Quantum Computation and Isomorphism Testing

David J. Rosenbaum

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2015

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Quantum Computation and Isomorphism Testing

David J. Rosenbaum

Co-Chairs of the Supervisory Committee:
Professor Paul W. Beame
Computer Science & Engineering

Affiliate Assistant Professor Aram Wettroth Harrow
Computer Science & Engineering

In this thesis, we study quantum computation and algorithms for isomorphism problems. Some of the problems that we cover are fundamentally quantum and therefore require quantum techniques. For other problems, classical approaches are more appropriate; however, we often give quantum variants of our classical algorithms as well.

The field of quantum computation aims to accelerate algorithms by exploiting the laws of quantum mechanics. Several quantum algorithms are known that are exponentially faster than the best classical algorithms known for the same problems, including several which are relevant to cryptography. Quantum computing is therefore of great importance. On the theory side, it has already transformed our notions of which problems are tractable and which are not. On the practical side, the construction of a quantum computer would render many popular techniques for encryption obsolete.

We derive several results in quantum computation. Quantum algorithms are normally formulated in an abstract way that ignores practical details such as where different parts of the computation are located physically. Using naive methods for translating these algorithms into practical quantum computing technologies increases the time complexity by a linear factor, while previous work reduced this factor to a square root. We further reduce this

factor to a constant at the cost of requiring additional space. This retroactively justifies an important assumption made in many quantum algorithms.

Another interesting problem is to determine the query complexity of extracting different types of information from physical processes. In the case of deterministic processes, it is well known that quantum algorithms cannot outperform deterministic algorithms by more than an exponential factor. We show — somewhat surprisingly — that when the process is allowed to be randomized, there are problems that have a constant quantum query complexity but cannot be solved classically no matter how many queries are made. In fact, we show how to construct such an *infinity-vs-one* separation from any weaker separation between the classical and quantum query complexities.

In an *isomorphism problem*, we seek to determine if two algebraic or combinatorial objects have the same structure. One of the most well-known of these is the graph isomorphism problem, which is interesting since it is suspected to be of complexity intermediate between P and the NP-complete problems.

We transition from quantum computation to isomorphism testing by considering the tree isomorphism problem. While linear-time classical algorithms are known for this problem, we show that a promising framework for developing efficient quantum algorithms for graph isomorphism, known as the *state preparation approach*, can also solve tree isomorphism. While this result might seem modest, it is important to know that the state preparation approach works for trees, since otherwise there would be little hope of using it to obtain efficient algorithms for more interesting classes of graphs. We also derive a powerful primitive along the way that is of independent interest.

Next, we study the *group isomorphism problem*, which is a special case of graph isomorphism. Group isomorphism is suspected to be significantly easier than graph isomorphism, but still has unknown complexity. While group isomorphism is already of interest since it is a fundamental computational question about groups, searching for faster algorithms for

group isomorphism is therefore also one way of approaching the graph isomorphism problem. We use collision detection methods to give a classical algorithm that obtains a square-root speedup over the best algorithm previously known for the general group isomorphism problem. For the solvable groups (which are conjectured to be difficult and contain almost all groups), we combine our collision detection techniques with group-theoretic methods to obtain a classical fourth-root speedup over the best algorithm known previously. Prior to this work, it was a longstanding open problem to obtain any improvement for either of these classes of groups. Finally, we give a general framework that yields speedups for many isomorphism problems. In particular, it yields the square-root speedup for testing isomorphism of general groups mentioned above. All of our group isomorphism-testing algorithms also have quantum variants that are slightly faster than their classical counterparts.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

First and foremost, I am especially grateful to my advisors Paul Beame and Aram Harrow for all of the valuable advice, encouragement and feedback that they have given me over the years that I have known them. Without their help, guidance and patience, this thesis would not have been possible.

I would also like to thank the other members of my committee (James Lee, Larry Ruzzo and William Stein), for taking the time to serve on my committee and also for their helpful comments and feedback.

Two of the chapters in this thesis are collaborations with others. Chapter 6 is joint work with my advisor Aram Harrow and Chapter 10 is joint work with Fabian Wagner.

Finally, the work described in this thesis benifited from useful comments and feedback from other researchers. Laci Babai provided extensive comments on Chapters 10 – 12 that greatly improved the presentation. I am also greatful to Dave Bacon, Joshua Grochow, Richard Lipton, Paul Pham and the many anonymous referees who have reviewed my work over the years for their comments and feedback.

DEDICATION

This thesis is dedicated to my parents, who have always been there for me when I needed them.

Chapter 1

INTRODUCTION

At the same time that Turing and Church were developing the foundations of classical computer science, physicists were formulating the theory of *quantum mechanics*. Quantum mechanics is fundamentally different from classical physics as a quantum system can be simultaneously in a superposition of many classical states. This leads to strange phenomena that are unique to quantum mechanics such as entanglement (which allows stronger correlations than are possible classically) and destructive interference (which occurs when different classical states in the superposition interact to cancel each other out). These effects are potentially of great use computationally; however, classical computers are unable to take advantage of them since they cannot store or manipulate quantum states.

Quantum computers are devices that are analogous to classical computers, but which use quantum states instead of classical bit strings to store information. In a physical sense, quantum computers are more natural than classical computers since there is no reason to prohibit operations as long as they are possible physically. Since a quantum superposition can contain exponentially many classical states, stimulating an arbitrary quantum computation classically seems to require exponential time. Quantum computers therefore appear to violate the strong Church-Turing thesis — which states that Turing machines can efficiently simulate any physically realistic model of computation — and are worthy of study for this reason alone. While the task of simulating an arbitrary quantum computation is not a classical problem, early papers [39, 115, 116, 38] showed that there are quantum algorithms that can solve certain classical problems exponentially faster than the best algorithms known classically. The utility of quantum computers is therefore not restricted quantum problems but applies to classical problems as well. The study of quantum computation is therefore strongly motivated by its ability to transform our notions of which problems are tractable and which are not.

One of the most impressive quantum algorithms is due to Shor [115] who shows that integer factoring and computing discrete logarithms (a problem which arises in cryptography) can be done in polynomial time on a quantum computer. The best classical algorithms known for these problems require $2^{n^{\Omega(1)}}$ time, so this is an exponential speedup. Since widely used encryption schemes such as RSA [101] and elliptic curve cryptography rely respectively on the assumptions that factoring and the discrete logarithm problem are difficult, many modern encryption systems will be vulnerable once a sufficiently large quantum computer is built [31]. Quantum computing therefore is not only of great theoretical importance, but also has profound real-world implications.

The underlying ideas behind Shor's result are in fact more general than either factoring or computing discrete logarithms. Both of these problems can be viewed as special cases of the *hidden subgroup problem*. We will define this problem formally later; however, for now it is enough to know that it is the basis of most quantum algorithms that exponentially outperform their classical counterparts.

Another important quantum algorithm is Grover's algorithm [52, 25] which is capable of performing brute force search over a set of size N in only $O(\sqrt{N})$ time. Classically, $\Theta(N)$ time is required even for randomized algorithms. This speedup applies even to problems that appear to be very difficult, such as the NP-complete problems. While this is a square-root rather than an exponential speedup, it is nonetheless surprising and impressive due to the wide range of search problems to which it can be applied.

Despite the promise of large speedups provided by quantum computers, early on, researchers were concerned that it might not be possible to build a quantum computer even in theory. Since quantum computers store and manipulate quantum information, they are also vulnerable to noise from the environment, which can disrupt computations. Noise is much less of a problem for classical computers, because each classical bit is stored using a very large number of particles, which causes any errors that occur to be automatically corrected.

On the other hand, quantum computers typically store information in individual atoms or subatomic particles[1], and are therefore much more vulnerable to errors. Fortunately, the discovery of quantum-error correction and the Threshold Theorem [1] show that any quantum computation can be made proof against errors with low computational overhead. These discoveries have transformed the quest to build a general-purpose quantum computer from something that, *a priori*, may not have even been theoretically possible, into an engineering challenge.

Quantum-error correction and the Threshold Theorem provide the theoretical justification for the first assumption of the *abstract model* of quantum computation, which states that we can assume that all quantum computations are performed exactly, without any errors. The abstract model of quantum computation is convenient since it allows one to ignore low-level implementation details and instead focus on algorithm design; for this reason, most quantum algorithms are formulated in this model.

However, the abstract model also has a second assumption that is more problematic. In order to explain this, we first need to discuss the basic building blocks of quantum algorithms. There are two elementary types of operations: single-particle operations and interactions between pairs of particles. In a physical implementation of a quantum computer, these particles must be arranged in space and interactions can only be performed between particles that are spatially close. If we wish to perform an interaction between a distant pair of particles, we must move them close together first.

The second assumption is that quantum operations may be performed on arbitrary pairs of particles in the quantum computer. This is physically unrealistic, since — as mentioned above — distant pairs of particles cannot interact directly. Moreover, naive methods for moving distant pairs of particles close to each other are inefficient and add significant computational overhead. This presents a problem since we wish to maintain efficiency when mapping abstract algorithms into practical architectures.

[1]In some quantum computing technologies, information is stored in other ways. However, for the purposes of this discussion, we shall assume that particles are used.

In this thesis, we shall consider one way of arranging particles in space that accurately models many quantum computing technologies. One of our main results in this model justifies the second assumption of the abstract model of quantum computation by showing that interactions between distant particles can be simulated with extremely low computational overhead. In fact, by using our construction, arbitrary interactions between distant pairs of particles can be handled while only increasing the runtime of an abstract algorithm by a constant factor. Coupled with the Threshold Theorem, this result shows that *all* of the assumptions of the abstract model can be removed without significantly reducing efficiency. This is of great theoretical importance since it retroactively justifies the model of quantum computation in which most algorithms are formulated. On the practical side, it gives us a concrete way of mapping theoretical algorithms into practical quantum computing architectures.

In both classical and quantum algorithms, it is often useful to have an abstraction that models the case where we have a subroutine that we are allowed to run, but cannot inspect its code. This could model the situation in which we do not understand the code for the subroutine or where the subroutine is stored on a remote server to which we are allowed to send queries but do not have direct access. The abstraction that models these situations is called an *oracle* or *black box*. Many quantum algorithms (including the aforementioned results of Shor [115] and Grover [52, 25]) are formulated in terms of oracles. An advantage of this is that the results are independent of the internal workings of the black box and thus hold for any implementation of the oracle.

Typically, oracles implement deterministic functions. In this case, a quantum algorithm for an oracle problem can be simulated classically with exponential overhead. Thus, any deterministic oracle problem which can be solved quantumly can also be solved classically (albeit much more slowly). In this thesis, we also consider oracles that are be allowed to behave randomly. From a quantum perspective, such oracles are natural since they can model random physical processes. For these randomized oracles, we show that there are problems that cannot be solved with any number of classical queries but can be solved quantumly using a single query. In other words, there are problems in which any number of classical queries yield no information but a single quantum query yields enough information to solve the problem. This demonstrates an even larger separation between classical and quantum computers than the aforementioned exponential speedups.

The problems of factoring integers and computing discrete logarithm[2] solved by Shor's algorithm are examples of problems that are suspected to be of complexity intermediate between P and NP. Another problem that is suspected to be of intermediate complexity is the *graph isomorphism problem*. There is complexity-theoretic evidence that graph isomorphism is not NP-complete, as this would contradict a widely-believed complexity-theoretic conjecture [9, 24, 48, 47]. In this problem, we are given two graphs and must determine if the first graph can be redrawn with different labels for its nodes such that it is identical to the second graph. In this case, the graphs are said to be *isomorphic*.

In addition to being of great theoretical interest, the graph isomorphism problem also has many important practical applications. A few of these are molecular databases [65], circuit verification [129] and generating instruction sets [35]. For this reason, much effort has been put into devising practical algorithms for graph isomorphism. Unfortunately, progress on worst-case algorithms has been much more limited: the best theoretical algorithm known for general graphs [18, 16] was devised in 1983 and has not been significantly improved since. Later in this thesis, we will show a small improvement to this algorithm via collision detection arguments.

Due to this lack of progress on faster worst-case classical algorithms for graph isomorphism, there has been strong interest in developing faster quantum algorithms for graph isomorphism. Since graph isomorphism can be cast as a special case of the hidden subgroup problem — which we will define later and is the basis of most quantum speedups [39, 115, 116, 38] — many in the quantum algorithms community were initially optimistic that an

[2]Technically, these are not decision problems. However, there are variants of both that are decision problems. It is these variants that seem to have complexity between P and NP (which are classes of decision problems).

efficient quantum algorithm for graph isomorphism would be found. Unfortunately, the instance of the hidden subgroup problem that arises in graph isomorphism seems much more difficult than those that arise in other contexts and, so far, efforts to solve it have been fruitless.

Because of the apparent difficulty of the general case of graph isomorphism, we focus on two special cases in this work: tree isomorphism and group isomorphism. While a linear-time classical algorithm [4] is known for tree isomorphism, we use tree isomorphism as a step towards quantum algorithms for graph isomorphism based on the *state preparation* approach [3] to graph isomorphism. Given a graph, the goal in this approach is to produce a *complete-invariant quantum state* that encodes the isomorphism class of the graph. Given the complete invariant states for two graphs, we require that there exists an efficient quantum algorithm that tests if the graphs are isomorphic. Fortunately, for a natural definition of a complete invariant state that corresponds to all permutations of the graph, such an algorithm does indeed exist [28]. The challenge is in preparing these complete-invariant states for interesting classes of graphs.

