\rangle$, where $\alpha_i \in \mathbb{C}$ is the amplitude of the basis state $|i\rangle$. Since observing a quantum state forces it to collapse into one of its basis states, how does it know which state to collapse to? It collapses to a basis state randomly, according to the probability distribution $\Pr[\text{system collapses to state } |i\rangle] = |\alpha_i|^2$. This induces a condition on the amplitudes, namely that $\sum_{i=1}^{N} |\alpha_i|^2 = 1$.

Another important concept is that of *interference*. Quantum states share traits with waves, which are subject to interference with other waves. When you throw two rocks in water, the waves generated by those two rocks will intersect. These intersecting waves interact in two ways: constructive and destructive. When two waves meet in a point and they are both in a crest or both in a trough, they reinforce each other, creating a bigger amplitude for the resulting wave in that point; this is called constructive interference. When the two waves meet in a point when one is in a crest and the other in a trough, they work against each other, and reduce the amplitude of the resulting wave in that point, possibly negating each other entirely; this is destructive interference. Both interactions are illustrated in Figure 1.1.

Quantum states can interact in a similar way. It is possible to manipulate quantum states, such that the amplitudes change, and with them, the probability of the basis state into which the quantum state will collapse. As will be shown in the section on Grover's search algorithm, the trick of quantum computing is to manipulate the amplitudes such that only those of "good" states will be big at the end of the algorithm.

These two concepts are important for quantum computing; together they are part of what makes quantum computing powerful. They are, however, not the only results from quantum mechanics. Another concept worth men-

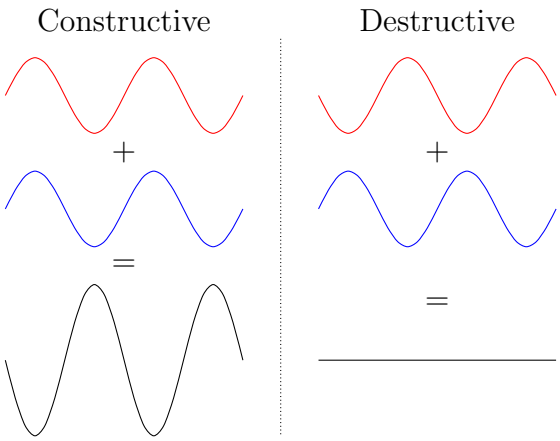

Figure 1.1: Constructive (left) and destructive (right) interference.

tioning is that of *entanglement*, which shows two quantum systems can be linked such that the collapse of one of the two results in the collapse of the other as well, even if the two systems are geographically far apart. The smallest example of entanglement is that of an EPR-pair, named after Einstein, Podolsky, and Rosen [15]. Entanglement is often encountered in the field of quantum information and communication, where this linking of distant quantum systems is very useful.

1.2.2 Quantum computing

The idea of using quantum mechanics for computation didn't come until the early 1980s, when Paul Benioff, Richard Feynman and Yuri Manin all independently came up with the idea of using quantum mechanics for computation. Richard Feynman observed that it appeared to be impossible to simulate quantum systems on classical computers efficiently, and suggested a basic model for a quantum computer which could simulate these systems efficiently. However, this is not the only reason why quantum computers are being studied.

Currently, microchips seem to be following Moore's law, which states that the number of transistors on these chips doubles about every two years. This requires the circuits to be made of such small components, that quantum effects are starting to affect them. Microchip manufacturers try to suppress these effects, because they are not accounted for, and therefore unwanted. At the same time, however, the fact that these effects arise can be seen as a sign suggesting that technology has advanced far enough to produce components that are so small, that quantum mechanics can be used effectively.

Another reason why quantum computers are of interest is the fact that quantum memory allows faster algorithms than is possible with classical computers. Some famous results are that of Peter Shor, who found a polynomial time algorithm for finding prime factors of natural numbers [32], and that of Lov Grover, who gave an algorithm for searching an unordered n-element database in $O(\sqrt{n})$ time [24]. In Section 3.2 we will discuss Grover's search algorithm in depth.

1.3 Contribution and outline of this thesis

In this thesis we focus on the property of Eulerianity[1], and give a quantum algorithm which tests whether a graph is Eulerian, with quantum query complexity $O(n^{1/3}/\sqrt{\epsilon}+\log(1/(\epsilon d))/\epsilon^2)$, which is lower than the lower bound proven for classical property testers. In Chapter 2 we explain the formal definition of property testing, we zoom in on graph property testing and the models used in graph property testing, and show some results. Chapter 3 contains the basics of quantum computing and explains the quantum algorithms used in this thesis. An overview of what is known in quantum property testing (on graphs) is given in Chapter 4, and our quantum algorithm for testing Eulerianity is presented in Chapter 5. Finally, we mention some possible subjects for future work in this area in Chapter 6.

[1] A graph G is Eulerian if there exists a cycle in G which visits every edge exactly once.

Chapter 2

Property Testing

In this chapter we introduce the notion of property testing, and explain its formal definition with some examples in Section 2.1. The focus of this thesis is on graph property testing, which will be introduced and explored further in Section 2.2.

2.1 General property testing

Let us formalise the concept of a property tester, as we introduced in Section 1.1. Following is the meta-definition of property testing, taken from [28]:

Property Testing.

Let $\mathcal{X}$ be a set of objects and $d : \mathcal{X} \times \mathcal{X} \to [0, 1]$ be a distance measure on $\mathcal{X}$. A subset $\mathcal{P} \subseteq \mathcal{X}$ is called a property*. An object $x \in \mathcal{X}$ is* ϵ-far *from $\mathcal{P}$ if $d(x, y) \geq \epsilon$ for all $y \in \mathcal{P}$; x is* ϵ-close *to $\mathcal{P}$ if there is a $y \in \mathcal{P}$ such that $d(x, y) \leq \epsilon$.*
An ϵ-property tester *for $\mathcal{P}$ is an algorithm that receives as input either an $x \in \mathcal{P}$ or an x that is ϵ-far from $\mathcal{P}$. In the former case, the algorithm accepts with probability at least $2/3$; in the latter case, the algorithm rejects with probability at least $2/3$.*[1]

The value for the probability in the above definition is chosen to be $2/3$, but could be changed to any constant in $(1/2, 1)$. The error level can be reduced to any desired level by taking the majority outcome after multiple runs of the algorithm. Note that if an input is accepted with high probability, it has to be ϵ-close to $\mathcal{P}$.

[1] Accepting an input x that lies in $\mathcal{P}$ with probability $\geq 2/3$ makes a property tester *complete*, and the rejecting when $x \notin \mathcal{P}$ with probability $\geq 2/3$ makes it *sound*.

As mentioned, the given definition is a meta-definition, meaning that it is incomplete on its own. We still need to provide definitions for $\mathcal{X}, \mathcal{P}$, and d, in order to be able to apply it. Although not required by the meta-definition, we also need to specify the way in which x can be accessed, as well as how the complexity is measured. A property tester is given access to one or more functions, which act as an oracle, with which the tester can access x. For example, let $\mathcal{X}$ be the set of all n-bit strings, such that $x = x_1 x_2 \dots x_n$. A natural choice for the access function would be a function $f : [n] \to \{0, 1\}$, which simply returns the bit of x at the queried index, i.e. $f(i) = x_i$. Another example when $\mathcal{X}$ is the set of all n-vertex directed graphs, is $g_{in}, g_{out} : [n] \to [n]$, which return the in- and out-degree, respectively, of the vertex given as input to the function. The complexity is then typically measured in the number of queries made to these oracle access functions.

As a simple warm-up example, let us instantiate the meta-definition for deciding whether a bit string is the all-zero string. Since our inputs will be bit strings, $\mathcal{X} = \bigcup_{n\in\mathbb{N}} \{0, 1\}^n$, the set of all bit strings. Then $\mathcal{P}_n = \{0^n\}$, the set containing only the all-zero n-bit string, and we can define $\mathcal{P} = \bigcup_{n\in\mathbb{N}} \mathcal{P}_n$. Let $|x|$ denote the Hamming weight of x.[2] For the distance measure, the natural choice is the normalised Hamming distance, $d_n(x, y) = \frac{|x\oplus y|}{n}$, where $\oplus$ is the XOR-operator. Note that we only compare n-bit input strings with strings of the same length. Also note that an n-bit string that is ϵ-far from the all-zero string is a bit string with at least ϵn 1-bits. To access our bit-string we use the same f we suggested before for bit strings: $f : [n] \to [n], f(i) = x_i$. The complexity is then measured in the number of times we use f. We are now set to give a property tester for testing whether a string is the all-zero string.

Consider the following tester:

Algorithm 2.1.
Pick $2/\epsilon$ indices uniformly at random, and query the input string at those indices. If any of the queries returns a 1-bit, then reject, otherwise accept.

Let us analyse the tester. To show that the tester satisfies the conditions set in the meta-definition, we need to show that it accepts inputs in $\mathcal{P}$ with probability $\geq 2/3$ (completeness), and rejects inputs which are ϵ-far from $\mathcal{P}$ with the same probability (soundness). Notice that the tester always accepts inputs in $\mathcal{P}$, i.e. the all-zero string, in which case we say that the tester has one-sided error. It remains to show that if the input is ϵ-far from $\mathcal{P}$, the tester rejects that input with probability $\geq 2/3$, or equivalently, the tester accepts that input with probability $\leq 1/3$. If the input

[2]The Hamming weight of an n-bit string x is the number of indices where x has a 1-bit, i.e. $\sum_{i=0}^{n} x_i$.

is ϵ-far from $\mathcal{P}$, we know that it contains at least ϵn 1-bits. Therefore, we can upper bound the probability that none of the queries picks a 1-bit: Pr[none of the queries returns a 1-bit] $\leq (1-\epsilon)^{2/\epsilon} < 1/3$,[3] which was exactly what we wanted to show. The number of queries we make to the input is $\Theta(1/\epsilon)$, which is independent of the length of the input. Note that using the complement of $\mathcal{P}$ does not result in the same setting: if $\epsilon > 1/n$, no string is ϵ-far from $\mathcal{P}$ and we could trivially accept all inputs without querying anything.

This was just a simple example, but many more interesting property testers exist. One of the better known testers is that of Blum, Luby and Rubinfeld (BLR) [35], on testing linearity of (Boolean) functions. A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is called linear if and only if $\forall x, y \in \{0,1\}^n$: $f(x \oplus y) = f(x) \oplus f(y)$. Here $\mathcal{X}$ is the set of all n-bit Boolean functions and $\mathcal{P}$ is the set of all n-bit linear Boolean functions. The distance measure between two Boolean functions f and g is the number of inputs on which the two functions differ, divided by 2^n: $\frac{\sum_{x\in\{0,1\}^n} f(x)\oplus g(x)}{2^n}$. It is possible to interpret these Boolean functions as 2^n-bit strings, in which case the distance measure would be the normalised Hamming distance which we also used in the warm-up example. The way we access f is by making calls to it, and the complexity is measured in the number of calls made to f. Checking whether the linearity condition holds for f on all inputs (including boundary cases of f that are very close to being linear), would take $\Omega(2^n)$ calls to f. If, however, the input function is linear or ϵ-far from it, the BLR algorithm can decide, with high probability, whether the function is linear or not, using only a constant number of calls to f. The test is as follows:

Algorithm 2.2 (BLR linearity test).

Choose $x, y \in \{0,1\}^n$ uniformly at random. Query f at the three points x, y and $x \oplus y$. Accept if $f(x \oplus y) = f(x) \oplus f(y)$, and reject otherwise.

The algorithm itself is very simple, and it is clear that it only uses a constant number of queries, since only 3 calls are being made to f.

Linear functions are always accepted by this tester, so it has one-sided error. The main difficulty here, as is often the case, lies in proving the soundness of the property tester. The proof of this is beyond the scope of this thesis, but can be found in [35], where Fourier analysis is used, and in [8], where a more classical, yet longer proof is given.