Our contribution towards this problem is to show that symmetrized complete-invariant states can be prepared for any tree. While tree isomorphism is not particularly interesting on its own, it is important to know that the state preparation approach at least works for the class of trees, since if it did not there would be no hope that it would work for more complicated classes of graphs. There are also some classes of graphs that generalize trees but have isomorphism problems that are not known to be solvable in polynomial time classically. One such class of graphs is the *cone graphs*, which are trees that are allowed to have *cross edges* between nodes at the same distance from the root.

A more interesting and difficult special case of graph isomorphism is the *group isomorphism problem*. *Groups* are mathematical abstractions that generalize operations such as addition and multiplication. However, abstract groups can be much more general. For instance, in the case of multiplication and addition, $x + y = y + x$ and $x \cdot y = y \cdot x$, but for a general group operation $*$, it is not necessarily that case that $x * y = y * x$. An example of such a group is the set of all permutations of $[n]$ with function composition as the operation.

One way of specifying a finite group is by a *multiplication table*, which stores the value $x * y$ for each pair of elements x and y in the group. Two groups are *isomorphic* if the elements of the first can be relabeled so that the first group operation becomes identical to the second. While for tree isomorphism, we only considered quantum algorithms, for group isomorphism we shall utilize both classical and quantum techniques.

In the group isomorphism problem, we are given two finite groups specified as multiplication tables and must decide if they are isomorphic. Group isomorphism essentially asks if two groups are the same modulo relabeling their elements. Thus, it is a fundamental problem in computational group theory and is interesting for this reason alone. However, there are at least two other reasons why group isomorphism is worthy of study.

First, in order to devise an efficient algorithm for a class of groups, is often necessary to obtain structural insights into the class of groups in question. This has already happened in a number of papers on the group isomorphism for certain classes of groups [70, 12, 19, 13, 51].

Second, group isomorphism is a special case of graph isomorphism that is still not known to be solvable in polynomial time. Moreover, there is good reason to believe that group isomorphism may be easier than graph isomorphism. Since the 1970's, group isomorphism has been known to be decidable in $n^{\log n + O(1)}$ time using the *generator-enumeration algorithm* [44, 74, 84]; this is much faster than the $n^{O(\sqrt{n/\log n})}$ runtime of the best worst-case algorithm [18, 16] known for graphs. There is also complexity-theoretic evidence [29, 91] that graph isomorphism cannot be reduced to group isomorphism. Thus, group isomorphism is a nontrivial special case of graph isomorphism that is probably considerably easier than the general graph isomorphism problem. Since progress on the general case of graph isomorphism is stalled, studying the special case of group isomorphism seems like a useful approach.

Since the discovery of the aforementioned generator-enumeration algorithm for general groups, there has been progress on group isomorphism for interesting subclasses of groups [74, 113, 125, 63, 70, 94, 12, 34, 19, 13, 71, 51]. However, until the work which we describe later in this thesis, even improving the constant 1 in front of the $\log n$ in the exponent of the

$n^{1\cdot\log n+O(1)}$ runtime of generator enumeration for general groups was an open problem for several decades [72, 73].

In fact, until the work described in this thesis, improving the constant in the exponent of the generator-enumeration algorithm was similarly open even for certain difficult subclasses of groups. One such class of groups is the *p-groups* (which we shall define later). Researchers conjecture [12, 34, 19] that the p-groups contain the hard case of the group isomorphism problem, since there are p-groups that have many complicated isomorphisms that are not well understood. It has also been shown empirically [22] that almost all groups are p-groups; this provides further evidence that testing isomorphism of p-groups is as difficult as testing isomorphism of general groups. One of the main results of this thesis is a deterministic classical algorithm that solves p-group isomorphism in $n^{(1/2)\log n+o(1)}$ time. This improves the constant in the exponent of the generator-enumeration algorithm from 1 to 1/2 and thereby solves the open problem mentioned in the last paragraph in the case of p-groups.

An even more general class of groups is the *solvable groups* which were developed by Galois for the purpose of studying the solvability of quintic polynomials. The solvable groups contain the p-groups as well as many other groups, such as the groups that contain an odd number of elements [43]. Thus, the solvable groups are one step closer to the general case of group isomorphism. Another important result of this work is the extension of the $n^{(1/2)\log n+o(1)}$ time upper bound for p-groups to the class of solvable groups. We accomplish this by using additional group-theoretic tricks which complicate the algorithm significantly but allow us to obtain the same value in the exponent up to slightly worse lower order terms.

The final contribution of this work is a general collision-detection framework for obtaining square-root speedups for isomorphism testing problems. As a result, we are able to show an algorithm with a runtime of $n^{(1/2)\log n+o(1)}$ for the class of general groups, which resolves the open problem of improving on generation-enumeration in the most general setting. By combining this framework with our algorithms for p- and solvable-group isomorphism, we are able to further reduce the runtime for these classes of groups to $n^{(1/4)\log n+o(1)}$. This constitutes a fourth-root speedup over the original generator-enumeration algorithm.

Our collision detection framework can also be used to obtain faster quantum algorithms for isomorphism problems. In the typical case, these are cube root speedups over the original algorithm. This yields an $n^{(1/3)\log n+O(1)}$ time quantum algorithm for general groups and $n^{(1/6)\log n+o(1)}$ time algorithms for the classes of p- and solvable-groups.

Chapter 2
OVERVIEW OF RESULTS

In this chapter, we state the main results of this thesis. Some of the theorem statements are informal and are less precise than those presented later in order to make this chapter more readable. We start with our quantum computing results in Section 2.1 and discuss our results on classical isomorphism testing in Section 2.2. We also mention quantum variants of these algorithms. Lastly, we give a roadmap for the rest of the thesis in Section 2.3.

2.1 Quantum computing

As mentioned in the introduction, quantum computation is a model of computation that exploits quantum mechanics in order to solve problems more quickly. It is unique among all known physically plausible models of computation due to its apparent violation of the strong Church-Turing thesis and is strongly motivated by efficient algorithms for problems that are conjectured to be intractable on classical computers.

In the introduction, we mentioned that most quantum algorithms are formulated in the abstract model where we assume that there are no errors. The errors that we discussed are those that result from noise caused by undesirable interactions with the environment. However, errors can also result when the *basic operations* from which all other quantum operations are built are performed imprecisely. Such errors occur due to defects in the underlying quantum device or errors in the classical circuit the controls the device. Fortunately, the Threshold Theorem can account for gate errors as well. As long as the basic operations can be implemented with error less than some universal constant, the Threshold Theorem [1] implies that any quantum computation can be protected against noise while increasing the overhead by only a polylogarithmic factor. While building gates accurate enough for the threshold theorem is a significant challenge for experimental physicists, there are no fundamental theoretical obstacles and it is likely only a matter of time before this is achieved [1].

In this work, we shall take advantage of the threshold theorem by ignoring both gate errors and noise from the environment. A significant advantage of this approach is that any results obtained are independent of the particular technology that is used to implement quantum computers. Thus, they are likely to be relevant regardless of whatever quantum computing technology ultimately proves most successful. This assumption also simplifies algorithm design considerably.

In order to discuss the second assumption of the abstract model in more detail, we need to introduce the two main primitives of quantum computation: *qubits* and basic operations. Qubits are the basic unit of quantum information and are analogous to classical bits. Previously, we discussed quantum information terms of particles, which are one way of implementing qubits. However, qubits can also be implemented in other ways. For this reason, we shall use the more standard term qubit from now on.

The other class of primitives from which quantum computers are built are basic operations (or gates). Just as classical circuits are built from basic classical gates such as AND, OR and NOT, *quantum circuits* are built from an analogous small set of one- and two-qubit quantum gates. Most of these basic operations are *reversible*, which means that there is another inverse operation that can be applied in order to restore the system to the state that it had before the first operation was applied. An example of a basic operation that is not reversible is a measurement. Unlike classical measurements, which do not change the state of the classical bit, quantum measurements force the quantum superposition of classical states into a single classical state and therefore affect the state of the system. This has many important implications to quantum computation.

The second assumption made in the abstract model of quantum computation is that two-qubit gates can be performed on arbitrary pairs of qubits. As mentioned previously,

[1] Aram Harrow (personal communication).

in a physical implementation of a quantum computer, the qubits have positions in space and therefore operations can only be directly performed between adjacent pairs of qubits. It is still possible to perform two-qubit operations between distant pairs of qubits, but the computational overhead is greater. Since most quantum algorithms are described in the abstract model, it is important to find a way to implement them efficiently on realistic quantum computers.

2.1.1 2D *quantum circuits*

Van Meter and Itoh [124] (cf. [32]) proposed a model that accounts for the spatial layout in many technologies by arranging the qubits on a k-dimensional grid. Two-qubit operations may be performed on neighboring pairs of qubits and single-qubit gates may be performed on any qubit. Operations are also allowed to be performed in parallel so long as they are on disjoint sets of qubits. This model accurately represents many quantum computing technologies such as ion-trap quantum computers, where the qubits are often arranged on a grid.

Quantum algorithms typically also assume that there is a *classical controller* that decides which operations to perform at each step based on the input and the measurements performed so far. The classical controller is allowed to perform arbitrary randomized polynomial time computations in order to accomplish this. One can also consider the *non-adaptive* case where a classical controller is not used. This means that the operations performed at the j^{th} timestep depend only on the input and j. In Chapter 5, we consider four models of quantum circuits: the abstract model with a classical controller, the abstract model without a classical controller, the k-dimensional grid model with a classical controller and the k-dimensional grid model without a classical controller.

The number of steps used in a quantum circuit is called the *depth*. The total number of basic operations is the *size* and the number of qubits is the *width*. In Chapter 5, we show that any quantum operation that can be implemented in the abstract model using a classical controller can be simulated on the 2D grid while increasing the depth by a constant factor and squaring the width.

Theorem 2.1.1. *Suppose that C is an abstract quantum circuit with a classical controller that has depth d, size s and width n. Then C can be simulated in $O(d)$ depth, $O(sn)$ size and n^2 width in the 2D grid using a classical controller.*

Since the depth corresponds to the time required to perform a computation and is therefore arguably the most important computational resource, this result can be thought of as justifying the second assumption of the abstract model.

The proof of this result is based on a lemma that shows that, on a 2D square grid, a column of qubits can be permuted arbitrarily in constant depth using quantum teleportation.

Chapter 5 also considers quantum circuits on a 2D grid without a classical controller. In this case, we show that an operation with n controls can be implemented in $\Theta(\sqrt[k]{n})$ depth in a kD grid without using a classical controller. We also prove a matching lower bound.

Theorem 2.1.2. *The depth required for controlled-U operations with n controls and fanouts with n targets in a kD grid without using a classical controller is $\Theta(\sqrt[k]{n})$. Moreover, this depth can be achieved with size $\Theta(n)$ and width $\Theta(n)$.*

2.1.2 Infinity-vs-one separations and uselessness

While Chapter 5 is very concrete and practical, in Chapter 6, we explore the more theoretical domain of *oracles*. As mentioned in the introduction, an oracle is a black-box that computes an unknown function. In an *oracle problem*, we are given an oracle that computes an unknown function and must decide if the function has some property by *querying the oracle*. A query consists of applying the oracle to a state of our choosing. However, queries are all that is allowed: we cannot inspect the inner workings of the oracle.

While it may seem artificial at first glance, oracles can be justified in several ways. One natural formulation is that the oracle is represented by an external server that computes a function that we are allowed to query. However, since we do not have access to the server itself, we cannot look inside the black box. Another more surprising approach is for the oracle to be specified explicitly by its source code. This is justified by Rice's theorem [100] which shows that it is impossible to decide anything interesting about what a Turing machine computes by inspecting its source code.

Typically, oracles act according to a deterministic function that is applied to the input. Therefore, a natural extension is to oracles whose behavior can depend on some internal random process. We call this an *oracle with internal randomness.* The study of such oracles is well-motivated since we can think of many randomized physical process as oracles with internal randomness.

Our interest here is in *query complexity*: the minimum number of calls to the oracle required to solve the problem with unlimited computational resources. Moreover, we only require that the algorithm obtains the correct result with some arbitrarily small advantage over guessing randomly.

As previously mentioned, there are a number of deterministic oracle problems [39, 115, 116, 38] that can be solved with quantum algorithms using a polynomial number of queries but require exponentially many queries classically. Such an *exponential separation* is the best we can hope for in the case of deterministic oracles, since a single quantum query can be simulated by an exponential number of classical queries.

In the first part of Chapter 6, we show that far stronger separations are possible for oracles with internal randomness. Namely, there are problems involving oracles with internal randomness that can be solved using a single quantum query but cannot be solved classically no matter how many queries are made. We now introduce several problems for which such *infinity-vs-one* separations exist.

A permutation is called an *involution* if composing it with itself yields the identity permutation. Another type of permutation is a *cycle*, so one can consider the problem of distinguishing an oracle that applies a random involution from one that applies a random cycle of length at least three. We call this the problem of *distinguishing involutions from cycles* and show that an infinity-vs-one exists for this problem.

In *Simon's problem*, we are given a deterministic oracle that allows us to query a binary function $f : \{0,1\}^n \to \{0,1\}$ where there exists some unknown a such that $f(x+a) = f(x)$ for all $x \in \{0,1\}^n$ where addition is performed coordinate-wise and modulo 2. Our goal is to find a. We show that one can modify Simon's problem by adding randomness to the oracle to obtain a second infinity-vs-one separation.

The *hidden linear structure problem* [38] is an oracle problem that can be solved exactly using a single quantum query but requires an exponential number of queries classically. By adding randomness to this oracle, we obtain yet another infinity-vs-one separation.

The basic reason behind these results is that when the oracle has internal randomness, each query is effectively on a different oracle, since the output of the internal random process can be different for each oracle call. This allows one to construct problems where a single quantum query can extract information from the oracle but classical queries yield random noise.

In the second part of Chapter 6, we study when k queries to an oracle yield information about the solution to the problem. We say that k queries are *useless* if there is no way to query the oracle k times that yields any information about the problem. One can talk about either *quantum uselessness* or *classical uselessness*[2], which are the concept of uselessness applied to classical and quantum queries respectively. We show that k quantum queries are useless if and only if $2k$ classical queries are useless, with the caveat that the classical queries come in pairs that share the same internal randomness. This generalizes a result of [82].

Theorem 2.1.3. *For any oracle problem, k quantum queries are useless if and only if $2k$ classical queries are useless where the classical queries come in pairs that share the same random seed.*

2.1.3 A quantum algorithm for tree isomorphism

In Chapter 7, we move back from considering query complexity to time complexity. This subsection is the last on a result that is primarily quantum. Since it is also an algorithm for

[2]The concept we refer to as classical uselessness here is called weak classical uselessness in Chapter 6 to distinguish it from the other types of uselessness introduced in that chapter.

isomorphism testing, it provides a useful transition to Section 2.2, which is primarily about classical algorithms for isomorphism testing. We start by introducing the general notion of an *isomorphism problem.*

We say two algebraic or combinatorial objects are *isomorphic* if their elements can be relabeled so that they have the same structure. For example, two graphs are isomorphic if the nodes of the first graph can be relabeled so that it has the same edges as the second graph.

Isomorphism problems are closely related to *group theory* which can be used to describe all isomorphisms between two objects. As mentioned in the introduction, a *group* is a mathematical abstraction the generalizes operations such as addition, multiplication of nonzero numbers and composition of permutations.

One of the biggest open problems in classical theoretical computer science is to find an efficient algorithm for the graph isomorphism problem. While efficient practical algorithms are available [80, 59, 37, 62, 81] and there is complexity-theoretic evidence [9, 24, 48, 47] that graph isomorphism is not NP-complete, to date the best worst-case classical algorithm known [16, 18, 76] for this problem runs in $2^{O(\sqrt{n \log n})}$ time and has not been improved for over thirty years. A major open problem in quantum algorithms has therefore been to find a faster quantum algorithm for graph isomorphism.

As mentioned earlier in this chapter, most quantum algorithms (cf. [39, 115, 116, 38]) that provide exponential speedups over their classical counterparts are based on a group-theoretic problem called the hidden subgroup problem over Abelian groups [64] (cf. [31]). Graph isomorphism has a natural reduction to the hidden subgroup problem over the symmetric group, so there was some reason to hope that the techniques used in other quantum algorithms might yield results for graph isomorphism. Unfortunately, developing quantum algorithms for the hidden subgroup problem over the symmetric group has proved to be quite difficult and a series of negative results [54, 87, 88] have made it seem increasingly unlikely that the hidden subgroup problem over the symmetric group will yield faster algorithms for solving graph isomorphism.

The *state preparation approach* to graph isomorphism [3] is based on preparing a quantum superposition that represents the isomorphism class of the graph. Let us assume without loss of generality that the vertices of the graph X are labelled by $[n]$.

Since quantum states are vectors in a complex Hilbert space, any labeling of a graph can be represented by a state. For example, we can use the $0-1$ vector that corresponds to its adjacency matrix. This allows us to define a state that represents the isomorphism class of the graph rather than a particular labeling. In keeping with standard quantum notation[3], we denote the state that represents the isomorphism class of X by $|X\rangle$ rather than the more conventional notation $\mathbf{X}$. We can then define the quantum state $|X\rangle$ to be the sum of the states that correspond to the graphs obtained by relabelling the vertices of X in all possible ways. Since by definition the graphs X and Y isomorphic if and only if there is a permutation that transforms X into Y, the set of all permutations of X is equal to the set of all permutations of Y if $X \cong Y$. On the other hand, if $X \ncong Y$, then the set of all permutations of X and the set of all permutations of Y are disjoint. This implies that the states $|X\rangle$ and $|Y\rangle$ for two graphs X and Y are are equal if X and Y are isomorphic and are orthogonal otherwise. Since the swap test [28] (which we cover in Section 4.5) provides a means of distinguishing these two cases, the ability to prepare $|X\rangle$ suffices to solve graph isomorphism.

In Chapter 7, we take a first step towards this goal by showing how to prepare $|X\rangle$ when X is a rooted tree. While it is well-known that tree isomorphism can be solved in linear time classically [4], it is important to know that the state preparation approach at least works on trees since, if it did not, it would be unlikely to work on more complicated graphs. There is also some hope that such a quantum algorithm for tree isomorphism could be generalized to more difficult classes of graphs that generalize trees, such as cone graphs. The main result of Chapter 7 is an algorithm for preparing an invariant state $|T\rangle$ for a rooted tree T. Shor observed that isomorphism testing algorithms such as the linear time algorithm for

[3]The $|X\rangle$ notation has certain advantages over the more conventional $\mathbf{X}$ notation that will become apparent in Chapter 4.

tree isomorphism [4] can be transformed into procedures for efficiently preparing complete invariant states. This yields an algorithm for computing complete invariant states for trees. However, all of the isomorphism problems that arise are handled by using the classical algorithm as a subroutine, so it seems unlikely that the resulting algorithm would lead to techniques that would be useful in efficient quantum algorithms for classes of graphs that are difficult classically.

Theorem 2.1.4. *Let T be a rooted tree. Then we can prepare a state $|T\rangle$ in polynomial time such that*

(a) if T' is a tree isomorphic to T, then $|T\rangle = |T'\rangle$ and

(b) if T' is not isomorphic to T, then $|T\rangle$ and $|T'\rangle$ are orthogonal.

Along the way, we also prove a useful lemma that allows one to permute a set of orthogonal states by all permutations in a given permutation group.

Lemma 2.1.5. *Let G be a permutation group of degree k and let $U_1, \ldots, U_k$ be unitary matrices on n qubits that can be implemented with a polynomial number of basic operations such that $\langle 0| U_i^\dagger U_j |0\rangle = 0$ for $i \neq j$ where $\langle 0|$ and $U_i^\dagger$ are the conjugate transposes of $|0\rangle$ and U_i. Then the state*

$$\frac{1}{\sqrt{|G|}} \sum_{\pi \in G} \bigotimes_{i=1}^{k} U_{\pi^{-1}(i)} |0\rangle \tag{2.1}$$

can be prepared in time polynomial in k and n.

The symbol $\otimes$ is called a *tensor product* and is the quantum analog of concatenating classical bit strings. Thus, (2.1) is the sum of all vectors that can be obtained by permuting the states $U_{\pi^{-1}(i)} |0\rangle$. The proof of Lemma 2.1.5 involves *strong generating sets*, whose existence is a result from permutation group theory that is central to many permutation group algorithms.

Modulo a number of technical details, the proof of Theorem 2.1.4 works by recursively preparing the state $|T_i\rangle$ for each subtree T_i rooted at a child of the root node of T. It then applies a generalization of Lemma 2.1.5 which allows the states $U_i |0\rangle$ to have different numbers of qubits in order to rearrange these subtrees in all possible ways.

2.2 Isomorphism testing

In the second half of this thesis, we move on to classical algorithms for isomorphism testing. Our algorithms in this part are primarily classical; however, all of them have quantum variants as well. The main problem we consider is the *group isomorphism problem* — a special case of graph isomorphism that is also of independent interest. In this problem, we are given two finite groups G and H as multiplications tables that specify the product of every pair of group elements under the group operation.

Group isomorphism is potentially much easier than graph isomorphism since the classic generator-enumeration algorithm [44, 74, 84] solves group isomorphism in $n^{\log_p n+O(1)}$ time where p is the smallest prime that divides the order of the group, whereas the best algorithm known for graph isomorphism is much slower. While there are a variety of faster algorithms [74, 113, 125, 63, 70, 94, 12, 34, 19, 13, 51] for restricted special cases of the group isomorphism problem, until recently, the generator-enumeration algorithm was still the fastest algorithm known for general groups over three decades after it was originally introduced [72, 73].

This part of the thesis is arranged as follows. We introduce the *color automorphism problem* in Chapter 8. Color automorphism is one of the two main ingredients in the best worst-case algorithm known for graph isomorphism [18, 16] and is also used in later chapters. We review some of the algorithms for restricted special cases of the group isomorphism problem in Chapter 9. In Chapters 10 – 12, we show the first improvements over the generator-enumeration algorithm for general groups as well as larger improvements for the hard special cases of p-groups and solvable groups.

2.2.1 p-group isomorphism

The hard case of group isomorphism is conjectured [12, 34, 19] to be the *class* 2 *nilpotent groups*. These groups are "almost Abelian" in the sense that the quotient group $G/Z(G)$ is Abelian and $Z(G) = \{x \in G \mid xg = gx \text{ for all } g \in G\}$ is Abelian by definition. However, these Abelian factors cause the number of candidate isomorphisms to be large, while the non-Abelian interactions between them defy methods for Abelian groups [74, 113, 125, 63] based on the Structure Theorem for Finitely Generated Abelian Groups (see Section 3.4).

A *p-group* is a group whose order is a power of p where p is prime. Testing isomorphism of class 2 nilpotent groups reduces to p-group isomorphism since every nilpotent group is a direct product of p_i-groups where the p_i's are distinct primes. Therefore, we can consider p-groups instead of class 2 nilpotent groups. The main result of Chapter 10 builds on work by Wagner [126] to show an improvement over generator-enumeration for the class of p-groups.

Theorem 2.2.1. *p-group isomorphism is decidable in* $n^{(1/2)\log_p n + o(\log n)}$ *time.*

In fact, a slightly sharper bound is possible, as we will see in Chapter 10.

The proof of this result has two main steps. Both steps are closely related to *composition series*, which are sequences of subgroups of a group with certain properties. First, we show that there are most $n^{(1/2)\log_p n + O(1)}$ composition series[4] for a group whose smallest prime divisor is p. Using this, we derive an $n^{(1/2)\log_p n + O(1)}$ time Turing reduction to testing isomorphism of composition series. The second part of the proof involves constructing a graph of degree $p + O(1)$ that represents the isomorphism class of a composition series. Since testing isomorphism of graphs of degree bounded by d is in $n^{O(d)}$ time [18], this implies an $n^{(1/2)\log_p n + O(p)}$ algorithm for p-group isomorphism. The bound in Theorem 2.2.1 then follows by using this algorithm when $p \le o(\log n)$ and the generator-enumeration algorithm when p is larger.

[4] A more complicated subclass of composition series was used originally to obtain the same result. However, Laci Babai pointed out that one can obtain a simpler proof by considering the class of all composition series.

2.2.2 Solvable-group isomorphism

The focus of Chapter 11 is to generalize Theorem 2.2.1 to the class of solvable groups. In order to accomplish this, we need to describe how the graph for composition series $G_0 = 1 \lhd \cdots \lhd G_m = G$ is constructed.

A *coset* of the group G by a subgroup G_i is a set of the form $xG_i = \{xg \mid g \in G_i\}$ where $x \in G$. One can show that the set G/G_i of all cosets of G with respect to G_i forms a partition of G and so the cosets yG_i that are contained in a coset xG_{i+i} partition xG_{i+1}. The idea is to construct a tree where the i^{th} level corresponds to the cosets G/G_i. The root node therefore corresponds to the group G and its children are the cosets of the form xG_{m-1}. In general, the children of a coset xG_{i+1} are the cosets yG_i such that $yG_i \subseteq xG_{i+1}$. Thus, the children of each coset are the cosets by the subgroup at the next level in the series that partition it. Since G was assumed to be a p-group, this tree has degree $p + 1$. The final step involves attaching multiplication gadgets to this tree in a careful way that increases the degree only by a constant.

Before generalizing this construction to solvable groups, we review a few relevant facts about composition series. In a composition series, the quotients G_{i+1}/G_i of adjacent subgroups are themselves groups and are called the *composition factors* of G. (The set of composition factors depends only on G and not on the particular composition series chosen by the Jordan-Hölder Theorem (see Section 3.5)).

The above construction of the tree for a composition series actually works even when the group is not a p-group. However, in this case, the degree of the graph will correspond to the order of the largest composition factor, which may be large. Wagner [126] showed a trick that allows large composition factors to be eliminated from this tree at the cost of multiplying the runtime by a factor of $n^{o(\log n)}$. This allows the degree of the tree to be reduced to $o(\log n)$ assuming that all composition factors occur at the top of the composition series. Unfortunately, this is not always the case for solvable groups.

We get around this problem using special structural results available for solvable groups.

A theorem of Hall [53] shows that every solvable group can be written as a product of groups that pairwise commute. In contrast to the case of nilpotent groups, this product is not a direct product, so one cannot trivially reduce to the case of p-groups. However, Hall's result allows us to create a generalization of the composition series that consists of the subgroup of a solvable-group G that contains all the large composition factors, as well as a composition series for the subgroup that consists of the small composition factors[5]. Though the construction becomes considerably more technical, this idea allows us to construct a low-degree graph that represents the isomorphism class of such a generalized composition series of a solvable group. This yields the main result of Chapter 11.