For further reading on property testing, see [19, 28].

[3]Here we used the inequality $(1 - \frac{x}{y})^y \leq e^{-x}$ for $y > 1,\ |x| \leq y$.

2.2 Graph property testing

As the name suggests, *graph property* testing is about property testing, where $\mathcal{X}$ from the meta-definition from Section 2.1 is the set of all (n-vertex) graphs. In this section, we will introduce the three most used instantiations (also called models) of the meta-definition in graph property testing: the dense graph model (Section 2.2.1), the bounded-degree model (Section 2.2.2), and the general graph model (Section 2.2.3). In each section we will discuss a model, its distance measure, and way of accessing the graph. The complexity measure is the same in all of the models: it is measured in the number of times the graph has been queried by means of its oracle function(s). In every section we will also discuss a property tester for Bipartiteness in that model, to see some property testing algorithms, and also to observe how different models give different results for the same property. As we will see, most models follow from a common representation of graphs, which in turn have natural choices for the distance measure and access functions. The discussion of the three models will be in the context of undirected graphs; directed graphs will be discussed on their own in Section 2.2.4.

2.2.1 Dense graph model

The *dense graph* model is best suited for graphs with many edges, as the name suggests. The representation it uses is effectively an adjacency matrix, which is why this model is sometimes referred to as the adjacency matrix model. From this representation follows a natural choice for the distance measure: the distance between two adjacency matrices is the number of matrix entries in which the two graphs differ, divided by the number of entries in a matrix. In other words, if x and y are two $n \times n$ matrices, the distance to each other is defined as $d_n(x, y) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} \oplus y_{ij}}{n^2}$, which is equal to the normalised Hamming distance. Note that we only compare graphs of the same number of vertices, so the number of entries of both adjacency matrices are equal. The distance measure does not reflect that the adjacency graph is symmetric, as is the case here since we assumed only undirected graphs as input. More often than not, self-loops are not allowed, and this is also not taken into account in the current distance measure. One can solve both these issues by setting the denominator in the distance measure to $\binom{n}{2}$ instead of n^2. The expression, however, becomes more complex, and the difference is not significant, which is why the chosen distance measure is seen more often. In further writing, we will omit the suffix n from the distance measure d when n is clear from the context. From the definition of ϵ-far and the chosen distance measure, it now follows that if a graph $G = (V, E)$ is ϵ-far from $\mathcal{P}$,

then it differs in more than ϵn^2 edges from every $H \in \mathcal{P}$, and so it requires at least that many edge modifications to be made to have the property. Edge modifications can be the removal or insertion of edges.

The way a property tester can access the graph in this model, is by making queries to the adjacency matrix representation. The property tester is given a function $f : [n] \times [n] \to \{0, 1\}$, which represents the adjacency matrix: $f(i, j)$ returns 1 if there is an edge between vertices i and j, and 0 otherwise. Queries of this kind are sometimes referred to as *vertex-pair* queries. Different types of queries are possible, and the problem setting often determines which are used. In addition to f, it is possible that other functions are available which have information about the input graph. For example, it is common to see that a property tester is allowed to query the degrees of vertices and do vertex-pair queries as well.

Bipartiteness in the dense graph model

Here we shall discuss a tester for Bipartiteness in the dense graph model. Recall that a graph $G = (V, E)$ is called *bipartite*, if and only if, there exists a partitioning V_1, V_2 of V, such that $E \subseteq V_1 \times V_2$.

The following tester is due to Goldreich, Goldwasser and Ron [20]:

Algorithm 2.3 (Bipartiteness Tester in the Dense Graph Model).

With input n, ϵ, and oracle access to the adjacency matrix of the n-vertex input graph $G = (V, E)$:

1. *Uniformly select a subset of $\tilde{O}(1/\epsilon^2)$ vertices of V.*[4]
2. *Accept if and only if the subgraph induced by this subset is bipartite.*

Step (2) consists of creating the subgraph in local memory, by querying all pairs in the subset of step (1), and checking whether the subgraph is bipartite, by using Breadth-first search (BFS) to find a cycle of odd length (if it finds one, the subgraph is not bipartite). It is clear that if G is bipartite, the algorithm will accept G with probability 1, and therefore the tester has one-sided error. Proof of soundness can be found in [20]. The query complexity of the algorithm is equal to the number of queries made in step (2), which is $\tilde{O}(1/\epsilon^4)$, since every possible pair of the chosen vertices has to be queried. A more complex analysis (which we skip here) by Alon and Krivelevich [1] shows that the algorithm is still a bipartite tester, if only $\tilde{O}(1/\epsilon)$ vertices

[4]The tilde sign is an indication that a logarithmic factor is omitted, i.e. $\tilde{O}(f(n)) = O(f(n) \cdot \log(f(n))$.

are picked in step (1), so that the query complexity becomes $\tilde{O}(1/\epsilon^2)$. Note that the query complexity is completely independent of the size of the input, and only depends on the proximity parameter ϵ. Because ϵ is often taken to be a small constant, we say that the tester uses only a constant number of queries, independent of n.

2.2.2 Bounded-degree model

The *bounded-degree* model is meant for graphs whose vertices all have a degree bounded by some parameter δ. Its representation is a number of adjacency lists, which is why this model is also referred to as the adjacency list model, or incidence list model. Each list belong to a vertex, and contains all the labels of the vertices it is connected to, in no particular order.

As in the dense graph model, the distance measure follows naturally from the representation, and is again the number of entries in which the two undirected graphs differ, over the maximum number of possible entries in the adjacency lists, δn. From this we can derive that $G = (V, E)$ is ϵ-far from $\mathcal{P}$, if and only if G differs more than $\epsilon\delta n$ edges from every graph $G' \in \mathcal{P}$. Again, the symmetry of undirected graphs is not taken into account, but this changes the computed distance only by a factor 2.

To access the graph in this model, we use a function $f : [n] \times [\delta] \to ([n] \cup \{0\})$, and $f(x, i)$ returns the i-th neighbour of the vertex x, if such a neighbour exists. If x has fewer than i neighbours, the value 0 is returned instead. Note that the neighbours need not be ordered in any particular way, and so it is classically not possible to check whether x is connected to another vertex y with fewer than $O(\delta)$ queries to f. These kind of queries are often referred to as *neighbour* queries.

For this model as well, there are more types of queries possible in addition to the one just explained. One such query uses a function $g : [n] \to ([\delta] \cup \{0\})$ which returns the degree of the vertex passed to g. Note that if no such function is available to the tester, it is possible to query the degree of a vertex in $O(\log \delta)$ queries, using the function f mentioned before in combination with binary search.

Bipartiteness in the bounded-degree model

As opposed to the Bipartiteness tester in the dense graph model, it is not possible to have a Bipartiteness tester in the bounded-degree model which uses only a constant number of queries. In fact, Goldreich and Ron [21] showed that the query complexity for testing Bipartiteness is lower bounded by $\Omega(\sqrt{n})$, where n is the number or vertices in the graph. In a subsequent

work [22], Goldreich and Ron present a Bipartiteness tester for the bounded-degree model, using $\tilde{O}(\text{poly}(1/\epsilon) \cdot \sqrt{n})$ queries:

Algorithm 2.4 (Algorithm Test-Bipartite).

- *Repeat* $\Theta(\frac{1}{\epsilon})$ *times:*
  1. *Uniformly select* s *in* V*.*
  2. *Let* $K = \text{poly}((\log n)/\epsilon) \cdot \sqrt{n}$*, and* $L = \text{poly}((\log n)/\epsilon)$*;*
  3. *Perform* K *random walks starting from* s*, each of length* L*;*
  4. *If some vertex* v *is reached (from* s*) both on a prefix of a random walk corresponding to an even-length path and on a walk-prefix corresponding to an odd-length path, we found a cycle of odd length and we reject. Otherwise, continue.*
- *In case none of the iterations rejected, accept.*

The tester selects $O(1/\epsilon)$ starting vertices and from each starting vertex it performs $\text{poly}((\log n)/\epsilon) \cdot \sqrt{n}$ random walks, each of length $\text{poly}((\log n)/\epsilon)$. The resulting complexity is $\tilde{O}(\text{poly}(1/\epsilon) \cdot \sqrt{n})$, which is essentially tight in terms of the dependency on the size of the input, n.

If the input graph G is bipartite, it is clear that the tester always accepts, since the algorithm tries to find a witness which proves that G isn't bipartite. The tester tries to find such a witness by looking for an odd cycle. The analysis is rather complex, so it won't be discussed here, but can be found in [21].

2.2.3 General graph model

In the previous models we have seen that the distance measure followed from the representation of the graph. In the general graph model, however, these are decoupled, and we have no reasonable absolute point of reference, since we assume neither that the graphs are sparse, nor that they are dense. Hence this model is suited for any type of graph, which is why this model is called the general graph model. The aim of this model was to strengthen the link between property testing and standard algorithmic studies, by being less dependent on the representation of the graph.

Since the distance measure now has no reasonable absolute point of reference, the distance between two graphs has to be relative to their size in terms of the number of edges.

The distance between two graphs is the number of edges they do not share, divided by the maximum number of edges of the two graphs, i.e. $\frac{E \triangle E'}{\max(|E|,|E'|)}$, where $A \triangle B = (A \cup B) \setminus (A \cap B)$. Sometimes the denominator in the distance measure is chosen to be $(|E| + |E'|)/2$ instead. These different choices can give different results, but they are related by a factor 2. In practice it is often the case that only the number of the input graph is known, and not that of the graphs in $\mathcal{P}$. That is why often a graph $G = (V, E)$ in this model is called ϵ-far from $\mathcal{P}$ if and only if G needs at least $\epsilon|E|$ edge modifications in order to be in $\mathcal{P}$.

Now that there is no particular representation that the graph depends on, we are also free to pick query types. In this model both vertex-pair and neighbour queries (queries that were allowed in the previous models) are used.

Bipartiteness in the general graph model

A tester for Bipartiteness in the general graph model is presented in a paper by Kaufman, Krivelevich and Ron [27]. Since the general graph has to take care of both dense and bounded-degree graphs, it is clear that it should also suffer from the drawbacks of those two models. Kaufman et al. are able to construct an algorithm whose complexity is $\tilde{O}(\min(\sqrt{n}, n^2/m))$, where n is the number of vertices in the graph and m the number of edges in the graph.

The algorithm itself is largely based on the tester for bipartiteness in the bounded-degree model, which uses random walks to find a cycle of odd length. If no such cycle is found, the input is claimed to be bipartite. In the paper, the aim is to give a tester for the case where the graph is almost regular, i.e. the maximum degree is of the same order as the average degree. In order to deal with the general case it is then shown that the general case can be reduced to a special case of an almost-regular graph. The proofs are beyond the scope of this introduction, but can be found in [27].

2.2.4 Directed vs. undirected graphs

In the previous sections, the graphs were always undirected graphs. The mentioned models translate naturally to directed graphs, but in the case of neighbour queries, such as in the bounded-degree model, there are two possible extensions.

The first simply involves having versions for both incoming and outgoing edges. Instead of having one adjacency list per vertex, we now have two: a list for incoming and outgoing edges. This means we also need functions to access these different list, i.e. $f_{in} : [n] \times [\delta] \to ([n] \cup \{0\})$ and $f_{out} : [n] \times [\delta] \to$

($[n] \cup \{0\}$), for incoming and outgoing edges, respectively. Similarly, we can have two functions, g_{in} and g_{out}, for checking the in- and out-degrees of vertices, if available in the model.

The second possible extension for the bounded-degree model only allows for one of the two directions of edges; either you can query incoming edges, or outgoing. The latter is often a more natural model for the problem setting. For example when we use web pages as vertices, we will likely use hyperlinks as outgoing edges, since it is difficult (if at all feasible) to gather all web pages that refer to a specific web page. These possible extensions to directed graphs are also called the bidirectional and unidirectional models, respectively.