Theorem 2.2.2. *Solvable-group isomorphism is decidable in $n^{(1/2)\log_p n + o(\log n)}$ deterministic time where p is the smallest prime dividing the order of the group.*

2.2.3 Bidirectional collision detection

While our results in Chapters 10 and 11 already improve on the best algorithms known for p- and solvable groups, we go even further in Chapter 12 and obtain a speedup for the case of general group isomorphism as well. The underlying method is a generic bidirectional collision detection lemma that is applicable to many isomorphism problems. As a result, we also obtain further speedups for p- and solvable groups.

Our lemma works for any problem for which one can compute a "partial canonical form" for the objects on which we wish to test isomorphism. Such a "partial canonical form" encodes the isomorphism class of the object plus some additional information. In the case of general groups, this additional information is a generating set. For p- and solvable groups, it is a composition series. As long as the additional pieces of information can be constructed gradually as a sequence of small steps, this bidirectional collision detection lemma can be applied. For general groups, each step corresponds to adding an additional generator to

[5]The simplification of breaking G into two subgroups consisting of the large and small composition factors was also suggested by Laci Babai. Originally, we achieved the same result using more complex methods.

the generating set. For p- and solvable groups, each step corresponds to adding another intermediate subgroup to the composition series.

The basic idea behind this bidirectional collision detection lemma is to note that the process of constructing the additional information yields a tree of low degree where the leaves correspond to "partial canonical forms." This can be used to deterministically compute two sets of leaves of size roughly $\sqrt{N}$ where N is the total number of leaves, with the property that the two objects for which we wish to test isomorphism are isomorphic if and only if the two sets contain leaves that correspond to a common canonical form. This can be determined efficiently using sorting or hashing, which allows the original isomorphism problem to be solved in roughly $\sqrt{N}$ time. By contrast, the natural algorithm takes roughly N time. We list some of the main corollaries of this bidirectional collision detection lemma below.

Theorem 2.2.3. *Solvable-group isomorphism (and hence p-group isomorphism) is decidable in $n^{(1/4)\log_p n+o(\log n)}$ deterministic time where p is the smallest prime dividing the order of the group.*

Theorem 2.2.4. *General group isomorphism is in $n^{(1/2)\log_p n+O(1)}$ deterministic time where p is the smallest prime dividing the order of the group.*

Thus, bidirectional collision detection yields square-root speedups over the best previous algorithms for p-groups, solvable groups and general groups. Square-root speedups are also possible for many other isomorphism problems including the graph and ring isomorphism problems.

There is also a quantum variant of our bidirectional collision detection lemma that typically yields cube-root speedups for the problems that it is applied to. In this way, we obtain an $n^{(1/6)\log_p n+O(1)}$ time quantum algorithm for p-group isomorphism, an $n^{(1/6)\log_p n+o(1)}$ time quantum algorithm for solvable-group isomorphism and an $n^{(1/3)\log_p n+O(1)}$ time quantum algorithm for testing isomorphism of general groups.

2.3 Chapter roadmap

In this section, we outline the chapters that follow and mention those which are joint work with others as well as those that have been published elsewhere. Chapters 3 and 4 cover basic results in group theory and quantum computing that are relevant to the rest of this thesis. Chapter 4 is necessary for Chapters 5 – 7 while Chapter 3 is required for Chapters 8 – 12. Chapters 5 – 7 are mostly independent of Chapter 3 and Chapters 8 – 12 are mostly independent of Chapter 4. Readers familiar with group theory and quantum computation may wish to skip Chapters 3 and 4.

In Chapter 5, we describe our results for 2D quantum circuits. A version of this chapter previously appeared [109] in the proceedings of the Conference on the Theory of Quantum Computation, Communication and Cryptography in 2013. Chapter 6 describes our results on oracles with internal randomness and is joint work with Aram Harrow that was published in the journal of Quantum Information and Computation [55]. Our tree isomorphism algorithm is described in Chapter 7 and was previously posted on the arXiv [110].

Chapter 8 reviews the color automorphism problem and Chapter 9 reviews previously known results on the group isomorphism problem. Chapter 10 describes a square-root speedup over the generator-enumeration algorithm for p-group isomorphism, is joint work with Fabian Wagner and will appear in the journal of Theoretical Computer Science [111]. Chapter 11 extends this speedup to solvable groups and previously appeared on the arXiv [106]. A preliminary version of the work in Chapters 10 and 11 appeared in the proceedings of the Symposium on Discrete Algorithms in 2013 [108]. The proofs were later refined (though the results remained the same). It is these refined proofs that appear in Chapters 10 and 11. In Chapter 12, we introduce our framework for obtaining square-root speedups for isomorphism problems and apply it to obtain a square-root speedup over the generator-enumeration algorithm for testing isomorphism of general groups as well as fourth-root speedups over the generator-enumeration algorithm for p- and solvable-groups. Chapter 12 was previously posted on the arXiv [107].

Chapter 3
GROUP THEORY BASICS

In this chapter, we review basic group theory with emphasis on ideas that are relevant to the algorithms that appear later in Chapters 8 – 12. For more details on group theory, see [112, 102] (or other group theory and algebra texts [58, 69, 105, 5, 128, 103]). Section 3.1 is about groups and subgroups: we start with the definition of a group, the notion of a subgroup and discuss related concepts including cosets, cyclic groups and Lagrange's theorem. We move on to normal subgroups in Section 3.2 and discuss quotient groups, simple groups and composition series. In Section 3.3, we define homomorphisms and isomorphisms and discuss the isomorphism theorems. We cover Abelian groups and their decomposition into cyclic groups in Section 3.4. In Section 3.5, we define central series, derived series and composition series and define the classes of nilpotent and solvable groups. We cover results for permutation groups in Section 3.6 including Cayley's theorem, the decomposition of permutations into cycles and algorithms for computing orbits and strong generating sets. Lastly, we discuss isomorphisms and automorphisms of graphs in Section 3.7.

3.1 Groups and subgroups

A group is an abstraction that encompasses many mathematical operations on sets including addition, multiplication (of non-zero numbers) and composition of permutations. We define it formally as follows.

Definition 3.1.1. *Let G be a set and let $* : G \times G \to G$ be a function. The pair $(G, *)$ is a* group *if the following axioms hold:*

Associativity *For all $x, y, z \in G$, $(x * y) * z = x * (y * z)$.*

Identity *There exists $e \in G$ such that for all $x \in G$, $e * x = x * e = x$.*

Inverses *For every $x \in G$, there exists $y \in G$ such that $x * y = y * x = e$ where e is as above.*

First, we mention a few notational conventions. Usually, the operation $*$ is clear from the context and we denote the group $(G, *)$ by just G. We also often abbreviate $x * y$ as xy; there is no ambiguity as long as we know what group x and y belong to. The group operation is often referred to as multiplication since the notation is similar. Due to the associativity axiom, we normally omit parenthesis and write $(xy)z = x(yz)$ as xyz. If the group is additive, we write $x + y$ instead of xy.

It is easy to see that the identity element e is unique, for if e and f both satisfy the identity axiom then $ef = f$ when we think of e as an identity, but, on the other hand $ef = e$ when we think of f as the identity. Thus, $e = f$. We therefore denote the *unique* identity element of the group G by 1_G or just 1 if G is clear from the context. The exception to this convention is additive groups where we denote the identity by 0.

The inverse of an element $x \in G$ is also unique. Let $y, z \in G$ such that $xy = yx = 1$ and $xz = zx = 1$. Then $yxz = y$ which implies that $z = y$. We therefore denote the *unique* inverse of x by x^{-1}. In an additive group, we denote the inverse of x by $-x$.

A group G is *finite* if the number of elements it contains is finite and is *infinite* otherwise. We will mostly only be concerned with finite groups in this thesis.

A *subgroup* of a group G is a subset H of G that is itself a group when $*$ is restricted to $H \times H$. It is easy to show the following simpler characterization of subgroups.

Proposition 3.1.2. *Let G be a group and let H be a subset of G. Then H is a subgroup of G (denoted $H \leq G$) if and only if all of the following hold:*

Closure *For all $x, y \in H$, $xy \in H$.*

Identity $1_G \in H$.

Inverses *For all $x \in H$, $x^{-1} \in H$.*

Note that we allow the case where $H = G$ when we say that H is a subgroup of G. If the elements of H form a proper subset of the elements of G, then we say that H is a *proper* *subgroup* of G and write $H < G$. The *improper* subgroup of G is G itself. The set $\{1\}$ is always a subgroup of G which we denote by 1 and call the *trivial subgroup*.

The identity 1_H of the subgroup H coincides with the identity 1_G of the group G; similarly, inverses taken over the subgroup H coincide with inverses taken over the group G.

Given two groups G and H, we can construct a new larger group called the *external direct product* of G and H which is denoted by $G \times H$. The elements of this group are $\{(g, h) \mid g \in G \text{ and } h \in H\}$ and the product of two elements (g_1, h_1) and (g_2, h_2) of $G \times H$ is defined to be (g_1g_2, h_1h_2). Usually, we abbreviate the term external direct product to *direct product*. The adjective external distinguishes the above construction from *internal direct products* which are equivalent but are defined differently. We'll discuss internal direct products further in Section 3.2, since they are defined terms of normal subgroups.

If S is a subset of a group G, then the *subgroup of G generated by S* (denoted $\langle S\rangle$) is the set of elements that can be obtained by finite sequences of group multiplication and inversion operations. Equivalently, it can be as the intersection of all subgroups of G that contain S. If $G = \langle S\rangle$, we say that S is a generating set for G. A group is *finitely generated* if it has a finite generating set.

Let H be a subgroup of a group G. If $x \in G$, then the set $xH = \{xh \mid h \in H\}$ is called a *left coset* of H in G. Similarly, the set $Hx = \{hx \mid h \in H\}$ is called a *right coset* of H in G. The set of all left cosets of H in G is denoted G/H. The size of of G/H is called the index of H in G and is denoted by $[G : H]$. An element of the coset xH is called a *representative.* A set that contains exactly one representative for each left coset is called a *complete left transversal*. *Complete right transversals* are defined analogously.

Proposition 3.1.3. *Let H be a subgroup of a group G. Then G/H is a partition of G.*

Proof. Clearly, every $x \in G$ is contained in the coset xH. Suppose that $x, y \in G$ such that $xH \cap yH \neq \emptyset$. Then $xh_1 = yh_2$ for some $h_i \in H$. Therefore, $xh_1h_2^{-1} = y$ so $y \in xH$ which implies that $xH = yH$.Therefore, every pair of cosets is either disjoint or equal. $\square$

The *order of a group G* is the size of the set G and is denoted by $|G|$. Lagrange's theorem relates the orders of a group to the orders of its subgroups.

Theorem 3.1.4 (Lagrange)**.** *Let H be a subgroup of a finite group G. Then $|H|$ divides $|G|$.*

Proof. This follows from the preceding proposition since it is easy to see that all cosets of H in G have the same cardinality. □

The *order of an element x of G* (denoted $|x|$) is the smallest positive integer k such that $x^k = 1$. If no such k exists, then x has infinite order and we write $|x| = \infty$.

A group G is called *cyclic* if $G = \langle x \rangle$. In this case, $G = \left\{x^k \mid k \in \mathbb{Z}\right\}$ where we define $x^k = \prod_{i=1}^{|k|} x^{-1}$ if $k < 0$, $x^k = \prod_{i=1}^{k} x$ if $k > 0$ and $x^k = 1$ if $k = 0$. One can easily verify that, if $i, j \in \mathbb{Z}$, then $x^i \cdot x^j = x^{i+j}$ and $(x^i)^j = x^{ij}$ as one would expect from this exponential notation. For any $x \in G$, we always have $|x| = |\langle x \rangle|$. This observation implies the following corollary of Lagrange's theorem.

Corollary 3.1.5. *Let x be an element of a finite group G. Then $|x|$ divides $|G|$.*

3.2 Normal subgroups and quotients

We now consider the circumstances under which the cosets G/H of a subgroup H in G themselves form a group. For this we need to define a way of multiplying two cosets. More generally, if A and B are subsets of a group G, we define their product $A \cdot B = \{ab \mid a \in A \text{ and } b \in B\}$. We can apply this operation to cosets xH and yH of H in G, but the result $xH \cdot yH$ is not always a coset of H in G. Since $x \in xH$, we see that $xH \cdot yH$ contains the coset xyH. On the other hand,

$$xH \cdot yH = xy(y^{-1}Hy) \cdot H$$

This is equal to xyH if and only if $(y^{-1}Hy) = H$. Since we want the product of any pair of cosets to yield another coset, we need $gHg^{-1} = H$ to hold for all $g \in G$. This is precisely the definition of a *normal subgroup.*

Definition 3.2.1. *A subgroup H of G is a* normal subgroup *of G (denoted $H \trianglelefteq G$) if $gHg^{-1} = H$ for all $g \in G$.*

If H is a proper normal subgroup of G, then we write $H \lhd G$. As alluded to above, $(G/H, \cdot)$ is a group if and only if $H \trianglelefteq G$; in this case, the product of two cosets xH and yH is xyH.

If H is a (not necessarily normal) subgroup of G and $g \in G$, then the set gHg^{-1} is called the *conjugate* of H by g and is written more compactly as H^g. If $x, g \in G$, then the *conjugate* of x by g is gxg^{-1} and is denoted more compactly by x^g.

The trivial and improper subgroups of a group are always normal; a group is called *simple* if it does not have any proper nontrivial normal subgroups.

Direct products can also be defined in terms of normal subgroups. If G and H are normal subgroups of K such that $G \cap H = 1$, then the *internal direct product* of G and H is defined to be $GH = \{gh \mid g \in G, h \in H\}$. We'll show that $(g_1h_1)(g_2h_2) = (g_1g_2)(h_1h_2)$ for all $g_i \in G$ and $h_i \in H$. To prove this, it suffices to show that $gh = hg$ for all $g \in G$ and $h \in H$. This is true if and only if each $ghg^{-1}h^{-1} = 1$. But $ghg^{-1} \in H$ and $hg^{-1}h^{-1} \in G$ which implies that $ghg^{-1}h^{-1} \in G \cap H = 1$. External and internal direct products are therefore essentially equivalent, are both referred to simply as direct products and the notation $G \times H$ is used for both.

3.3 Group homorphisms and isomorphisms

Now we move on to homomorphism and isomorphisms which allow us to relate the structure of one group to another. We start with the definition of a homomorphism.

Definition 3.3.1. *Let G and H be groups. A function $\phi : G \to H$ is a* homomorphism *if, for all $x, y \in G$, $\phi(xy) = \phi(x)\phi(y)$.*

Note that the multiplication of x by y is performed in G while the multiplication of $\phi(x)$ by $\phi(y)$ is in H. Thus, a homomorphism relates the operation of G to the operation of H. It is easy to see that every homomorphism ϕ satisfies $\phi(1) = 1$. The mapping $\phi : G \to H$ defined by $\phi(x) = 1$ for all $x \in G$ is called the *trivial homomorphism.* Homomorphisms also respect inverses in the sense that $\phi(x^{-1}) = (\phi(x))^{-1}$

An injective homomorphism is called a *monomorphism* and a surjective homomorphism is called an *epimorphism.* A homomorphism that is bijective is called an *isomorphism.* Two groups G and H are *isomorphic* (denoted $G \cong H$) if there exists an isomorphism between them. Intuitively, this means that the groups are the same except that the elements have different names. The set of all isomorphisms from G to H is denoted by $\mathrm{Iso}(G, H)$.

An isomorphism from a group G to itself is called an *automorphism.* The identity is always an automorphism. The set $\mathrm{Aut}(G)$ of all automorphisms of G form a group under function composition called the *automorphism group* of G. For every $g \in G$, define $\iota_g : G \to G$ by $\iota_g(x) = x^g$. Each ι_g is called an *inner automorphism* and the set $\mathrm{Inn}(G)$ of all inner automorphisms is a normal subgroup of $\mathrm{Aut}(G)$. The quotient $\mathrm{Out}(G) = \mathrm{Aut}(G)/\mathrm{Inn}(G)$ is called the *outer automorphism group* of G and its elements are called *outer automorphisms.*

3.3.1 Isomorphism theorems

Every homomorphism $\phi : G \to H$ gives rise to two important subgroups. The *kernel* of ϕ is the subgroup $\ker \phi = \{x \in G \mid \phi(x) = 1\}$. It is easy to verify that the kernel is a normal subgroup of G. The second subgroup is the *image* of ϕ which is defined as $\mathrm{Im}\, \phi = \phi[G]$. The image of a homomorphism is always a subgroup of H, but it need not be normal. The first isomorphism theorem relates the kernel to the image.

Theorem 3.3.2 (First isomorphism theorem)**.** *Let $\phi : G \to H$ be a homomorphism. Then*

$$G/\ker \phi \cong \mathrm{Im}\, \phi$$

The second and third isomorphism theorems can be obtained from the first by applying it to the right homomorphisms.

Theorem 3.3.3 (Second isomorphism theorem)**.** *Let G be a group, $K \leq G$ and $N \trianglelefteq G$. Then*

$$\frac{KN}{N} \cong \frac{K}{K \cap N}$$

Note that $KN \leq G$, since $(k_1n_1)(k_2n_2) = (k_1k_2)(k_2^{-1}n_1k_2n_2)$, $k_1k_2 \in K$ and $k_2^{-1}n_1k_2n_2 \in N$ since $N \trianglelefteq G$. (The other subgroup conditions are also easy to verify.) It is also easy to check that $K \cap N \trianglelefteq K$. The third isomorphism theorem allows us to cancel denominators in quotient groups in a manner analogous to fractions.

Theorem 3.3.4 (Third isomorphism theorem)**.** *Let G be a group, $N \trianglelefteq G$ and $H \trianglelefteq G$ with $N \leq H$. Then*

$$\frac{G/N}{H/N} \cong G/H$$

3.4 Abelian groups

An important class of groups are the *Abelian* groups where $xy = yx$ for all elements x and y in the group. The *center* of a group G is defined to be $Z(G) = \{z \in G \mid xz = zx \text{ for all } x \in G\}$. The subgroup $Z(G)$ is always Abelian and is a normal subgroup of G. Every group also always has a quotient that is Abelian. The *commutator* of $x, y \in G$ is defined to be $[x, y] = xyx^{-1}y^{-1}$. It is easy to see that $[x, y] = 1$ if and only if $xy = yx$. The subgroup $G' = [G, G]$ generated by all commutators of elements of G is called the *derived subgroup* or *commutator subgroup* of G. We can think of G' as all the ways in which two elements of G might not commute. The derived subgroup G' is a normal subgroup of G and the *Abelianization* of G is defined as G/G'. The quotient G/G' is Abelian since

$$\begin{aligned}(xG')(yG') &= xyG' \\ &= [x, y]yxG' \\ &= (yG')(xG')\end{aligned}$$

3.4.1 The structure of Abelian groups

The structure of finitely generated Abelian groups is fully understood and is defined in terms of the cyclic groups so first we consider the isomorphism classes of cyclic groups. Two finite cyclic groups are isomorphic if and only if they have the same order. We denote *the* cyclic group of order n by $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$ where $\mathbb{Z}$ is the group of integers under addition. All infinite cyclic groups are isomorphic to $\mathbb{Z}$ (and hence to each other by transitivity).

The structure theorem for finitely generated Abelian groups can be stated either in terms of *elementary divisors* or *invariant factors*[1]. Both forms are equivalent but sometimes one is more convenient than the other.

Theorem 3.4.1 (The structure of finitely generated Abelian groups (elementary divisor version))**.** *Let G be a finitely generated Abelian group. Then there exist (not necessarily distinct) primes $p_1, \ldots, p_k$, positive integers $e_1, \ldots, e_k$ and a positive integer m such that*

$$G \cong \mathbb{Z}_{p_1^{e_1}} \times \cdots \times \mathbb{Z}_{p_k^{e_k}} \times \mathbb{Z}^m$$

Moreover, this decomposition is unique up to reordering the factors.

Theorem 3.4.2 (The structure of finitely generated Abelian groups (invariant factor version))**.** *Let G be a finitely generated Abelian group. Then there exist positive integers $d_1, \ldots, d_k$ and m such that $d_i \mid d_{i+1}$ for each i and*

$$G \cong \mathbb{Z}_{d_1} \times \cdots \times \mathbb{Z}_{d_k} \times \mathbb{Z}^m$$

Moreover, this decomposition is unique up to reordering the factors.

This theorem is quite powerful and its proof is more involved than the other results considered up to this point. One way to prove it is to choose a finite generating set of G and write down an integer matrix that corresponds to the linear dependence relations between these generators. By computing a canonical form of this matrix known as the smith normal form, one can obtain the structure constants in the above theorem.

[1]These terms come from a more general version of the theorem for finitely-generated modules over a principal ideal domain (cf. [104]).

3.5 Series of subgroups

In this section, we introduce the notion of a series of a group.

Definition 3.5.1. *Let G be a group. A* series *for G is a sequence of subgroups $G_0 = 1 < \cdots < G_m = G$.*

The *length* of a series is the number of subgroups in the series minus one (i.e. m in the definition above). Two series for groups G and H are *isomorphic* if there is an isomorphism $\phi : G \to H$ that sends each subgroup in the series for G to the corresponding subgroup in the series for H.

Almost all important types of series fall under the class of *subnormal series* which are series where $G_i \lhd G_{i+1}$ for each i. In a subnormal series, it is not necessarily the case that each G_i is normal in the entire group G, as it is only required to be normal in G_{i+1}. When each G_i is a normal subgroup of G, we say that the series is a *normal series*. There are several subclasses of series that can be used to relax the notion of an Abelian group. The first of these is the notion of a *central series*.

Definition 3.5.2. *Let G be a group. A* central series *for G is a normal series*

$$G_0 = 1 \lhd \cdots \lhd G_m = G$$

such that $G_{i+1}/G_i \leq Z(G/G_i)$.

Not all groups have a central series. If a group does have a central series, it is called *nilpotent*. The length of the shortest central series of a group is called its *nilpotency class*. The nilpotency class can be thought of as a measure of how far a nilpotent group is from being Abelian. An important subclass of nilpotent groups is the nilpotent groups of class 2. These are those groups G where $G/Z(G)$ is Abelian; in such a group, every pair of elements commutes up to a central element. We now consider a more general class of groups.

A class of groups closely related to the nilpotent groups are the groups whose order is a power of a prime p. Such a group is called a *p-group*. It can be shown that every p-group is nilpotent so the p-groups form another subclass of the nilpotent groups. In fact, every nilpotent group is a direct product of p_i-groups where the p_i's are distinct primes.

Definition 3.5.3. *Let G be a group. The* derived series *of G is*

$$\cdots \lhd \cdots \lhd G^{(1)} \lhd G^{(0)} = G$$

where $G^{(i+1)} = [G^{(i)}, G^{(i)}]$ is the i^{th} derived subgroup.

In general, it need not be the case that the derived series terminates with the identity subgroup. (In the case of an infinite group, it may not even terminate at all). If there exists a finite k such that $G^{(k)} = 1$, then G is a *solvable group* and the least such k is called the *derived length* of G. The condition $G_{i+1}/G_i \leq Z(G/G_i)$ in the definition of a nilpotent group is equivalent to the condition $[G_{i+1}, G] \trianglelefteq G_i$. From this, we see that every nilpotent group is solvable. However, the converse does not hold.

An essential definition for Part II is the notion of a composition series.

Definition 3.5.4. *Let G be a group. A* composition series *for G is a subnormal series $G_0 = 1 \lhd \cdots \lhd G_m = G$ such that each G_{i+1}/G_i is simple.*

Alternatively, a composition series can be equivalently defined as a maximal subnormal series. Unlike central series, every group has a composition series. The factors of a composition series are called *composition factors*; by the Jordan-Hölder Theorem (cf. [102, 105]), the multiset of composition factors is determined up to isomorphism by G. It can be shown that a group is solvable if and only if all of its composition factors are cyclic.

3.6 Permutation groups

In this section, we cover the basics of permutation group theory. For more details, see [40].

Let Ω be a set. Then a *permutation* π of Ω is a bijection from Ω to itself. It is easy to verify that the set S_Ω of all permutations of Ω forms a group under composition of functions. We call this the *symmetric group* on Ω. A *permutation group* G on Ω is a subgroup of S_Ω. The *degree* of G is equal to $|\Omega|$. For each positive integer n, we define $S_n = S_{[n]}$.

Cayley's theorem tells us that every group is isomorphic to a subgroup of a symmetric group.

Theorem 3.6.1 (Cayley's theorem)**.** *Let G be a group. Then G is isomorphic to the subgroup of S_G defined by $\{\pi_g\alpha = g\alpha \text{ for all } \alpha \in G \mid g \in G\}$.*

A *cycle* is a permutation π where there exist distinct $\alpha_1, \ldots, \alpha_k \in \Omega$ such that $\pi\alpha_i = \alpha_{i+1}$ for $1 \leq i < k$ and $\pi\alpha_k = \alpha_1$. We denote the cycle π by $(\alpha_1 \ldots \alpha_k)$. The *orbit* $G\alpha$ of an element $\alpha \in \Omega$ under the action of a permutation group G is is the set $\{g\alpha \mid g \in G\}$. Now let $\pi \in S_\Omega$. We can partition Ω into its orbits under the subgroup $\langle\pi\rangle$ generated by π. Let $\Omega_i = \{\alpha_1, \ldots, \alpha_{n_i}\}$ denote the i^{th} such orbit and let m be the number of orbits. It is easy to see that the restriction $\pi\big|_{\Omega_i} : \Omega_i \to \Omega_i$ of π is a cycle and is an element of S_{Ω_i}. Then

$$\pi = \pi\big|_{\Omega_1} \cdots \pi\big|_{\Omega_m}$$

This is called the *cycle decomposition* of π.

Let $\alpha \in \Omega$, $\Delta \subseteq \Omega$ and let G be a permutation group on Ω. The *orbit* $G\Delta$ of the subset Δ under the permutation group G is is the set $\{g\beta \mid g \in G, \beta \in \Omega\}$. The *stabilizer subgroup* of α is the set $G_\alpha = \{g \in G \mid g\alpha = \alpha\}$. The Orbit-Stabilizer Theorem relates the orbit of an element to its stabilizer subgroup.