In a recent paper by Czumaj, Peng and Sohler [13], a link was made between the complexity of the bidirectional model, and that of the unidirectional model. They show that if a property can be tested in the bidirectional model with constant query complexity, the same property can be tested in the unidirectional model with sublinear query complexity.

Chapter 3

Quantum Computing

In this chapter we will explain the basics of quantum computing in Section 3.1, and introduce the quantum algorithms that will be used in this thesis in Section 3.2. Most of the material in this chapter is based on the lecture notes of de Wolf [34], and for more information on quantum computing and quantum algorithms, we refer the reader to [30].

3.1 Quantum computing

3.1.1 Qubits

What is the difference between quantum computers and classical computers, which we use now? Classical computers use bits to store information, and use logical gates, such as AND- and NOT-gates, to manipulate these bits. Quantum computers use quantum bits, or *qubits* for short, which are manipulated by using a different kind of gate.

Qubits are quantum states as described in Section 1.2.1, with two possible basis states, represented by orthogonal 2-dimensional vectors of length 1: $|0\rangle = \binom{1}{0}$ and $|1\rangle = \binom{0}{1}$. A qubit $|\psi\rangle$ therefore represents a quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, with $|\alpha|^2 + |\beta|^2 = 1$, but can also be interpreted as a two-dimensional vector of length 1: $|\psi\rangle = \binom{\alpha}{\beta}$, with $|\alpha|^2 + |\beta|^2 = 1$.

For multiple qubits, we use the tensor product $\otimes$ to combine the basis states into a bigger system.[1] For example, suppose that we have 2

[1] The following definition of a tensor product is taken from [34]: If $A = (A_{ij})$ is an $m \times n$ matrix and B a $p \times q$ matrix, then their *tensor* or *Kronecker* product is the $mp \times nq$ matrix: $A \otimes B = \begin{pmatrix} A_{11}B & \cdots & A_{1n}B \\ A_{21}B & \cdots & A_{2n}B \\ & \ddots & \\ A_{m1}B & \cdots & A_{mn}B \end{pmatrix}$.

qubits. Each of the qubits has two possible basis states, which results in 4 possible combinations. This gives us 4 basis states for a 2-qubit system, just like there are 4 possible classical states for bits: 00, 01, 10, and 11. Similarly, we have the following basis states in a quantum computer: $|0\rangle \otimes |0\rangle = (1,0,0,0)^T$, $|0\rangle \otimes |1\rangle = (0,1,0,0)^T$, $|1\rangle \otimes |0\rangle = (0,0,1,0)^T$ and $|1\rangle \otimes |1\rangle = (0,0,0,1)^T$, where $\mathbf{v}^T$ is the transpose of the vector $\mathbf{v}$. The basis states correspond to 4-dimensional, pairwise orthogonal vectors of length 1. In general, the $\otimes$ operator is omitted and we write $|0\rangle|0\rangle$ or $|00\rangle$ for $|0\rangle \otimes |0\rangle$. For n qubits, we can also write $|0\rangle, |1\rangle, |2\rangle, \ldots, |2^n - 1\rangle$, since any n-bit string can be interpreted as a number from 0 to $2^n - 1$. A general n-qubit state can be written as $\sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$, where $\sum_{x \in \{0,1\}^n} |\alpha_x|^2 = 1$.

At any point during a computation, we can measure one or multiple qubits. As explained in Section 1.2.1, measuring a quantum state will cause it to collapse into one of its basis states according to the probability distribution generated by the squared amplitudes of the corresponding basis states. For example, suppose we have the following two-qubit state $|\zeta\rangle = \frac{1}{\sqrt{3}} |0\rangle|\phi\rangle + \sqrt{\frac{2}{3}} |1\rangle|\psi\rangle$. Measuring the first qubit will collapse the two-qubit state in the state $|0\rangle|\phi\rangle$ with probability 1/3, or in state $|1\rangle|\psi\rangle$ with probability 2/3.

Apart from measuring in the standard basis $|0\rangle$ and $|1\rangle$, it is also possible to measure a qubit in a different orthonormal basis. For example, $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ form an orthonormal basis. When measured in this basis, the quantum state will collapse to either $|-\rangle$ or $|+\rangle$. Although it will not be used in this thesis, it has been put to good use by Bennett et al. in their paper on quantum cryptography [6] and much subsequent work.

3.1.2 Gates

Now we will look into manipulating qubits. Quantum mechanics only allow linear operations to transform quantum states. Since this model represents qubits as vectors, linear operations are modelled by matrices. Not any matrix can be used, however, because the resulting vector represents the qubit(s) after the transformation, and therefore has to have length 1. The matrices have to be length-preserving, also referred to as norm-preserving. A $d \times d$ complex matrix U is called *unitary*, if its conjugate transpose[2] U^* is also its inverse, i.e. $UU^* = U^*U = I$, where I is the $d \times d$ identity matrix. Unitary matrices are length-preserving, and we will refer to unitary matrices on a few qubits as *gates*.

[2]The conjugate transpose of a matrix A is the transpose A, after taking the complex conjugate of its entries.

Here are some examples of gates that work on one qubit:

- X, the bitflip gate: $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$,
- Z, the phaseflip gate: $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$,
- R_ϕ, the phase gate: $\begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$,
- H, the Hadamard gate: $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.

The bitflip gate X is comparable to the classical NOT-gate: it flips the basis states of the qubit on which it is acting. For example, suppose $|\phi\rangle = \alpha |00\rangle + \beta |11\rangle$, and X is applied to the second qubit, the result will be $(I \otimes X) |\phi\rangle = \alpha |01\rangle + \beta |10\rangle$. Note that we used the tensor product of the 2×2 identity matrix and X. Since we only wanted to manipulate the second qubit, all other qubits in the system should stay the same. Mathematically, we can't ignore those qubits, so we apply the identity matrix I to them. To apply these linear operations in parallel, we combine the individual linear operations with the tensor product. If we want to apply gates sequentially, we use matrix multiplication, e.g. if we want to apply U after V on a quantum state $|\phi\rangle$, this will be equal to $UV |\phi\rangle$. In future writing, we shall use $U^{\otimes k}$ to represent the tensor product of k times the linear operation U, i.e. $U \otimes U \otimes \ldots \otimes U$.

The phaseflip gate Z has no equal in classical computing, since classical bits are not subject to phase. In quantum computing, however, phase is important for interference, as mentioned in Section 1.2.1. The phaseflip gate simply switches the polarity of a the basis state $|1\rangle$: $Z |0\rangle = |0\rangle$, $Z |1\rangle = -|1\rangle$. If we view a qubit as a vector in the two-dimensional plane, applying Z would be equal to reflecting through $|0\rangle$.

The phase gate R_ϕ is similar to the phaseflip gate in the sense that it has no classical equivalent. The phase gate, however, rotates the $|1\rangle$ component with an angle of ϕ in the two-dimensional plane. In fact, Z is a special case of the phase gate where the phase is rotated by an angle π.

The Hadamard gate H is a very common gate, even though its use is not immediately apparent. Let us apply H to the 1-qubit basis states:

$$H |0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad H |1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

Since H is unitary and equal to its conjugate transpose H^*, H is its own inverse. The second application of H should therefore result in the starting

state. Let us verify this for $|0\rangle$:

$$H(H|0\rangle) = \frac{1}{\sqrt{2}}H(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}(H|0\rangle + H|1\rangle)$$
$$= \frac{1}{2}((|0\rangle + |1\rangle) + (|0\rangle - |1\rangle)) = |0\rangle\,.$$

Here we see an example of destructive interference; the phase of the first $|1\rangle$ negates that of the second.

In case of $H|0\rangle$, we have a uniform superposition over all the possible 1-qubit basis states; $|0\rangle$ and $|1\rangle$. In general, it is the case that for n qubits, $H^{\otimes n}|i\rangle = \frac{1}{\sqrt{2^n}}\sum_{j\in\{0,1\}^n}(-1)^{i\cdot j}|j\rangle$, where $i \cdot j = \sum_{k=1}^{n} i_k \cdot j_k$ is the inner product on the n-bit strings i and j. In particular, it follows that:

$$H^{\otimes n}|0^n\rangle = \frac{1}{\sqrt{2^n}}\sum_{j\in\{0,1\}^n}(-1)^{0\cdot j}|j\rangle = \frac{1}{\sqrt{2^n}}\sum_{j\in\{0,1\}^n}|j\rangle\,,$$

meaning that $H^{\otimes n}|0^n\rangle$ is equal to the uniform superposition of all 2^n basis states of n-qubit strings. This is often used at the beginning of algorithms, to set up the state of the algorithm, such that every possible string j is considered. It is then up to the algorithm to use interference in a clever way.

We can also construct gates that work on multiple qubits together. These gates take more than 1 qubit as input, and produce the same number of qubits as output. We will show two common gates for more than 1 qubit, and we start with the CNOT gate:

$$\text{CNOT, controlled-not gate: } \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

The controlled-not gate CNOT is a conditioned bitflip gate: if the first qubit is $|0\rangle$, nothing happens, but if it is $|1\rangle$, the second qubit is flipped. In fact, for every unitary U, there exists a controlled version of it, which we refer to as a controlled-U gate. For controlled gates it is important to know which bit is used to check whether U has to be applied, called the control-bit. The qubit(s) which may be transformed, is called the target-bit(s). We assume that the first of the two qubits is the control-bit, unless stated otherwise. Consider the following example of a controlled-not gate. Let $|\phi\rangle = \alpha|01\rangle + \beta|11\rangle$, and let C be the controlled-not gate. If we apply C to $|\phi\rangle$, where the first qubit is the control-bit, this will result in:

$$C|\phi\rangle = \alpha(C|01\rangle) + \beta(C|11\rangle) = \alpha|01\rangle + \beta|10\rangle\,.$$

The second multiple-qubit gate we want to show is the CCNOT gate:

$$\text{CCNOT, Toffoli gate: } \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

The controlled-controlled-not gate CCNOT, also known as the Toffoli gate, is similar to the controlled-not gate. Instead of two, it takes three qubits as its input, and flips the third qubit, if and only if, the first two are both $|1\rangle$.

Using length-preserving linear operations with real entries implies that there is an infinite number of possible gates, which in practice can't all possibly be implemented exactly. To tackle this problem, some sets of gates have been considered for usage:

1. The set of all 1-qubit unitaries together with the 2-qubit CNOT gate,
2. The set consisting of the controlled-not, Hadamard, and the phase-gate $R_{\pi/4}$,
3. The set consisting of the Hadamard and Toffoli gates.

The first set is universal, meaning that any unitary can be built from these gates, but it still contains an infinite number of gates. The second set is universal with approximation, meaning that any unitary can be arbitrarily well approximated by these gates, efficiently, as a result from the Solovay-Kitaev theorem. The same holds for the third set, if we only consider unitary matrices that have real-numbered entries. From a theoretical point of view, we are satisfied knowing that it is possible to approximate the necessary gates efficiently with only a small set of gates, but we will use any unitary (with real entries) in algorithms as needed, for simplification.

3.1.3 Quantum parallelism

One of the advantages of quantum computing we want to highlight, is that of quantum parallelism. Suppose there exists a classical circuit for computing some function $f : \{0,1\}^n \to \{0,1\}^m$, and remember that the NAND-gate is universal for classical circuits. Since we can simulate a NAND-gate with quantum gates (using a Toffoli gate, and setting the target-bit to $|1\rangle$), any

classical circuit can be recreated, using qubits and quantum gates. Therefore, we can build a quantum circuit U that computes f as well, i.e. a mapping $U : |x\rangle |0^m\rangle \mapsto |x\rangle |f(x)\rangle$, $\forall x \in \{0,1\}^n$. So far, this can also be done by the classical circuit, but if $|\phi\rangle$ is the uniform superposition of all 2^n possible states, U computes f for all those states simultaneously:

$$U |\phi\rangle = U \left(\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |0^m\rangle \right) = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle .$$

By itself, however, the resulting state is not very useful: measurement of the final state would result in collapsing the state into a computed value for a single random $x \in \{0,1\}^n$. By combining quantum parallelism with interference, it becomes possible to get speed-ups over classical computing.

3.2 Quantum algorithms

In this section we explain the quantum algorithms used in this thesis. In particular, we shall look at Grover's database search algorithm, and an extension to it, which takes care of variable search times.

Almost every quantum algorithm deals with queries, which we shall explain here. Consider an N-bit string input $x = (x_1, \ldots, x_N) \in \{0,1\}^N$. Usually, $N = 2^n$, so that the bit x_i can be addressed by the n-bit string representing an index i. Access to the input x is through a "black-box" interface, which is modelled as a quantum operation that works on $n+1$ bits:

$$O_x : |i, b\rangle \mapsto |i, b \oplus x_i\rangle ,$$

where $\oplus$ is addition modulo 2.

Using this quantum operation to access the memory, it is also possible to do queries which map $|i\rangle \mapsto (-1)^{x_i} |i\rangle$ by setting the target bit to $|-\rangle = H |1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$:

$$\begin{aligned} O_x(|i\rangle|-\rangle) &= \frac{1}{\sqrt{2}} O_x |i, 0\rangle - \frac{1}{\sqrt{2}} O_x |i, 1\rangle \\ &= \frac{1}{\sqrt{2}} |i, x_i\rangle - \frac{1}{\sqrt{2}} |i, 1 - x_i\rangle \\ &= |i\rangle \frac{1}{\sqrt{2}} (|x_i\rangle - |1 - x_i\rangle) = (-1)^{x_i} |i\rangle|-\rangle . \end{aligned}$$

This kind of query is often more convenient than the regular query, as we shall see in Grover's search algorithm. When we wish to use this kind of oracle, we will write $O_{x,\pm}$ as the unitary for the map $|i\rangle \mapsto (-1)^{x_i} |i\rangle$.

3.2.1 Grover's search algorithm

Consider the following problem. Let x be an $N = 2^n$ bit string, i.e. $x \in \{0,1\}^N$. Our goal is to find an index i such that $x_i = 1$, if there is such an i, or to output that there isn't one otherwise. In 1996, Lov Grover [24] invented his quantum search algorithm, which uses $O(\sqrt{N})$ queries to the memory. This is optimal up to a constant, since a lower bound on the number of queries of $\Omega(\sqrt{N})$ was proven by Bennett et al. in 1993 [5], interestingly, two years before Grover's algorithm was invented. The quadratic speed-up compared to classical algorithms is not as big as some other quantum algorithms provide, but because searching is such a basic task in many algorithms, it is widely applicable.

The idea behind the algorithm is as follows. The solution is going to be one of the indices, so we take n qubits, and to start, we set them to the uniform superposition of all possible indices. Then we do the Grover iteration a number of times, each iteration using interference to increase the amplitude of those indices at which there is a 1-bit. After a well-chosen number of Grover iterations, we measure the qubits, collapsing them to a basis state $|i\rangle$, and check whether x_i is a 1-bit. If it is a 1-bit, we output i, otherwise we output that there is no 1-bit in the string x.

Before the actual algorithm and analysis are given, a few definitions are needed. As described in the idea behind the algorithm, we divide the indices into two groups, the "good" and the "bad" indices. Similarly, we will name the basis states good or bad, depending on what kind of index they represent. Let t be the number of good states, we define three quantum states:

- $|U\rangle = \frac{1}{\sqrt{2^n}} \sum_{i \in \{0,1\}^n} |i\rangle$, the uniform superposition of all possible indices,
- $|B\rangle = \frac{1}{\sqrt{N-t}} \sum_{i:x_i=0} |i\rangle$, the superposition of all bad indices,
- $|G\rangle = \frac{1}{\sqrt{t}} \sum_{i:x_i=1} |i\rangle$, the superposition of all good indices.

Lastly, we will use R to denote the unitary transformation that maps every state $|i\rangle \mapsto -|i\rangle$, $i \in \{0,1\}^n \setminus \{0^n\}$, and maps 0^n to itself.

Suppose that we know that there are t 1-bits in x (we will deal with this assumption later on). The algorithm and analysis are then as follows (taken from [34]):

Algorithm 3.1 (Grover's Search).

1. *Setup the starting state* $|U\rangle = H^{\otimes n} |0^n\rangle$
2. *Repeat the following* $k = O(\sqrt{N/t})$ *times:*

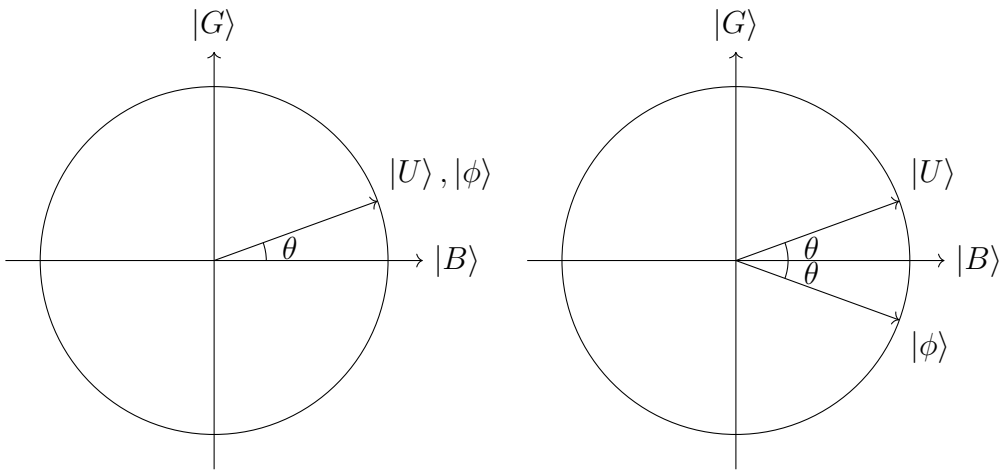

Figure 3.1: (left) The starting state of the algorithm. (right) The state after the reflection through $|B\rangle$ during the first iteration.

(a) Apply $O_{x,\pm}$, i.e. reflect through $|B\rangle$

(b) Apply $H^{\otimes n}RH^{\otimes n}$, i.e. reflect through $|U\rangle$

3. Measure and check that the resulting i is a solution

To explain that this algorithm is correct, we will first show that $O_{x,\pm}$ and $H^{\otimes n}RH^{\otimes n}$ are reflections through $|B\rangle$ and $|U\rangle$, respectively, in the space spanned by $|B\rangle$ and $|G\rangle$. A reflection through a vector u is a unitary A such that $Av = v$ for $v = u$, and $Aw = -w$ for all w orthogonal to u. The application of $O_{x,\pm}$ is by definition the reflection through $|B\rangle$: the phase of every good state is flipped, while every bad state remains the same. For the reflection through $|U\rangle$, we use the known formula for a reflection through a subspace V: $2|V\rangle\langle V| - I$.[3] In our case, this resolves to

$$\begin{aligned} 2|U\rangle\langle U| - I &= 2(H^{\otimes n}|0\rangle)(\langle 0|H^{\otimes n}) - H^{\otimes n}IH^{\otimes n} \\ &= H^{\otimes n}(2|0\rangle\langle 0| - I)H^{\otimes n} = H^{\otimes n}RH^{\otimes n}, \end{aligned}$$

which is precisely the transformation we apply in order to reflect through $|U\rangle$.

The following analysis will be geometric in nature and although there is an equivalent algebraic one, the geometric version is a bit more intuitive and easier to follow. Before the first iteration, the situation is as in the left side of Figure 3.1, where $|\phi\rangle$ is the current state of the algorithm. Since

[3]Like $|\cdot\rangle$, $\langle\cdot|$ is part of the bra-ket notation used in physics, also called Dirac notation. Both represent vectors, but $\langle\cdot|$ is the conjugate transpose of $|\cdot\rangle$, i.e. $|v\rangle = \langle v|^*$.

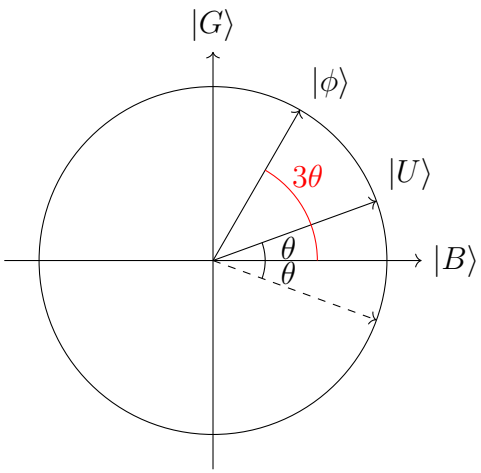

Figure 3.2: The state after the reflection through $|U\rangle$ during the first iteration.

$|B\rangle$ and $|G\rangle$ form a 2-dimensional basis, we can write $|U\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle = \sin(\theta)\,|G\rangle + \cos(\theta)\,|B\rangle$, for $\theta = \arcsin(\sqrt{t/N})$. Let us see what happens to $|\phi\rangle$ during a Grover iteration. After applying $O_{x,\pm}$, $|\phi\rangle$ has been reflected through $|B\rangle$, and we end up in the situation illustrated in the right side of Figure 3.1. After the reflection through $|U\rangle$, we end up in the situation in which $|\phi\rangle$ is at an angle of 3θ with $|B\rangle$. After another iteration, $|\phi\rangle$ will be at an angle of 5θ, and so on. Every Grover iteration rotates the state of the program towards the "good" state $|G\rangle$, by an angle of 2θ. After k iterations, $|\phi\rangle$ will have gone from $|U\rangle$ to the state $\sin((2k+1)\theta)\,|G\rangle + \cos((2k+1)\theta)\,|B\rangle$. The probability of seeing a good state after measuring $|phi\rangle$, is $P_k = \sin^2((2k+1)\theta)$, which we want to be as close to 1 as possible. If we choose $\tilde{k} = \pi/4\theta - 1/2$, then $(2\tilde{k}+1)\theta = \pi/2$, which results in a probability of exactly 1, for finding a good solution. However, $\pi/4\theta - 1/2$ will usually not be an integer, and so we have to take the integer closest to $\tilde{k}$. The error probability will still be small, if t is sufficiently small:

$$\begin{aligned} 1 - P_k &= \cos^2((2k+1)\theta) = \cos^2((2\tilde{k}+1)\theta + 2(k-\tilde{k})\theta) \\ &= \cos^2(\pi/2 + 2(k-\tilde{k})\theta) = \sin^2(2(k-\tilde{k})\theta) \leq \sin^2(\theta) = \frac{t}{N}, \end{aligned}$$

where we used $|k - \tilde{k}| \leq 1/2$ in the last inequality.

In the above analysis, we assumed that t, the number of solutions, was known. This way it was possible to compute θ, and in turn, k: the optimal number of Grover iterations. If t is not known beforehand, it is not possible to compute the optimal k, and we have to use systematic guesses. The expected

number of queries we have to do in this case, is still $O(\sqrt{N})$, as is analysed by Boyer et al. in [9].