Theorem 3.6.2. *Let G be a group of permutations on a set Ω and let $\alpha \in \Omega$. Then*

$$|G/G_\alpha| = |G\alpha|$$

Stabilizers can also be defined for subsets of Ω as well as individual elements. In this case there are two different types of stabilizers. Let $\Delta \subseteq \Omega$. The *pointwise stabilizer* of Δ is $G_{(\Delta)} = \{g \in G \mid g\beta = \beta \text{ for all } \beta \in \Delta\}$. The *setwise stabilizer* of Δ is $G_\Delta = \{g \in G \mid g\Delta = \Delta\}$. As we will see later in this section, the pointwise stabilizer can be computed in polynomial time. The complexity of computing the setwise worst case in the worst case is not known, but there is evidence that it is NP-hard. Efficient algorithms [76, 18, 16] are known for cases where G has certain structural properties and are one of the building blocks for the current best algorithm for graph isomorphism [16]. We will discuss the complexity of computing the setwise stabilizer further later in Chapter 8.

3.6.1 Strong generating sets

Many algorithms for permutation groups are based on (or at least use) a concept called a *strong generating set* [117] (which we will define shortly.) Strong generating sets can be used to perform many tasks for permutation groups in polynomial time including computing the order of a permutation group, testing membership in a permutation group, finding any pointwise stabilizer of a permutation group and determining the kernel of a homomorphism between permutation groups. As a concrete example, one can use strong generating sets to solve the Rubik's cube. To define a strong generating set, we need the notion of a base.

For notational convenience, we define strong generating sets only for subgroups of S_n, but of course everything we do also works for any permutation group.

Definition 3.6.3. *Let $G \leq S_n$ and define $G_i = G_{(n,\ldots,i+1)}$. Then a subset $S \subseteq G$ is a* strong generating set *if $G_i = \langle G_i \cap S \rangle$ for all i.*

First, it is obvious that strong generating sets always exist since we can just take a generating set for each G_i. We will sketch how to compute a strong generating set in polynomial time. First, a brief digression is necessary since we haven't yet explained what polynomial time means for permutation groups.

In computational contexts, a permutation group G on Ω is specified by a generating set S. A permutation is represented by listing the image of each element in Ω. The complexity of permutation group algorithms is measured in terms of the size of S and the degree of G. Note that the input is linear in both of these quantities, so this is consistent with the usual definition of polynomial time.

To compute a strong generating set for G, define an $n \times n$ matrix M indexed by $[n]$ whose elements are either elements of G or $\emptyset$. Initially, we set all entries of M to $\emptyset$. If an entry is not $\emptyset$, then we require that $M_{ij} \in G_i$ and that it maps i to j. Our plan is to fill in the entries of M using a sifting procedure.

Suppose that $\pi \in G$. If $M_{n,\pi(n)} \neq \emptyset$, then let $\sigma = M_{n,\pi(n)}^{-1}\pi \in G_{n-1}$. Then we compute $\sigma(n-1)$. Either $M_{n-1,\sigma(n-1)} \neq \emptyset$ or $M_{n-1,\sigma(n-1)}^{-1}\sigma \in G_{n-2}$. Continuing in this manner, we eventually obtain $\sigma = M_{i+1,k_{i+1}}^{-1} \cdots M_{n,k_n}^{-1}\pi \in G_i$ where either $i = 1$ or $M_{i,\sigma(i)} = \emptyset$. If $i = 1$, then we have $\pi = M_{n,k_n} \cdots M_{2,k_2}$. When $M_{i,\sigma(i)} = \emptyset$, we update M by setting $M_{i,\sigma(i)} = \sigma$. We call the procedure just described in this paragraph *sifting by* π.

We repeatedly sift elements of G until the entries of M that are not $\emptyset$ form a strong generating set. Let T be a set that is initialized to S. At each step we choose an element π from T and sift by π. Whenever a new element σ is added to M by setting $M_{i,\sigma(i)} = \sigma$, we add all products of the forms $M_{i,\sigma(i)}M_{jk}$ and $M_{jk}M_{i,\sigma(i)}$ where $M_{jk} \neq \emptyset$ to T. This procedure continues until T is empty.

It is clear that this procedure halts in polynomial time since an element can be sifted in polynomial time and at most $|S| + n^4$ elements are ever added to T. We claim that when it halts, the elements of T form a strong generating set. The elements of M contained in G_i are $S_i = \{M_{jk} \neq \emptyset \mid j \leq i\}$. The argument that $G_i = \langle S_i \rangle$ is slightly more involved and we will not give it here; however, it follows from the fact that any product of elements of M must sift to the identity after the algorithm has terminated (cf. [120]).

For more details on the analysis as well as more carefully optimized variants of this algorithm, see [114].

3.7 Isomorphisms and automorphisms of graphs

One important application of group theory we will see later in this chapter is symmetries of graphs. Let X and Y be graphs. A bijection $\phi : X \to Y$ is a *graph isomorphism* if each pair (x, y) is an edge if and only if $(\phi(x), \phi(y))$ is an edge. Two graphs are *isomorphic* if there is an isomorphism between them. Intuitively, this means that the graphs have the same structure but the elements have different labels. An *graph automorphism* is an isomorphism from a graph to itself. The set $\mathrm{Aut}(X)$ denotes the group of all automorphisms of the graph X.

The isomorphisms from a graph X to a graph Y are closely related to the automorphism groups of X and Y. Suppose that $\phi, \theta : X \to Y$ are isomorphisms. Then $\phi^{-1}\theta \in \mathrm{Aut}(X)$ so $\theta \in \phi\mathrm{Aut}(X)$. Thus, the set $\mathrm{Iso}(X, Y)$ of all isomorphisms from X to Y is the coset $\phi\mathrm{Aut}(X)$ of the automorphism group. The problem of testing if two graphs are isomorphic is equivalent to the problem of computing generators of the automorphism group under Turing reductions.

Chapter 4
QUANTUM COMPUTING BASICS

In this chapter, we introduce the basic quantum computing background that is required for the rest of this work. For a more extensive treatment of the subject, see [89]. In Section 4.2, we introduce qubits and Dirac notation. We introduce elementary operations and universal gate sets in Section 4.3. We show that entanglement can be used to move a quantum state from one register to another using only local operations in Section 4.4. In Section 4.5, we introduce the swap test which allows us to compare quantum states under certain conditions. In Section 4.6, we discuss Grover's algorithm which shows that brute force search over a set of size N takes only $O(\sqrt{N})$ time on a quantum computer. Finally, we cover the hidden subgroup problem in Section 4.7, which is the basis of most exponential speedups over classical algorithms.

4.1 Quantum states and operations

In this section, we describe the basics of quantum computation without regard for efficiency. While the state of a classical computer is described by a binary string, the state of a quantum computer is a complex vector in $\mathbb{C}^N$ for some N. The standard basis vectors for this space are denoted by $|k\rangle$ which corresponds to the N-dimensional column vector that has 0 in all of its entries except the k^{th} which is 1 where $0 \leq k < N$. A general state is denoted by $|\psi\rangle$ and has the form

$$|\psi\rangle = \sum_{k=0}^{N} \alpha_k |k\rangle$$

The state $|\psi\rangle$ is called a *ket*. The *amplitude* of $|k\rangle$ is α_k and the *phase* of $|k\rangle$ is $\alpha_k / |\alpha_k|$. The *complex conjugate transpose* of a vector is the vector one obtains by taking the transpose of the vector and then taking the complex conjugate of each element. We denote the complex conjugate transpose of $|\psi\rangle$ by $\langle\psi|$ (which is called a *bra*). The *outer product* $|j\rangle\langle k|$ denotes the $N \times N$ matrix that has a 1 at (j,k) and 0 elsewhere. For general states $|\psi\rangle = \sum_{k=0}^{N} \alpha_k |k\rangle$ and $|\phi\rangle = \sum_{k=0}^{N} \beta_k |k\rangle$,

$$|\psi\rangle\langle\phi| = \sum_{j,k=0}^{N} \alpha_j \beta_k^* |j\rangle\langle k|$$

The inner product of two states $|\psi\rangle$ and $|\phi\rangle$ is denoted $\langle\psi|\phi\rangle$ and is sometimes also called a *braket* (hence the names bra and ket).

4.1.1 Unitary matrices

An $N \times N$ matrix U is *unitary* if $UU^\dagger = I$ where $U^\dagger$ denotes the complex conjugate transpose which is the transpose of the matrix one obtains by taking the complex conjugate of each element of U. Multiplication by a unitary matrix is one class of quantum operations that can be performed on $\mathbb{C}^N$.

4.1.2 Measurements

Unlike a classical computer, we cannot directly inspect the current state of a quantum computer. Instead, we must perform *measurements* on the state in order to recover information about it. After a measurement is performed, we obtain each *measurement outcome* with some probability (which may be 0 for some measurement outcomes) and the state is transformed into a new state. This is an important difference from classical computing since inspecting a classical bit string does not change it.

The most basic measurement simply projects onto the standard basis. In this case, the measurement outcomes are simply the labels $0 \leq k < N$ of the standard basis vectors. If the state is $|\psi\rangle = \sum_{k=0}^{N-1} \alpha_k |k\rangle$ before the measurement is performed, than with probability $|\alpha_k|^2 / \|\psi\|^2$, outcome k occurs and the state becomes $|k\rangle$ after the measurement. For convenience, we will require from now on that all states are normalized so that $\|\psi\| = 1$. There is nothing special here about the standard basis. More generally, if $\mathcal{B} = \{|\psi_k\rangle \mid 1 \leq k \leq N\}$ is an arbitrary basis of $\mathbb{C}^N$, then we can also perform a projective measurement onto the basis $\mathcal{B}$.

In the most general setting, a measurement can be any collection of matrices $\{M_j \mid 1 \leq j \leq m\}$ such that $\sum_{j=1}^{m} M_j^\dagger M_j = I$. Each matrix M_j is then referred to as the j^{th} *measurement operator*. When the measurement is performed on a state $|\psi\rangle$, with probability $\langle\psi| M_j^\dagger M_j |\psi\rangle$ outcome j occurs and the state is transformed into

$$\frac{M_j |\psi\rangle}{\sqrt{\langle\psi| M_j^\dagger M_j |\psi\rangle}}$$

From these equations, one can see that the states $|\psi\rangle$ and $e^{i\theta} |\psi\rangle$ (where $\theta \in \mathbb{R}$) are indistinguishable under any measurements. We therefore can multiply a quantum state by any complex number of norm 1 without changing the behavior of the system.

4.1.3 Density matrices

Up until now, we have represented quantum states by vectors of the form

$$|\psi\rangle = \sum_{k=0}^{N} \alpha_k |k\rangle$$

From now on, we will refer to such states as a *pure states*, in order to distinguish them from the more general class of *mixed states*, which we will now introduce. In this case, the state of a quantum system is a mixture of a pure states. In other words, there is a collection of states $|\psi_i\rangle$ and probabilities p_i such that the system is in state $|\psi_i\rangle$ with probability p_i. Such a state is represented by the *density matrix*

$$\rho = \sum_i p_i |\psi_i\rangle \langle\psi_i| \tag{4.1}$$

For brevity, the density matrix $|\psi\rangle\langle\psi|$ for a pure state $|\psi\rangle$ is often denoted by ϕ. Using this convention, the above equation becomes $\rho = \sum_i p_i \psi_i$.

Mixed states typically occur when one measures part of the quantum system but leaves the rest of it undisturbed. In this case, each of the states $|\psi_i\rangle$ is the state that remains when measurement outcome i occurs (which happens with probability p_i).

A matrix M is *Hermitian* if $M = M^\dagger$. We say that a Hermitian matrix is *positive semidefinite* if all of its eigenvalues are nonegative. The *trace* $\operatorname{tr} M$ of a matrix M is defined to be the sum of its diagonal entries in any basis. It is a basic property of the trace that it is independent of the basis chosen. If M is Hermitian, than the trace is also the sum of the eigenvalues of M. A density matrix can alternatively be defined as a positive semidefinite matrix ρ such that $\operatorname{tr} \rho = 1$. In general, the state of a quantum system can be any density matrix. By diagonalization, this definition is equivalent to (4.1).

If the system is in state ρ where ρ is a density matrix, then applying a unitary U results in the state $U\rho U^\dagger$. Applying the measurement results in outcome j with probability $\operatorname{tr} M_j^\dagger M_j \rho$ and transforms the system into the state

$$\frac{M_j \rho M_j^\dagger}{\operatorname{tr} M_j^\dagger M_j \rho}$$

One can verify that these definitions are consistent with the definitions previously given for pure states.

4.2 Tensor products and qubits

In the previous section, we took an abstract approach without worrying about how states are actually constructed. On a classical computer, the basic unit of information is the bit which takes values in $\{0, 1\}$. On a quantum computer, the basic unit of information is the *qubit*. When the state is pure, a qubit takes values in the complex vector space $\mathbb{C}^2$. When it is a mixed state, a qubit is represented by a 2×2 density matrix.

On a classical computer, the state space of two smaller m- and n-bit systems is the direct product of their state spaces and the global state is simply the concatenation of the states of the subsystems. The *tensor product* is the quantum analogue of concatenation. We define the tensor product of dimensions M and N to be the MN-dimensional complex space $\mathbb{C}^M \otimes \mathbb{C}^N$; it is spanned by tensor products of vectors of the form $|j\rangle \otimes |k\rangle$ which we define to be linearly independent and orthogonal. The tensor product $|\psi\rangle \otimes |\phi\rangle$ is defined by requiring that the operator $\otimes$ is bilinear. We often abbreviate $|\psi\rangle \otimes |\phi\rangle$ as $|\psi\rangle\,|\phi\rangle$, $|\psi, \phi\rangle$ or $|\psi\phi\rangle$. It is important to note that $\psi\phi$ does not denote multiplication in this context, even when ψ and ϕ are numbers.

Let $|\psi\rangle = \sum_{j=0}^{M-1} \sum k = 0^{N-1} \alpha_{jk} |jk\rangle$ and $|\phi\rangle = \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} \beta_{jk} |jk\rangle$ be pure states in $\mathbb{C}^M \otimes \mathbb{C}^N$. The inner product of $|\psi\rangle$ and $|\phi\rangle$ is defined to be $\langle\psi|\phi\rangle = \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} \alpha\beta^*$. It is often convenient to refer to the components of a tensor product as *registers*. For example, in the basis state $|jk\rangle = |j\rangle \otimes |k\rangle$ in the above superpositions, $|j\rangle$ is stored in the first register and $|k\rangle$ is stored in the second. Using this terminology, if U and V are $M \times M$ and $N \times N$ unitary matrices, than $U \otimes V$ corresponds to applying U to the M-dimensional register and V to the N-dimensional register. Formally, $U \otimes V$ is defined by

$$(U \otimes V)(|\psi\rangle \otimes |\phi\rangle) = (U|\psi\rangle) \otimes (V|\phi\rangle)$$

for all $|\psi\rangle \in \mathbb{C}^M$ and $|\phi\rangle \in \mathbb{C}^N$. If $\{A_j \mid 1 \le j \le a\}$ and $\{B_k \mid 1 \le k \le b\}$ are measurement operators on $\mathbb{C}^M$ and $\mathbb{C}^N$, then $\{A_j \otimes B_k \mid 1 \le j \le a \text{ and } 1 \le k \le b\}$ is a measurement operator on $\mathbb{C}^M \otimes \mathbb{C}^N$.

If the density matrix of the M-dimensional register is ρ and the state of the N-dimensional register is σ, then the density matrix for the overall system is $\rho \otimes \sigma$. In general, the state of the overall system is an $MN \times MN$ density matrix.

Since quantum systems are combined by taking tensor products, all states on a quantum computer are built from tensor products of qubits. Therefore, the state space of an n-qubit quantum computer is $(\mathbb{C}^2)^{\otimes n} = \bigotimes_{k=1}^{n} \mathbb{C}^2 \cong \mathbb{C}^{2^n}$. As we often do with classical computers, we will work from a higher level of abstraction and deal with N-dimensional registers. However, it is important to remember that in the end everything must be done in terms of qubits[1].

[1]Just as one could construct a classical computer in which the basic unit of information had more than 2 values, it is conceivable that one could implement a quantum computer the basic unit of information was a d-valued register. However, since these can be implemented in terms of qubits, we shall assume that qubits are the basic unit of storage.

4.3 Elementary operations

In the last section, we introduced qubits: the elementary storage primitives used on a quantum computer to construct larger quantum registers. In this section, we discuss the elementary operations from which all other quantum operations are built. A composition of the basic operations (or gates) introduced in this section will be called a *quantum circuit.* When we say that something can be done on a quantum computer in time T, we mean that there is a quantum circuit with T gates that accomplishes this task.

Because quantum error correction can only handle a finite set of gates, it isn't feasible to implement arbitrary quantum operations directly. Instead there is a small set of gates that can be performed fault-tolerantly and all other logical operations must be created by composing these gates. A set of gates is called *universal* if compositions of gates in the set can approximate any other operation to an arbitrary degree of precision. One example of a universal gate set consists of the Hadamard, $\pi/8$ and CNOT gates. We will now introduce each of the gates in this set. The *Hadamard gate* acts on a single qubit and is represented by the 2×2 unitary matrix

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

This matrix maps the state $|0\rangle$ to the superposition $(|0\rangle + |1\rangle)/\sqrt{2}$. When the input state is $|1\rangle$, the output is $(|0\rangle - |1\rangle)/\sqrt{2}$. Thus, in general, it sends $|k\rangle$ to $(|0\rangle + (-1)^k |1\rangle)/\sqrt{2}$, so the Hadamard gate creates a superposition of the same basis states for both inputs but multiplies the coefficient of $|1\rangle$ by -1 in the output state when the input is $|1\rangle$. The $\pi/8$ gate is also a single qubit gate and has the effect of multiplying the phase of $|1\rangle$ by $e^{i\pi/4}$; it is represented by the 2×2 unitary matrix

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$$

The controlled-NOT (CNOT) gate is a two-qubit gate. If the input is a basis state $|j\rangle\,|k\rangle$ where $j, k \in \{0, 1\}$, then the output is $|j\rangle\,|j \oplus k\rangle$ where $\oplus$ denotes addition modulo 2. It is represented by the 4×4 unitary matrix

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

While the Hadamard, $\pi/8$ and CNOT gates are sufficient to approximate any $N \times N$ unitary matrix, there are also several other useful gates that we will now introduce. The Pauli gates are a set of single-qubit operations that form a group under multiplication (up to global phase). They are given by the 2×2 unitary matrices

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad X = \sigma_X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Y = \sigma_Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad Z = \sigma_Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

The CNOT gate we introduced earlier is a special case of a *controlled operation*. In general, if U is a unitary acting on an N-dimensional register then a *controlled-U* operation with n controls is an operation that acts on the space $(\mathbb{C}^2)^{\otimes n} \otimes \mathbb{C}^N$. If the input is $|b_1 \cdots b_n\rangle \otimes |\psi\rangle$ where each $b_k \in \{0, 1\}$ and $|\psi\rangle \in \mathbb{C}^N$, then the output is $|b_1 \cdots b_n\rangle \otimes (U |\psi\rangle)$ if each $b_k = 1$ and $|b_1 \cdots b_n\rangle \otimes |\psi\rangle$ otherwise. The unitary matrix for this operation is given by

$$\text{CU} = (I - |1^n\rangle \langle 1^n|) \otimes I + |1^n\rangle \langle 1^n| \otimes U$$

We note that the CNOT operation is recovered from this formula when $U = X$ and $n = 1$. Another important controlled operation is the *Toffoli gate*. In this case, $U = X$ and $n = 2$; this operation has the effect of taking the AND of the first two qubits and XORing it into the third when the input is a computational basis state. One can also consider the problem of implementing a controlled operation for some fixed U and general n. Barenco et al. showed [20] that this problem can be solved using $O(n^2)$ basic operations.

4.4 Quantum teleportation

In this section we introduce *quantum teleportation* [21]. As we shall see later in Chapter 5, teleportation has applications to efficiently implementing quantum circuits in 2D quantum architectures.

Quantum teleportation allows the information in a qubit to be moved to a distant location using a phenomenon known as *quantum entanglement.* A pure state $|\psi\rangle$ in $\mathbb{C}^M \otimes \mathbb{C}^N$ is *separable* if $|\psi\rangle = |\psi_1\rangle |\psi_2\rangle$ where $|\psi_1\rangle \in \mathbb{C}^M$ and $|\psi_2\rangle \in \mathbb{C}^N$; it is *entangled* if no such decomposition exists.

The classic examples of entangled states are the *Bell basis* which we denote by

$$|\Phi_0\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \quad |\Phi_1\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}} \quad |\Phi_2\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} \quad |\Phi_3\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$

Up to global phase, these can be written as $|\Phi_\ell\rangle^{AB} = \sigma_\ell^B |\Phi_0\rangle^{AB}$. (The superscripts A and B are simply labels that allow us to refer to the corresponding registers.) In the quantum teleportation setting, Alice has a state $|\psi\rangle^S = \alpha |0\rangle^S + \beta |1\rangle^S$ that she wishes to send to Bob. The two parties are not allowed to send quantum states to each other but each have one qubit of a Bell state $\sigma_\ell^B |\Phi_0\rangle$ and can communicate classically.

To perform quantum teleportation, Alice performs a measurement in the Bell basis on the SA registers. If the measurement outcome is $|\Phi_k\rangle$, then a simple calculation shows that the resulting state is

$$|\Phi_k\rangle^{SA} \otimes \sigma_\ell \sigma_k |\psi\rangle^B$$

Alice then sends the classical measurement outcome k to Bob; since ℓ is known, Bob then causes the overall state to become

$$|\Phi_k\rangle^{SA} \otimes |\psi\rangle^B$$

up to global phase by applying the Pauli operation $(\sigma_\ell \sigma_k)^{-1}$ to his register B. Observe that Alice's state $|\psi\rangle$ has been recovered in Bob's register. This process only uses local operations, entanglement and classical communication, so it can be interpreted as showing that entanglement combined with classical communication yields quantum communication.

4.5 The swap test

As we mentioned in Chapters 1 and 2, one way of approaching isomorphism problems from a quantum perspective is to prepare a quantum state that represents the isomorphism class of an object [3]. In the case of two graphs X and Y, the desired states $|X\rangle$ and $|Y\rangle$ have the property that $|X\rangle = |Y\rangle$ if $X \cong Y$ and $\langle X|Y\rangle = 0$ if $X \ncong Y$.

In order to make use of such states, we need a way to compare two orthonormal states. This is of course easy for computational basis states. It can also be done in the case where $|X\rangle$ and $|Y\rangle$ can be prepared efficiently using the quantum circuits U_X and U_Y applied to the state $|0\rangle$. In this case, we can simply prepare the state $|X\rangle$, apply $U_Y^\dagger$ and measure in the computational basis. If $|X\rangle = |Y\rangle$, then we will observe $|0\rangle$ while if $\langle X|Y\rangle = 0$, we will observe a computational basis state that is orthogonal $|0\rangle$. This latter claim is a simple consequence of the fact that unitary matrices respect the inner product.

However, we would also like a way to compare states that are prepared using non-unitary procedures such as those that involve measurements. The *swap test* [28] can compare pairs of arbitrary states. It does not depend on how the states are prepared so any method can be used. In fact, the swap test provides a method for estimating the absolute value of inner product between two states, so we can do more than just distinguish equal states from orthogonal states.

We now give a description of the swap test. Let $|\psi\rangle$ and $|\phi\rangle$ be the two states in $\mathbb{C}^N$ that we wish to compare. The swap test is performed by preparing a qubit $|c\rangle$ in the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and applying a swap controlled by $|c\rangle$ to the states $|\psi\rangle$ and $|\phi\rangle$. This results in the state

$$\frac{1}{\sqrt{2}}(|0\rangle|\psi\rangle|\phi\rangle + |1\rangle|\phi\rangle|\psi\rangle)$$

A Hadamard gate is then applied to the control qubit and it is measured in the computational basis. A simple calculation shows that the probability of measuring 0 is $\Pr(0) = (1 + |\langle\psi|\phi\rangle|^2)/2$. Therefore, if $|\psi\rangle = |\phi\rangle$, then the swap test will always output $|0\rangle$. However, if $\langle\psi|\phi\rangle = 0$ then 0 will be observed with probability exactly $1/2$. Thus, the swap test allows these two cases to be distinguished with one-sided error. By repeating the swap test on more pairs of states, the probability of error can be made arbitrarily close to 1.

4.6 Grover's algorithm

In this section, we'll explore Grover's algorithm [52, 25][2], which we will apply to isomorphism testing later in Chapter 12.

Grover's algorithm can be interpreted as a quantum analogue of brute-force search. Suppose that we are trying to solve a difficult problem. An obvious algorithm is to perform a brute force search in which we enumerate all candidates for solutions and test if each of them really is a solution. If the space being searched has size N, this takes $\Theta(N)$ time classically even when we allow randomized algorithms. On a quantum computer, we can use Grover's algorithm to accomplish this task in $\Theta(\sqrt{N}\mathsf{polylog} N)$ time.

4.6.1 The algorithm

The notion of an *oracle* is essential in Grover's algorithm. Assume[3] that $N = 2^n$ and let $f : \{0,1\}^n \to \{0,1\}$ be a function such that $f(x) = 1$ if and only if x is a solution to the search problem. Then the oracle for f is $\mathcal{O}_f : \mathbb{C}^n \otimes \mathbb{C}^2 \to \mathbb{C}^n \otimes \mathbb{C}^2$ where $\mathcal{O}_f \left|x\right\rangle \left|y\right\rangle = \left|x\right\rangle \left|y \oplus f(x)\right\rangle$. When $\mathcal{O}_f$ is applied to a state $\left|x\right\rangle \left(\frac{|0\rangle - |1\rangle}{2}\right)$, we get $(-1)^{f(x)} \left|x\right\rangle \left(\frac{|0\rangle - |1\rangle}{2}\right)$. Thus, the phase of each basis state that corresponds to a solution is multiplied by -1. Since $\frac{|0\rangle - |1\rangle}{2} = HX\left|0\right\rangle$, this state can be initialized efficiently so we can view the oracle as acting on the phases in this way. Since it is more convenient for Grover's algorithm, we define the operation $\mathcal{O}'_f : \mathbb{C}^n \to \mathbb{C}^n$ where $\mathcal{O}'_f \left|x\right\rangle = (-1)^{f(x)} \left|x\right\rangle$ and use it instead of $\mathcal{O}_f$.

We start with the state $\left|0\right\rangle$. The first step in Grover's algorithm is transform this into the state

$$\left|\psi\right\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \left|x\right\rangle \tag{4.2}$$

This is accomplished by applying a Hadamard gate to every qubit.

Grover's algorithm works by applying the *Grover iteration* $G = H^{\otimes n}(2\left|0^n\right\rangle\left\langle 0\right| - I)H^{\otimes n}\mathcal{O}'_f$ k times where k is a carefully chosen positive integer which we shall discuss later. Note that

[2]We follow the description from [89].