3.2.2 Amplitude amplification

The underlying idea in Grover's search algorithm of moving the computer's state towards the superposition of the good states, can also be applied to other search-related situations. Suppose we have a function $\chi : \mathbb{Z} \to \{0, 1\}$ for which any $z \in \mathbb{Z}$ is called a solution if $\chi(z) = 1$, and an algorithm to check whether a given z is a solution, which we write as O_χ. If an algorithm (classical or quantum) finds a solution with probability p, we would normally need to do about $1/p$ runs before we would find a solution. Using amplitude amplification we can find a solution with only $O(\sqrt{1/p})$ runs of the quantum algorithm $\mathcal{A}$ that simulates the algorithm. Amplitude amplification is a generalisation of the method used in Grover's algorithm, and works as follows:

1. Setup the starting state $|U\rangle = \mathcal{A}|0\rangle$
2. Repeat the following $O(1/\sqrt{p})$ times:
   (a) Apply O_χ (i.e. reflect through $|B\rangle$)
   (b) Apply $\mathcal{A}R\mathcal{A}^*$ (i.e. reflect through $|U\rangle$)
3. Measure and check whether the resulting z is a solution.

As we did in the analysis for Grover's algorithm, let us check that $\mathcal{A}R\mathcal{A}^*$ is the reflection through $|U\rangle$ by rewriting the known formula for a reflection through a subspace:

$$\begin{aligned} 2|U\rangle\langle U| - I &= 2(\mathcal{A}|0\rangle)(\langle 0|\mathcal{A}^*) - \mathcal{A}I\mathcal{A}^* \\ &= \mathcal{A}(2|0\rangle\langle 0| - I)\mathcal{A}^* = \mathcal{A}R\mathcal{A}^*. \end{aligned}$$

The analysis is analogous to the one done for Grover's algorithm. In fact, Grover's algorithm is a special case of amplitude amplification, where $\mathcal{A} = H^{\otimes n}$ is a simple algorithm with success probability p, and O_χ is the query to the input string.

3.2.3 Quantum search with variable search times

Another result we shall use in this thesis, is that of quantum search with variable search times by Ambainis [2]. In Grover's algorithm we made the assumption that the cost of evaluating x_i was the same for all i. The problem setting in Ambainis's algorithm is similar to that of Grover's algorithm,

although instead of N bits, we have N items, and to evaluate an item x_i, i.e. apply $O_{x,\pm}$, we need t_i steps, which is not necessarily the same for all i.

A naive way to solve this problem, would be to use Grover's search and to take the worst-case scenario in which each evaluation has maximum cost $t_{max} = \max_{i\in\{0,1\}^n} t_i$. This results in an $O(\sqrt{N}t_{max})$ step algorithm. In case the evaluation times t_i are known, we can use the quantum algorithm of Ambainis to reduce this to $O(\sqrt{t_1^2 + t_2^2 + \cdots + t_N^2})$ steps. If the t_i are not known, Ambainis's algorithm has an additional polylogarithmic overhead.

In this thesis we will use Ambainis's quantum search algorithm to search through a number of lists. The cost we are interested in in this case will be the number of quantum queries that have to be made. The situation is as follows: we have $2k$ lists of lengths $\gamma_1, \ldots, \gamma_{2k}$, and evaluating an item in a list has constant query cost. Using Grover's algorithm on a list $j \in \{1, \ldots, 2k\}$ will cost $O(\sqrt{\gamma_j})$ queries. Hence the cost of evaluating a list j is $O(\sqrt{\gamma_j})$, and when we plug this into Ambainis's algorithm, we see that the number of queries to evaluate $2k$ lists is $O\left(\sqrt{\sum_{j=1}^{2k}(\sqrt{\gamma_j})^2}\right) = O\left(\sqrt{\sum_{j=1}^{2k}\gamma_j}\right)$.

Chapter 4

Quantum Property Testing

In this chapter, we look at the combination of the two areas discussed in Chapters 2 and 3: quantum property testing. Quantum property testers, or quantum testers, are quantum algorithms which satisfy the conditions of property testers, as defined in Section 2.1. In Section 4.1, we discuss some general results in quantum property testing, and in Section 4.2 we zoom in on quantum property testers whose input are graphs.

4.1 General quantum property testing

The notion of *quantum property testing*, as opposed to classical property testing, was first defined by Buhrman et al. [11]. Their definition is a modified version of the classical property testing definition by Goldreich in his survey [18]: instead of using the original definition of M as a probabilistic oracle machine, they allow M to be a quantum oracle machine.

Buhrman et al. [11] continued by giving a number of results, showcasing some of the possibilities and limitations of quantum property testing. Their first result is an example of the power of property testing: there exists a property that requires $O(1/\epsilon)$ queries for a quantum tester, for which any classical tester requires $\Omega(\log n)$ queries. The example given is that of a random subset of the Hadamard code, which is an error-correcting code used to detect and correct errors which occur during transmission of messages. Let $N = 2^n$. The Hadamard code of a string $y \in \{0,1\}^n$ is an N-bit string x, defined as $x_i = y \cdot i \mod 2$. The Bernstein-Vazirani quantum algorithm [7] can find y using only 1 query, while classical algorithms need $\Omega(n)$ queries. The algorithm is as follows:

Algorithm 4.1 (Bernstein-Vazirani algorithm).

1. *Start with the state* $|0^n\rangle$.
2. *Apply Hadamard gates to every qubit: the resulting state will be*
$$\frac{1}{\sqrt{2^n}} \sum_{i \in \{0,1\}^n} |i\rangle .$$
3. *Do a phase query: the resulting state becomes*
$$\frac{1}{\sqrt{2^n}} \sum_{i \in \{0,1\}^n} (-1)^{x_i} |i\rangle = \frac{1}{\sqrt{2^n}} \sum_{i \in \{0,1\}^n} (-1)^{y \cdot i} |i\rangle .$$
4. *Apply Hadamard gates to all qubits again, and the resulting state will be* $|y\rangle$.[1]

Buhrman et al. construct for a subset $A \subseteq \{0,1\}^n$ of messages, the property $\mathcal{P}_A = \{x \in \{0,1\}^N : \exists y \in A \text{ such that } x = h(y)\}$, which contains all Hadamard codewords for the strings in A. Using the Bernstein-Vazirani algorithm, they prove that for all $A \subseteq \{0,1\}^n$ there exists a quantum tester for $\mathcal{P}_A$ which uses $O(1/\epsilon)$ queries, and has one-sided error. They also prove that for most A of size $|A| = N/2$, a classical tester for $\mathcal{P}_A$ requires $\Omega(n)$ queries, even if they have two-sided error.

For more information and proofs, see [7, 11, 28].

The second result proven in [11], is that there exist quantum testers which are exponentially faster than their best classical counterparts. Again, a quantum decision algorithm with the required separation is used to design a property. Again, let $N = 2^n$. Consider a function $f : \{0,1\}^n \to \{0,1\}^n$, with the promise that there exists an $s \in \{0,1\}^n \setminus \{0^n\}$ such that $x = y \oplus s$ if and only if $f(x) = f(y)$, where $y \oplus s$ is the bitwise addition modulo 2. Simon's problem [33] is to find s. Simon's algorithm [33] solves this problem with $\Theta(n)$ queries, by using a subroutine which results in an equation $s \cdot y = 0$, for some y. The algorithm is as follows:

Algorithm 4.2 (Simon's algorithm).

1. *Repeat the following subroutine until* $n - 1$ *linearly independent equations have been obtained*

[1] Here we have used the fact from Chapter 3 that $H^{\otimes n} |j\rangle = \frac{1}{\sqrt{2^n}} \sum_{i \in \{0,1\}^n} (-1)^{j \cdot i} |i\rangle$.

(a) *Start with the state* $|0^n\rangle\,|0^n\rangle$*, and apply Hadamard gates to the first* n *qubits, so that we end up with the state* $\frac{1}{\sqrt{2^n}}\sum_{i\in\{0,1\}^n}|i\rangle\,|0^n\rangle$.

(b) *Query* f *on the first register of* n *qubits, storing the result in the second register:* $\frac{1}{\sqrt{2^n}}\sum_{i\in\{0,1\}^n}|i\rangle\,|f(i)\rangle$.

(c) *Measure the second register, such that the state collapses to* $\frac{1}{\sqrt{2}}(|i\rangle+|i\oplus s\rangle)\,|f(i)\rangle$*, for some* $i\in\{0,\ldots,2^n-1\}$.

(d) *Ignore the second register, and apply Hadamard gates to the first* n *qubits, obtaining:*

$$\begin{aligned}
&\frac{1}{\sqrt{2}}\left(H^{\otimes n}|i\rangle+H^{\otimes n}|i\oplus s\rangle\right) &&= \\
&\frac{1}{\sqrt{2^{n+1}}}\left(\sum_{j\in\{0,1\}^n}(-1)^{i\cdot j}|j\rangle+\sum_{j\in\{0,1\}^n}(-1)^{(i\oplus s)\cdot j}|j\rangle\right) &&= \\
&\frac{1}{\sqrt{2^{n+1}}}\left(\sum_{j\in\{0,1\}^n}(-1)^{i\cdot j}(1+(-1)^{s\cdot j})\,|j\rangle\right).
\end{aligned}$$

(e) *Measure the* n *qubits to obtain* y*, resulting in a linear equation:* $y\cdot s=0$.

2. *Solve, for* s*, the system of* $n-1$ *linearly independent equations obtained from step 1, using Gaussian elimination.*

The reason why the resulting y at the end of the subroutine satisfies $s\cdot y=0$, is because the only basis states $|j\rangle$ with a non-zero amplitude, are those for which $s\cdot j=0$. Brassard and Høyer [10] created a modified version of Simon's algorithm, which finds an equation that is linearly independent from previous equations found for each run of the subroutine. In turn, Buhrman et al. use the algorithm to give a tester which uses $O(n\log n)$ queries, for the property

$$L=\{f:\{0,1\}^n\to\{0,1\}^n\mid\exists s\in\{0,1\}^n\backslash\{0^n\}\;\forall x\in\{0,1\}^n\;f(x)=f(x\oplus s)\}.$$

They also prove that any classical tester for L uses $\Omega(\sqrt{N})$ queries, which demonstrates the exponential separation in query complexity. For more information and proofs, see [33, 10, 11, 28].

The last result from [11] we mention here, is that there exist properties for which there does not exist an efficient quantum tester. In fact, Buhrman et al. prove that most properties containing $2^{n/20}$ elements from $\{0,1\}^n$ require $\Omega(n)$ queries.

After these results from Buhrman et al. [11], a lot of research has been done regarding quantum testers. For example, significant results have been achieved in the areas of juntas, linearity of Boolean functions, group theoretic properties, and more. Note that the testers discussed here only consider classical properties. There also exist quantum testers for quantum properties and classical testers for quantum properties. For more information on quantum property testing, see the survey by Montanaro and de Wolf [28].

4.2 Quantum property testing on graphs

Compared to classical graph property testing, relatively little research has been done on quantum property testing on graphs. Only a small number of papers on quantum property testing concern graphs, each of which we will discuss. In the following, let the input graph $G = (V, E)$.

In the work of Chakraborty et al. [12], the problem of graph isomorphism is studied. The authors take the classical tester for graph isomorphism developed by Fischer and Matsliah [16] as their starting point. The model used for the classical tester, is that of the dense graph model: graphs are represented by their adjacency matrices. The graphs can be accessed by querying a single entry at a time, and the goal is to determine whether two graphs G and H are isomorphic, or ϵ-far from being isomorphic, meaning that in order for them to be isomorphic, one would need to modify at least an ϵ-fraction of the entries in their adjacency matrices. In their work [16], Fischer and Matsliah considered two cases:

- **unknown-unknown case.** Both G and H are unknown, and they can only be accessed by querying their adjacency matrices.
- **known-unknown case.** The graph H is known (given in advance to the tester), and the graph G is unknown (can only be accessed by querying its adjacency matrix).

For the known-unknown case they give nearly tight bounds of $\tilde{\Theta}(\sqrt{|V|})$ on the query complexity. For the unknown-unknown case they give a classical tester that needs $\tilde{O}(|V|^{5/4})$ queries, and prove a lower bound of $\Omega(|V|)$. Chakraborty et al. [12] focus on the bottleneck of the classical tester: the subroutine that checks how much two probability distributions are alike. By using quantum queries, they manage to create quantum testers which have query complexity $\tilde{O}(|V|^{1/3})$ in the known-unknown case, and between $\Omega(|V|^{1/3})$ and $\tilde{O}(|V|^{7/6})$ in the unknown-unknown case.

Recently, Harrow and Montanaro [25] discovered a way to, given a state $|\psi\rangle$ and a set of n measurements, determine with high probability whether one

of the measurements would accept $|\psi\rangle$. Since measurements possibly modify the state, this is not simply applying the measurements one after another. In their paper [25], they mention a number of applications for this technique, one of them being the testing of graph isomorphism. Using the new technique, the query complexity of testing graph isomorphism has been reduced to $O((|V| \log |V|)/\epsilon)$.

Ambainis, Childs and Liu [3] provide quantum speedups for the testers of Bipartiteness and Expander[2] [3] in the bounded-degree model. The classical testers for Bipartiteness [22] and Expander [23, 14, 26, 29] use random walks to construct two sets of vertices. These sets are then checked for common vertices, also called *collisions*. The query complexity for these classical testers is $\tilde{O}(\sqrt{N})$, where N is the number of vertices in the graph. This is almost optimal, as both problems have query complexity lower bounds of $\Omega(\sqrt{N})$. For more detail on the Bipartiteness tester, see Section 2.2.2. Ambainis et al. [3] use the element distinctness quantum algorithm, also by Ambainis [4], which given a set of elements checks whether all elements in the set are distinct. As a result, they speed-up the check for collisions and end up with quantum algorithms for testing both Bipartiteness and Expansion with query complexities of $\tilde{O}(N^{1/3})$, which are polynomially better than the classical testers' lower bounds. They also provide a lower bound on the query complexity of a quantum tester for Expansion of $\Omega(N^{1/4})$, proving that there is no exponential speedup possible for that property. They have, however, been unable to prove a lower bound for Bipartiteness.

[2] A graph G is an α-expander if for every $U \subseteq V$ with $|U| \leq |V|/2$, we have $|\partial(U)| \geq \alpha|U|$, where $\partial(U)$ is the set of vertices in $V \setminus U$ adjacent to at least one vertex in U.

Chapter 5

A Quantum Tester for Eulerianity

In this chapter we look at the results obtained from our research on finding a quantum property tester for Eulerianity that has lower query complexity than is classically possible.

In Section 5.1 we focus on explaining the Eulerianity property and the model that is used, as well as present some lemma's about graphs that are ϵ-far from Eulerianity. The best classical tester known for Eulerianity has query complexity $\tilde{O}(\sqrt{n/\epsilon^3})$ with a lower bound of $\Omega(\sqrt{n/\epsilon})$ where n is the number of vertices in the graph, and is discussed in Section 5.2. In Section 5.3 we show that there exists a quantum tester for Eulerianity which has quantum query complexity $O(n^{1/3}/\sqrt{\epsilon} + \log(1/(\epsilon d))/\epsilon^2)$ where d is half of the average degree vertex in the input graph.

5.1 Eulerianity

Eulerianity is a graph property: a graph is called Eulerian, if it contains a cycle that traverses every edge in the graph exactly once. The definition was first discussed in 1736, when Euler was solving the Seven Bridges of Königsberg problem. The problem statement asked whether it was possible to take a walk through the city of Königsberg, on which you would cross every bridge in the city exactly once and return to the starting point. Euler proved that this was impossible: he modelled the city as a graph, where the city areas were vertices and the bridges were represented by edges, and showed that for a graph to be Eulerian, every vertex needs to have even degree. He also hypothesised that every (undirected) connected graph of which all vertices had even degree, were Eulerian, which was later proven by

Carl Hierholzer in 1873. In the case of directed graphs, there is a theorem that says that a graph is Eulerian if and only if every vertex has equal in and outdegree, and all vertices with non-zero degree belong to a single connected component.

The model we use for the testers is the general graph model with *directed* graphs without parallel edges. The distance measure depends on the number of edges in the graph, meaning that a graph with m edges is ϵ-far from being Eulerian, if at least ϵm edge modifications (adding/removing an edge) are needed to make G Eulerian. The queries that the tester may use, are:

- vertex-pair queries: for $u, v \in V$ check whether $(u, v) \in E$,
- neighbour queries: for $v \in V,\ i \in [n]$ return the i-th neighbour of v,
- degree queries: for $v \in V$, return the in or outdegree of v.

Each of these queries has cost 1.

We shall now prove some properties from [31] that we will need later on. In the following, let $G = (V, E)$ be the input graph, and let $|V| = n$ and $|E| = m$. For any $v \in V$, we write $d^+(v)$ for the indegree of v, and $d^-(v)$ for its outdegree, and let $d = \frac{m}{n}$ be the average degree of the graph G. We shall also write *components*, or *connected components*, for sets of vertices that are connected in the underlying undirected graph G^U.[1]

When a graph is ϵ-far from being Eulerian, we know that following lemma from [31]:

Lemma 5.1. *Let G be a directed graph with m edges that is ϵ-far from being Eulerian. Then at least one of the following holds:*

- *G has more than $\frac{\epsilon}{8}m$ biased edges.*
- *The number of connected components in G is greater than $\frac{\epsilon}{8}m$.*

Proof. We proceed with proof by contradiction. Assume that G is ϵ-far from being Eulerian and has at most $\frac{\epsilon}{8}m$ biased edges and has at most $\frac{\epsilon}{8}m$ undirected connected components. We shall show that it is possible to make G Eulerian with fewer than ϵm edge modifications. First, for every vertex v we will make the indegree of v equal to its outdegree, and we will refer to this new graph as G'. After creating G', we will connect the underlying undirected graph of G' and conclude that the newly formed graph is Eulerian.

[1] The underlying directed graph G^U of a directed graph G, is the graph G with the directions on its edges dropped. Note that G^U can have parallel edges, unlike G itself, if there are pairs of vertices that have edges in both directions.

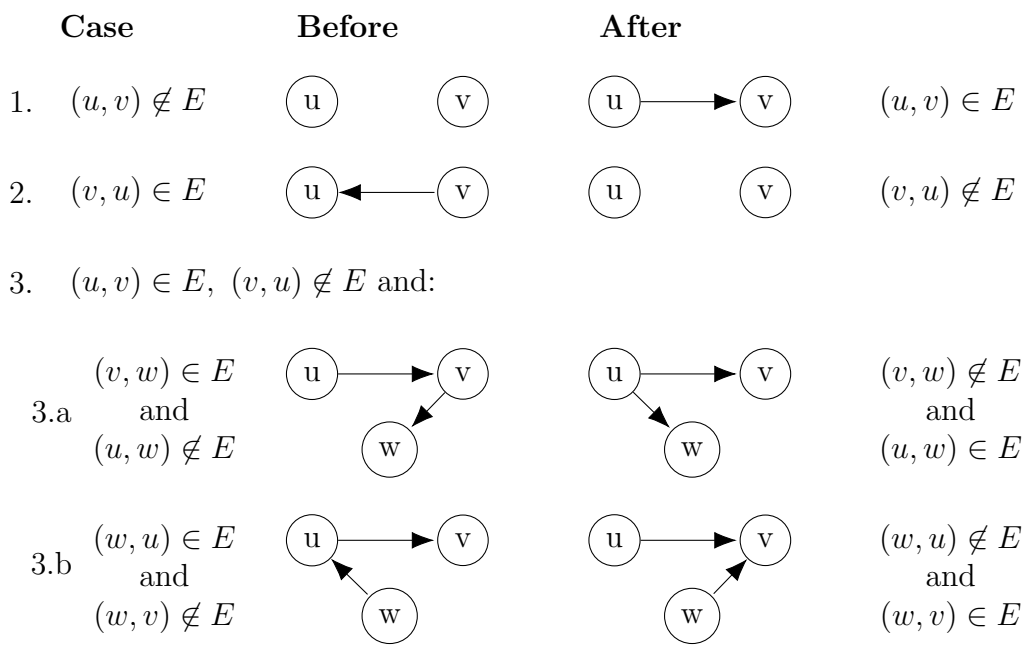

Figure 5.1: An illustration for the process of inducing degree equality.

To fix the degrees of the vertices, note that if there is a vertex $u \in V$ with $d^+(u) > d^-(u)$, there must be a vertex $v \in V$ with $d^-(v) > d^+(v)$. Consider such u, v and their possible cases (for an illustration, see Figure 5.1):

1. $(u,v) \notin E$, we add the edge (u,v).
2. $(v,u) \in E$, we remove the edge (v,u).
3. $(u,v) \in E$ and $(v,u) \notin E$. We consider two subcases.
   (a) There exists a vertex a such that $(v,a) \in E$ (so that necessarily $a \neq u$) and $(u,a) \notin E$, then we remove (v,a) and add (u,a).
   (b) There exists a vertex a such that $(a,u) \in E$ (so that necessarily $a \neq v$) and $(a,v) \notin E$, then we remove the edge (a,u) and add the edge (a,v).

It is clear that the conditions for the three cases together cover all possibilities. It remains to show that necessarily at least one of the two subconditions in case 3 holds. Suppose, contrary to the claim, that neither of the two conditions holds. Since for every a such that $(v,a) \in E$, we have that $(u,a) \in E$ and $(u,v) \in E$, we get that $d^-(u) > d^-(v)$. With the same reasoning, because

we have that for every a such that $(a, u) \in E$ we have that $(a, v) \in E$, and $(u, v) \in E$, we get that $d^+(v) > d^+(u)$. But since $d^+(u) > d^-(u)$, this implies that $d^+(v) > d^-(v)$, which contradicts our assumption that $d^-(v) > d^+(v)$.

By our assumption that G has at most $\frac{\epsilon}{8}m$ biased edges, we have that

$$(1/2)\sum_{v \in V : v \text{ is biased}} |d^+(v) + d^-(v)| = \#\text{ biased edges} \leq \frac{\epsilon}{8}m.$$

Using the method described above, we reduce this sum in steps of two, using at most 2 edge modifications per step. When all vertices have equal indegree and outdegree, then $\sum_{v \in V} |d^+(v) - d^-(v)| = 0$. Since

$$\begin{aligned}\sum_{v \in V} |d^+(v) - d^-(v)| &= \sum_{v \in V:\ v \text{ is biased}} |d^+(v) - d^-(v)| \\ &\leq \sum_{v \in V:\ v \text{ is biased}} |d^+(v) + d^-(v)| \leq \frac{\epsilon}{4}m,\end{aligned}$$

the total cost of reducing it to 0 is at most $\frac{\epsilon}{4}m$ edge modifications. Let the resulting graph be denoted by G'.

We now proceed to connect the undirected components. By fixing the degree inequalities, we removed at most $\frac{\epsilon}{8}m$ edges, meaning that in the worst case, the number of components has grown to at most $\frac{\epsilon}{4}m$. From each of these components, we select a representative vertex, and we add edges to make a directed cycle over the representative vertices. Now the components are connected, and we used at most $\frac{\epsilon}{4}m$ edge modifications to do so. Note that the in and outdegrees of the representative vertices remain equal, but are increased by 1.

The resulting graph is now Eulerian, and since we used at most $\frac{\epsilon}{4}m + \frac{\epsilon}{4}m = \frac{\epsilon m}{2}$ edge modifications, it follows that G is $\frac{\epsilon}{2}$-close to being Eulerian, which contradicts our assumption that G was ϵ-far from being Eulerian. □

The following claim is also from [31].

Claim 5.1. *If a graph G has more than $\frac{\epsilon}{8}m$ undirected connected components, then G has at least $\frac{\epsilon}{16}m$ undirected connected components each containing fewer than $\frac{16}{\epsilon d}$ vertices.*

Proof. The claim follows from a simple counting argument: if G has fewer than $\frac{\epsilon}{16}m$ components each of which contains fewer than $\frac{16}{\epsilon d}$ vertices, it follows that G has more than $\frac{\epsilon}{16}m$ components with at least $\frac{16}{\epsilon d}$ vertices, meaning that G has more than $\frac{\epsilon}{16}m \cdot \frac{16}{\epsilon d} = n$ vertices, which contradicts the assumption that G contains n vertices. □

5.2 Classical tester for Eulerianity

The following classical algorithm and analysis are due to Orenstein and Ron [31]. Their algorithm has query complexity $\tilde{O}(\sqrt{n}/\epsilon^{3/2})$, and they prove a lower bound of $\Omega(\sqrt{n/\epsilon})$ queries. The model they use is as described above.

As stated before, there is a theorem that tells us that in order to prove that a directed graph G is Eulerian, it is sufficient to show that its underlying undirected graph G^U (the same graph as G, but with the directions of the edges dropped) is connected, and every vertex has equal in and outdegrees. Note that this allows isolated vertices in G with no incoming or outgoing edges. The algorithm from [31] uses these criteria and therefore has two phases: the first phase tests if the in and outdegrees of some of the vertices are equal, while the second phase looks for a disconnected component. In their paper, Orenstein and Ron [31] define *biased vertices* as vertices whose indegree does not equal their outdegree, and *biased edges* as edges of which at least one vertex is biased.

First, we will state a necessary lemma for the algorithm, taken from [27]:

Lemma 5.2. *There exists a procedure* Sample-Edges-almost-Uniformly-in-G *that uses* $\tilde{O}(\sqrt{n/\delta})$ *degree and neighbour queries in* G *and for which the following holds: For all but* $(\delta/4)m$ *of the edges* e *in* G*, the probability that the procedure outputs* e *is at least* $1/(64m)$*. Furthermore, there exists a subset* $U_0 \subset V(G), |U_0| \leq (\delta n/2)$*, such that for all edges* $(u, v) \in E$ *that are output with probability less than* $1/(64m)$*, we have* $u, v \in U_0$*.*

This procedure Sample-Edges-Almost-Uniformly-in-G is used in the algorithm for testing Eulerianity, and the proof of Lemma 5.2 can be found in [27]. Let $G = (V, E)$ be the directed input graph, $|V| = n$, $|E| = m$, and $d = \frac{m}{n}$ the average degree. The algorithm is then as follows:

Algorithm 5.1 (Classical Eulerianity tester [31] (input: G, ϵ and d)).

Phase 1: Checking for biased vertices

1. *Sample* $s = \frac{2048}{\epsilon}$ *edges by running Sample-Edges-Almost-Uniformly-in-G with the parameter* δ *set to* $\epsilon/4$*.*

2. *For each sampled edge check if one of its endpoints is a biased vertex (by performing degree queries), implying that it is a biased edge.*

3. *If a biased edge is found, then reject.*

Phase 2: Checking connectivity

4. *For $i = 1$ to $\ell = \log(16/\epsilon d)$ do:*
   (a) *Uniformly and independently select $s_i = \frac{64\log(16/\epsilon d)}{2^i \epsilon d}$ vertices:*
   (b) *From each sampled vertex perform a Breadth-First Search (BFS) on the underlying undirected graph G^U. Stop the BFS when 2^i vertices have been reached or it is impossible to continue the search.*
5. *If any of the above searches stopped before reaching 2^i vertices, then reject.*
6. *If no step caused rejection, accept.*

The following theorem is the result from Orenstein and Ron [31] on testing Eulerianity in directed graphs for which we sketch its proof.

Theorem 5.1. *Algorithm 5.1 is a property tester for Eulerianity in the model described in Section 5.1.*

Proof. Let G be the input graph. The algorithm clearly accepts any G that is Eulerian, so it remains to show that every G that is ϵ-far from being Eulerian, is rejected with probability at least 2/3. For the remainder of this proof, assume that G is ϵ-far from being Eulerian. It follows from Lemma 5.1 that G has more than $\frac{\epsilon}{8}m$ biased edges, or G has more than $\frac{\epsilon}{8}m$ connected components.

In the first case, there are at least $\frac{\epsilon}{8}m$ biased edges, and phase 1 of the algorithm will reject with probability at least 2/3. By sampling $\frac{2048}{\epsilon}$ edges "almost uniformly", using the procedure from Lemma 5.2 with the parameter δ set to $\frac{\epsilon}{4}$, with high constant probability, a biased edge will be selected with probability $1 - (1 - (1/64) \cdot (\epsilon/16))^{2048/\epsilon} > 2/3$, which will lead to rejection of G.

In the second case, there are at least $\frac{\epsilon}{8}m$ components, and phase 2 will reject with probability at least 2/3. Let B_i be the set of components which contain at least 2^{i-1} and at most $2^i - 1$ vertices. By Claim 5.1, there are at least $\frac{\epsilon}{16}m$ components which contain fewer than $\frac{16}{\epsilon d}$ vertices. Let $\ell = \log(16/(\epsilon d))$, then $\sum_{i=1}^{\ell} |B_i| \geq \frac{\epsilon}{16}m$. Note that it is unlikely that ℓ is an integer, and although it is not a big problem in the analysis, we will assume that ℓ is an integer for simplicity. Therefore, there must exist a $j \in [\ell]$, such that $|B_j| \geq \frac{\epsilon m}{16\ell}$. The probability that a uniformly random vertex $v \in V$ belongs to a component in B_j is:

$$\frac{\sum_{C \in B_j} |C|}{n} \geq \frac{|B_j| \cdot 2^{j-1}}{n} \geq \frac{2^{j-1} \cdot \epsilon m}{16\ell n} = \frac{2^j \epsilon d}{32\ell}.$$

In phase 2 of the algorithm, for $i \in [\ell]$, $s_i = \frac{64\ell}{2^i \epsilon d}$ vertices are picked uniformly at random. For $i = j$, the probability that none of those s_j vertices lies in $\bigcup_{C \in B_j} C$, is at most:

$$\left(1 - \frac{2^j \epsilon d}{32\ell}\right)^{s_j} = \left(1 - \frac{2}{s_j}\right)^{s_j} \leq e^{-2} < 1/3.$$

Therefore, the probability of sampling one of the vertices that lie in a component in B_j is at least $1 - e^{-2} > 2/3$. Let v be the vertex that is sampled from a component in B_j. The BFS will start from v and search that component until 2^j vertices have been reached or until it can't reach any vertices that haven't been reached during the BFS. Since the component in which v lies contains at most $2^j - 1$ vertices, the BFS will get stuck and this will cause Phase 2 to reject G. □

Now we compute the query complexity of Algorithm 5.1. Phase 1 uses the procedure Sample-Edges-Almost-Uniformly from Lemma 5.2 with $\delta = \frac{\epsilon}{4}$ which uses $\tilde{O}(\sqrt{n/\delta}) = \tilde{O}(\sqrt{n/\epsilon})$ queries. Since that procedure is called $\frac{2048}{\epsilon}$ times, phase 1 uses $\tilde{O}(\sqrt{n}/\epsilon^{3/2})$ queries. The query complexity of phase 2 is bounded by the sum over the possible values of ℓ, of the number of vertices sampled times the maximum number of vertices to look for, times the maximum degree of every vertex, i.e.

$$\sum_{i=1}^{\ell} s_i \cdot 2^i \cdot 2^i = \left(\frac{64 \log(16/\epsilon d)}{\epsilon d}\right) \cdot \sum_{i=1}^{\ell} 2^i = O\left(\frac{\log(1/\epsilon d)^2}{\epsilon^2}\right).$$

The total query complexity is therefore

$$\tilde{O}\left(\sqrt{\frac{n}{\epsilon^3}} + \frac{\log(1/\epsilon d)^2}{\epsilon^2}\right).^2$$

5.3 Quantum tester for Eulerianity

Our main result is a quantum tester for Eulerianity with query complexity $O(n^{1/3}/\sqrt{\epsilon} + \log(1/(\epsilon d))/\epsilon^2)$, which is lower than the classical lower bound in terms of the dependency on n. The input and the model are the same as for the classical tester.

The quantum tester is given below:

[2]The calculated query complexity is not the upper bound of $\tilde{O}(\sqrt{n}/\epsilon^{3/2})$ that Orenstein and Ron [31] claim. However, we were unsuccessful in reproducing their analysis, so we work with the slightly higher upper bound.

Algorithm 5.2 (Quantum Eulerianity tester (input: G, ϵ and $d = \frac{m}{n}$)).

Phase 1: Checking for biased vertices

1. *If $d > n^{1/3}$, apply amplitude amplification for $O(n^{1/3}/\sqrt{\epsilon})$ iterations to the following procedure:*
   - *Pick $(u, v) \in V \times V$ uniformly at random.*
   - *Check if (u, v) is biased, by querying and comparing the in and out degrees of both u and v.*
2. *If a biased edge is found, reject.*
3. *If $d \leq n^{1/3}$, do the following:*
   - *Apply amplitude amplification for $O(n^{1/3})$ iterations to the following procedure:*
     - *Pick $v \in V$ uniformly at random.*
     - *Check whether v is biased by querying and comparing its in and outdegree.*
   - *If a biased vertex is found, reject.*
   - *Choose $k = \frac{16n^{1/3}}{\epsilon}$ vertices uniformly at random.*
   - *Use Ambainis' algorithm from Section 3.2.3 to search for a biased vertex in both the in and outdegree adjacency lists of the k vertices.*
   - *If a biased vertex is found, reject.*

Phase 2: Checking connectivity

4. *Do the same as phase 2 in Algorithm 5.1.*

The correctness of Algorithm 5.2 is captured in the following theorem:

Theorem 5.2. *Algorithm 5.2 is a quantum property tester for Eulerianity in the general model.*

Proof. Since the algorithm only looks for witnesses to the claim that the input graph G is not Eulerian, it is clear that any graph that is Eulerian will be accepted. It remains to show that if G is ϵ-far from being Eulerian, Algorithm 5.2 rejects G with probability at least 2/3.

For the following, assume that G is ϵ-far from being Eulerian. From Lemma 5.1 it follows that either G has more than $\frac{\epsilon}{8}m$ biased edges, or G has more than $\frac{\epsilon}{8}m$ components. Since phase 2 of Algorithm 5.2 is the same as

that of the classical tester, Algorithm 5.1, we know from Theorem 5.1 that in the case that G has more than $\frac{\epsilon}{8}m$ components, the quantum tester will reject G with probability at least $2/3$.

We proceed to prove that, assuming G has more than $\frac{\epsilon}{8}m$ biased edges, the quantum tester rejects G with probability at least $2/3$. Let β be the number of biased vertices in G. We consider the following cases:

1. $d > n^{1/3}$,
2. $d \leq n^{1/3}$ and $\beta \geq n^{1/3}$,
3. $d \leq n^{1/3}$ and $\beta < n^{1/3}$.

Case 1: $d > n^{1/3}$.
In this case, the probability that a random vertex pair is an edge in G is relatively high, and a reasonable portion of those edges is biased. The quantum tester picks a random pair of vertices $(u, v) \in V \times V$, and uses amplitude amplification to amplify the states which correspond to biased edges. The probability that (u, v) is a biased edge is at least $p_1 = \frac{\epsilon m}{8n^2} = \frac{\epsilon d}{8n} > \frac{\epsilon n^{1/3}}{8n} = \frac{\epsilon}{8n^{2/3}}$. Using amplitude amplification, we find a biased edge with high probability after $O(\sqrt{1/p_1}) = O(\sqrt{\frac{8n^{2/3}}{\epsilon}}) = O(\frac{n^{1/3}}{\sqrt{\epsilon}})$ iterations.

Case 2: $d \leq n^{1/3}$ and $\beta \geq n^{1/3}$.
In this case, the number of biased vertices β is relatively high, and we have a decent probability to pick a biased vertex when picking uniformly at random. The quantum tester picks a vertex $v \in V$ uniformly at random, and uses amplitude amplification to boost the states corresponding to biased vertices. The probability that v is a biased vertex is at least $p_2 = \frac{\beta}{n} \geq \frac{1}{n^{2/3}}$. Using amplitude amplification, we find a biased vertex with high probability, after $O(\sqrt{1/p_2}) = O(n^{1/3})$ iterations.

Case 3: $d \leq n^{1/3}$ and $\beta < n^{1/3}$.
In this case, the graph is pretty sparse and the number of biased vertices is low. However, since Lemma 5.1 still gives us that there must be at least $\frac{\epsilon}{8}m$ biased edges, it must be the case that some biased vertices have high degrees, and are therefore connected to many vertices. The degrees of vertices are low on average, and so it is feasible to search through their adjacency lists. Note that if $V_B \subseteq V$ is the set of biased vertices in V, then by Lemma 5.1: $\frac{1}{2}\sum_{v \in V_B} d^-(v) + d^+(v) > \frac{\epsilon m}{8}$. Since there are at most $n^{1/3}$ biased vertices, it follows that there is a $u \in V_B$ such that $d^-(u) + d^+(u) > \frac{\epsilon m}{4\beta} > \frac{\epsilon m}{4n^{1/3}} \geq \frac{\epsilon n}{4n^{1/3}} = \frac{\epsilon n^{2/3}}{4}$, where we have used that $m \geq n$, since otherwise we know from the start that the graph does not contain a cycle. This implies that at least $\frac{\epsilon n^{2/3}}{8}$ different vertices are connected to u. Choosing $k = \frac{16n^{1/3}}{\epsilon}$

vertices uniformly at random, the probability that none of those k vertices is connected to u, is at most $(1 - \frac{\epsilon}{8n^{1/3}})^k = (1 - \frac{2}{k})^k \leq e^{-2} < 1/3$. Using the quantum search with variable search times from Section 3.2.3 we find a biased vertex in the adjacency lists of the k vertices with probability at least $2/3$.

The three cases together cover all possibilities, and so phase 1 rejects G with probability at least $2/3$ if G is ϵ-far from Eulerianity. □

The query complexity of Phase 1 is as follows. Case 1 uses a constant number of queries in each iteration of the amplitude amplification, to check whether a state is a biased edge. This results in $O(\sqrt{\frac{n^{2/3}}{\epsilon}})$ queries for this case. Case 2 also uses a constant number of queries in each iteration of the amplitude amplification, but in this case to check whether a state is a biased vertex. This results in $O(\sqrt{n^{2/3}})$ queries.

Case 3 requires a bit more analysis. Note that to evaluate an adjacency list entry, the algorithm uses a constant number of queries to check whether that vertex is biased or not, by querying and comparing its in and outdegree. The k vertices we picked can be connected to a biased vertex with an incoming or outgoing edge, so we have to check both those lists. As in Section 3.2.3, we can search the $2k$ lists of lengths $\gamma_1, \gamma_2, \ldots, \gamma_{2k}$ with Ambainis' quantum search algorithm, using $O(\sqrt{\sum_{j=1}^{2k} \gamma_j})$ queries. We know that the average length of the adjacency lists is $d/2 = \frac{m}{2n}$, since every vertex is connected to d other vertices on average. If we view the sum of the lengths γ_i as a random variable X, we can use Markov's inequality[3] to upper bound the sum of the lengths, and therefore the number of queries the algorithm needs to perform. The expected value is

$$E(X) = E\left(\sum_{j=1}^{2k} \gamma_j\right) = \sum_{j=1}^{2k} E(\gamma_j) = \sum_{j=1}^{2k} d/2 = kd,$$

and so it follows that,

$$\Pr[X \geq 100kd] \leq kd/(100kd) = 1/100.$$

Putting it all together, we let the algorithm use at most $\sqrt{\sum_{i=1}^{2k} \gamma_i} < \sqrt{100kd} = \sqrt{100\frac{16n^{1/3}d}{\epsilon}} \leq O(\sqrt{\frac{n^{2/3}}{\epsilon}})$ queries in Phase 1. There is a $1/100$ probability that Phase 1 is not finished after that many queries, and in that case we stop Phase 1 and proceed as if no biased vertex has been found.

[3]Markov's inequality: If X is a non-negative random variable, then $\Pr[X \geq a] \leq E(X)/a$, for every $a > 0$.

The query complexity for Phase 2 is the same as we saw in Section 5.2: $O\left(\frac{\log(16/(\epsilon d))}{\epsilon^2}\right)$. The total complexity therefore is

$$O\left(2\sqrt{\frac{n^{2/3}}{\epsilon}} + \sqrt{n^{2/3}} + \frac{\log(16/(\epsilon d))}{\epsilon^2}\right) = O\left(\frac{n^{1/3}}{\sqrt{\epsilon}} + \frac{\log(16/(\epsilon d))}{\epsilon^2}\right).$$

We see that our quantum tester has only a factor $n^{1/3}$ in its query complexity, whereas the lower bound for classical testers contains a factor $\sqrt{n}$. This means that our quantum tester has a polynomial speed-up compared to the best known, or possible, classical tester for Eulerianity.

Chapter 6

Future Work

As mentioned in Chapter 4, little work has been done on quantum property testing on graphs as of yet, and so there is a lot of opportunity for future work to be done there. Below we list a number of subjects which might be of interest for further research.

- **A better tester for Eulerianity?** In this thesis we found a quantum tester for Eulerianity, but we do not know if it is optimal in terms of query complexity. Phase 1 of the quantum tester may be improved in order to get an even better (i.e. smaller) dependence on n. Phase 2 might also benefit from quantum algorithms, such as quantum BFS [17], but could possibly already have query complexity $O(\frac{1}{\epsilon})$ if the analysis from Orenstein and Ron [31] is correct.

- **A quantum lower bound for Eulerianity.** This thesis has focused on finding a quantum tester for Eulerianity, with an upper bound on the query complexity that was below the classical lower bound from [31], in terms of the dependence on n. In this we succeeded, but it is possible that there exists a quantum tester with an even lower query complexity. Finding a quantum lower bound, however, is difficult, and might be worthy of research on its own. Ambainis et al. [3] found quantum testers for both Bipartiteness and Expander in the bounded-degree model, but could only give a lower bound on the complexity for the Expander property. This demonstrates that finding a quantum lower bound is not straightforward, but the techniques used in [3] might be useful in the case of Eulerianity.

- **Bipartiteness in the general graph model.** Ambainis et al. [3] found a quantum tester for Bipartiteness in the bounded-degree model. Their algorithm is based on the ideas of the classical algorithm, for

which they managed to find significant speed-ups. Kaufman et al. give a classical tester for Bipartiteness in the general graph model, and a lower bound on the query complexity (see Section 2.2.3). The classical tester transforms the input graph in such a way, that it can then use the solutions from the bounded-degree and dense models to test for Bipartiteness. Using the insights gained from Ambainis et al. and this thesis, it might be possible to find a quantum tester with a query complexity below the classical lower bound of $\Omega(\min(\sqrt{n}, n^2/m))$.

- **Relation between quantum uni- and bidirectional graph models.** Czumaj et al. recently published their result which demonstrates a relationship between testing in the uni- and bidirectional bounded-degree graph model. Among their results they claim that if a property can be tested with constant query complexity in the bidirectional model, it can be tested with sub-linear query complexity in the unidirectional model. They also prove that the same does not necessarily hold for digraphs with an arbitrary maximum degree: there exist digraph properties which can be tested with a constant number of queries in the bidirectional model, but require an almost linear number of queries in the unidirectional model. It would be interesting to see if the first claim can be improved upon in the quantum setting, and if the second claim still holds in the quantum setting.

- **Find quantum testers for more graph properties.** As mentioned, not a lot of papers exist on the topic of quantum graph property testing. A good start would be to look for quantum testers which perform better than is possible in the classical setting. This may eventually lead to more general insights in this field.

Bibliography

[1] N. Alon and M. Krivelevich. Testing k-colorability. *SIAM Journal on Discrete Mathematics*, 15(2):211–227, 2002.

[2] A. Ambainis. Quantum search with variable times. *Theory of Computing Systems*, 47(3):786–807, 2010.

[3] A. Ambainis, A. M. Childs, and Y.-K. Liu. Quantum property testing for bounded-degree graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 365–376. Springer, 2011.

[4] Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007.

[5] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997.

[6] C. H. Bennett, G. Brassard, S. Breidbart, and S. Wiesner. Quantum cryptography, or unforgeable subway tokens. In *Advances in Cryptology*, pages 267–275. Springer, 1983.

[7] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

[8] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 73–83. ACM, 1990.

[9] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998. Earlier version in Physcomp '96, quant-ph/9605034.

[10] G. Brassard and P. Hoyer. An exact quantum polynomial-time algorithm for Simon's problem. In *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems, 1997*, pages 12–23, 1997.

[11] H. Buhrman, L. Fortnow, I. Newman, and H. Röhrig. Quantum property testing. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 480–488. Society for Industrial and Applied Mathematics, 2003.

[12] S. Chakraborty, E. Fischer, A. Matsliah, and R. de Wolf. New results on quantum property testing. In *30th Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, pages 145–156, 2010.

[13] A. Czumaj, P. Peng, and C. Sohler. Relating two property testing models for bounded degree directed graphs. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1033–1045. ACM, 2016.

[14] A. Czumaj and C. Sohler. Testing expansion in bounded-degree graphs. *Combinatorics, Probability and Computing*, 19(5-6):693–709, 2010.

[15] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Physical review*, 47(10):777, 1935.

[16] E. Fischer and A. Matsliah. Testing graph isomorphism. *SIAM Journal on Computing*, 38(1):207–225, 2008.

[17] Bartholomew Furrow. A panoply of quantum algorithms. *Quantum Information & Computation*, 8(8):834–859, 2008.

[18] O. Goldreich. Combinatorial property testing (a survey), 1997. `http://www.wisdom.weizmann.ac.il/~oded/PS/testSU.ps`.

[19] O. Goldreich. Introduction to testing graph properties. In *Property testing*, pages 105–141. Springer, 2010.

[20] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.

[21] O. Goldreich and D. Ron. Property testing in bounded degree graphs. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 406–415. ACM, 1997.

[22] O. Goldreich and D. Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.

[23] Oded Goldreich and Dana Ron. On testing expansion in bounded-degree graphs. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 68–75. Springer, 2011.

[24] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[25] A. Harrow and A. Montanaro. Sequential measurements, disturbance and property testing, 2016. *arXiv preprint arXiv:1607.03236*, 74:75.

[26] S. Kale and C. Seshadhri. Testing expansion in bounded degree graphs. *35th ICALP*, pages 527–538, 2008.

[27] T. Kaufman, M. Krivelevich, and D. Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM Journal on computing*, 33(6):1441–1483, 2004.

[28] A. Montanaro and R. de Wolf. A survey of quantum property testing. *Theory of Computing, Graduate Surveys 7*, 2016.

[29] A. Nachmias and A. Shapira. Testing the expansion of a graph. *Information and Computation*, 208(4):309–314, 2010.

[30] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[31] Y. Orenstein and D. Ron. Testing Eulerianity and connectivity in directed sparse graphs. *Theoretical Computer Science*, 412(45):6390–6408, 2011.

[32] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. Earlier version in FOCS'94.

[33] D. R. Simon. On the power of quantum computation. *SIAM journal on computing*, 26(5):1474–1483, 1997.

[34] R. de Wolf. Quantum computing: Lecture notes. (still being updated) `http://homepages.cwi.nl/~rdewolf/qcnotes.pdf`.

[35] R. de Wolf. A brief introduction to Fourier analysis on the Boolean cube. *Theory of Computing, Graduate Surveys*, 1:1–20, 2008.