[3]Note that we can always round N up to the next power of 2.

$2\left|0^n\right\rangle\left\langle 0\right| - I$ is the operation that multiplies the phase of the basis state $|0\rangle$ by -1 and leaves all other phases unchanged. This operation can therefore be implemented using a controlled AND operation in conjunction with single-qubit gates.

4.6.2 Analysis

We will now sketch the analysis of Grover's algorithm. First, we note that

$$H^{\otimes n}(2\left|0^n\right\rangle\left\langle 0\right| - I)H^{\otimes n} = 2\left|\psi\right\rangle\left\langle\psi\right| - I$$

so

$$G = (2\left|\psi\right\rangle\left\langle\psi\right| - I)\mathcal{O}'_f$$

Let $M = |f^{-1}(1)|$ be the number of solutions in the search problem. The main idea is to consider the two dimensional subspace S spanned by the uniform superposition $|\alpha\rangle = \frac{1}{\sqrt{N-M}}\sum_{x\in f^{-1}(0)}|x\rangle$ of all non-solutions and the uniform superposition $|\beta\rangle = \frac{1}{\sqrt{M}}\sum_{x\in f^{-1}(1)}|x\rangle$ of all solutions. It is easy to see that

$$|\psi\rangle = \sqrt{\frac{N-M}{N}}\,|\alpha\rangle + \sqrt{\frac{M}{N}}\,|\beta\rangle$$

so the initial state before any Grover iterations have been performed is in the subspace S. This also implies that $2\left|\psi\right\rangle\left\langle\psi\right| - I$ maps vectors in S into S. It is easy to verify that on the subspace S, $\mathcal{O}'_f$ is equivalent to the operation

$$2\left|\alpha\right\rangle\left\langle\alpha\right| - I$$

so $\mathcal{O}'_f$ preserves the subspace S as well.

By the preceding paragraph, we can restrict our analysis to the subspace S. Each Grover iteration then corresponds to the operation

$$(2\left|\psi\right\rangle\left\langle\psi\right| - I)\,(2\left|\alpha\right\rangle\left\langle\alpha\right| - I)$$

which is a reflection about $|\alpha\rangle$ followed by a reflection about $|\psi\rangle$. Thus, each Grover iteration is a rotation in S. One then shows that each Grover iteration rotates towards the solutions

$|\beta\rangle$ by an angle of $\theta = \Theta(\sqrt{M/N})$. Since we need to rotate by a total angle of about $\pi/2$, it follows that we can obtain a good approximation to $|\beta\rangle$ after $O(\sqrt{N/M})$ Grover iterations. It is worth noting that — as described — this algorithm requires that we know the number of solutions M beforehand. Of course, this is not the case in real search problems. Fortunately, this assumption was eliminated in subsequent work [25].

4.6.3 Collision detection

As we shall see later in Chapter 12, an application of Grover's algorithm that has important implications for isomorphism testing is the *collision detection* problem. In this problem, we are given function $f : [N] \to [N]$ that is k-1 where $k \geq 2$. That is, for each y in the image of f, there are exactly k values $x \in [N]$ such that $f(x) = y$. The problem is to find a *collision*; this is a pair of distinct elements $x_1 \neq x_2 \in [N]$ such that $f(x_1) = f(x_2)$.

We can solve this problem by applying Grover's algorithm as in [26]. First, we choose a set A of size $\sqrt[3]{N/k}$ uniformly at random. We can test if A contains a collision in $O(\sqrt[3]{N/k})$ time by hashing. If it does not, there are $M = (k-1)\sqrt[3]{N/k} = \Theta(k^{2/3}N^{1/3})$ values $x \in [N]$ such that $x \notin A$ but $f(x) \in f[A]$. We can construct an oracle $\mathcal{O}_g$ that tests this condition using a circuit that implements binary search since A can be sorted beforehand in time $O(\sqrt[3]{N/k}\log(N/k))$. This takes $O(\log N)$ time plus a query to $\mathcal{O}_f$. By applying Grover's algorithm to $\mathcal{O}_g$, we obtain an algorithm that solves the collision problem in $O(\sqrt{N/M}) = O(\sqrt[3]{N/k})$ iterations. This translates to $\tilde{O}(\sqrt[3]{N/k})$ time and $O(\sqrt[3]{N/k})$ queries to $\mathcal{O}_f$.

4.7 The hidden subgroup problem

The *hidden subgroup problem*, is at the heart of most exponential quantum speedups (cf. [39, 115, 116, 38]) and is closely related to isomorphism testing. It is especially relevant to this work since many isomorphism testing problems can be formulated as hidden subgroup problems.

In this problem, we are given a generating set S, a group G, and a function $f : G \to A$ such that $f(x) = f(y)$ if and only if $xy^{-1} \in H$ for some unknown subgroup H and an arbitrary set A. As with Grover's algorithm, we are able to access f via an oracle $\mathcal{O}_f$ and our goal is to compute a generating set for the hidden subgroup H.

The hidden subgroup problem has important applications. For instance, integer factorization reduces to the hidden subgroup problem on the group $\mathbb{Z}$. This is the basis of Shor's [115] algorithm which can factor integers in polynomial time. Shor's algorithm [115] for solving the discrete logarithm problem is similarly based on a hidden subgroup problem over a finite Abelian group [64] (cf. [31]).

The problem of testing isomorphism of two graphs reduces to computing generators for the automorphism group of a graph. To see that this is an instance of the hidden subgroup problem over the symmetric group (cf. [41]), let X be a graph and for any $\pi \in \mathrm{Sym}(X)$, let X^π denote the graph obtained by relabeling the vertices of the graph according to π. We then define the function $f : \mathrm{Sym}(X) \to A$ by $f(\pi) = X^\pi$ where A is the set of all graphs on $|X|$ vertices. The subgroup hidden by f is then $\mathrm{Aut}(G)$, so solving this hidden subgroup problem is equivalent to computing a generating set for the automorphism group.

A similar reduction is possible for the group isomorphism problem. In fact, the reduction is the same as for graph isomorphism except that the concepts for graphs are replaced with the analogous concepts for groups. Group isomorphism reduces to the problem of computing the automorphism group of the group via a clever counting argument[4]. Let us suppose that we wish to compute a generating set for the automorphism group of G. For $\pi \in \mathrm{Sym}(G)$, let G^π be the group obtained by relabeling the elements of G according to the permutation π. Define the function $f : \mathrm{Sym}(G) \to A$ by $f(\pi) = G^\pi$ where A is the set of all groups of size $|G|$. Then solving this hidden subgroup problem is equivalent to computing generators for $\mathrm{Aut}(G)$. It is worth noting that the hidden subgroup problems for graph and group isomorphism are both non-Abelian.

Unfortunately, progress has been extremely limited on finding efficient quantum algorithms for non-Abelian hidden subgroup problems. Even for the *Dihedral group* of order

[4]James Wilson (personal communication)

$2N$ (which has a cyclic normal subgroup of order N), the best quantum algorithm known requires $2^{\tilde{O}(\sqrt{N})}$ time [98, 67, 66].

A more successful area of research has been the study of quantum algorithms for graph isomorphism based on the symmetric hidden subgroup problem. However, most of the results are negative. While it is known that there is a measurement that solves the symmetric hidden subgroup problem [41], there is no evidence that this measurement can be performed efficiently. A series of results [54, 87] have since shown that the measurement must satisfy a series of increasingly onerous requirements which make it increasingly unlikely that it can be implemented efficiently. Most recently, it was shown [88] that the methods used for the dihedral hidden subgroup problem cannot solve the symmetric hidden subgroup problem efficiently enough to outperform the best classical algorithm known for graph isomorphism [18, 16]. It is still conceivable that there could be an efficient quantum algorithm for the symmetric hidden subgroup problem. However, these results do strongly suggest that new ideas would be required and it is unclear what they would be.

Much more success has been achieved for Abelian groups. In this case, Kitaev [64] (cf. [31]) showed that the hidden subgroup problem can be solved in quantum polynomial time.

4.7.1 Shor's algorithm

Though the rest of this thesis does not depend on it, we present the the ideas behind Shor's algorithm for finite cyclic groups [115] in this section in order to give a flavor for the Fourier sampling methods used in quantum algorithms for the Abelian hidden subgroup problem. The algorithm for the Abelian hidden subgroup problem follows the same framework. The concepts are simply generalized from cyclic groups to Abelian groups using *representation theory* (the study of homomorphisms from groups to complex invertible matrices).

For the cyclic hidden subgroup problem, our group $G = \mathbb{Z}_N$ is cyclic. This implies that H is also cyclic since every subgroup of a cyclic group is also cyclic. Let r be the smallest nonnegative integer such that $H = \langle r \rangle$. We note that the subgroup H is then simply all nonnegative multiples of r that are less than N. Moreover, $H \cong \mathbb{Z}_{N/r}$. Our goal will be to compute r.

For this, we require the notion of a *quantum Fourier transform*. This is simply a unitary version of the usual discrete Fourier transform and is defined by the matrix

$$F = \left[\frac{1}{\sqrt{N}}\omega^{xy}\right]_{0 \le x,y < N}$$

where $\omega = e^{2\pi i/N}$. It is easy to verify that this matrix is unitary. It can also be implemented using $\mathsf{poly}(n)$ basic operations [115].

The first step is to prepare a uniform superposition

$$\frac{1}{\sqrt{N}} \sum_{x \in \mathbb{Z}_N} |x\rangle$$

of all group elements. Then we compute f by evaluating the oracle $\mathcal{O}_f$ in a second register

$$\frac{1}{\sqrt{N}} \sum_{x \in \mathbb{Z}_N} |x\rangle\, |f(x)\rangle$$

By measuring the second register in the computational basis, we obtain the coset state

$$|zH\rangle = \frac{1}{\sqrt{N/r}} \sum_{x=0}^{N/r-1} |xr + z\rangle\, |f(z)\rangle$$

for some $z \in \mathbb{Z}_N$. (Note that we are assuming that computations in the first register are performed modulo N so the value in this register is always at least 0 and less than N.) By discarding the second register and applying the quantum Fourier transform to the first register, we obtain

$$\frac{\sqrt{r}}{N} \sum_{x=0}^{N/r-1} \sum_{y=0}^{N-1} \omega^{(xr+z)y}\, |y\rangle = \frac{\sqrt{r}}{N} \sum_{y=0}^{N-1} \omega^{zy} \sum_{x=0}^{N/r-1} \omega^{xyr}\, |y\rangle$$

By considering the powers of ω on the unit circle in the complex plane, $\sum_{j=0}^{N/r-1} \omega^{jkr}$ is N/r if k is a multiple of N/r and 0 otherwise, so this simplifies to

$$\frac{\sqrt{r}}{N} \sum_{k=0}^{r-1} \omega^{zkN/r}\, |kN/r\rangle$$

so measuring in the computational basis yields a multiple of N/r. The group of all such multiples forms the subgroup $\langle N/r \rangle$ of $\mathbb{Z}_N$ which has order r. Thus, repeating this process $\log r \leq O(\log N)$ times, we obtain a generating set for $\langle N/r \rangle$ with high probability. We can then recover N/r by taking the greatest common divisor of the generating set. Finally, we obtain r by dividing N by N/r.

There is a classical reduction from factoring integers to order finding on the group $\mathbb{Z}_N^\times$. A modification (cf. [89]) of the above algorithm yields an algorithm for order finding on this group, which results in a quantum algorithm for factoring integers.

Part I

QUANTUM COMPUTING

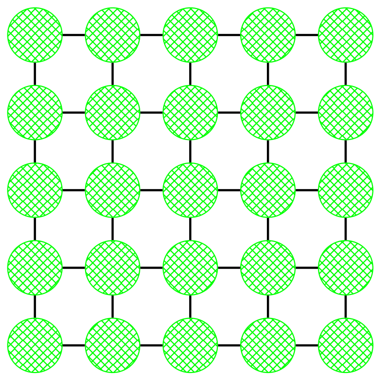
Chapter 5

2D QUANTUM CIRCUITS

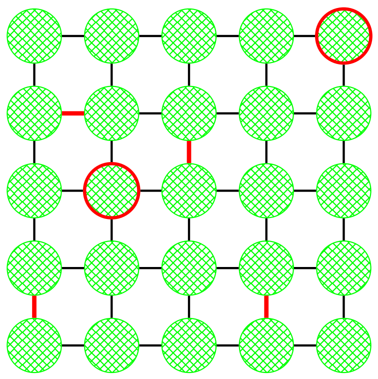
5.1 Introduction

As discussed in Chapters 1 and 2, quantum algorithms are typically formulated in an abstract model that allows interactions between arbitrary pairs of qubits. However, on a physical quantum computing device, the qubits are positioned in space and only neighboring qubits are allowed to interact. One common arrangement that is used for the qubits is a two-dimensional grid. Since it is usually possible for operations that act on disjoint sets of qubits to be performed simultaneously, many quantum computing technologies also offer a large amount of parallelism. These two considerations were the motivation for the kD *nearest-neighbor two-qubit concurrent* (kD NTC) quantum architecture [124] (cf. [32]), in which the qubits are arranged on the kD grid $\mathbb{Z}^k$ and operations on disjoint sets of qubits are allowed to be done in parallel. We show this grid along with an example set of operations that could be performed simultaneously in Figure 5.1a for the case where $k = 2$.

Another important aspect of a practical quantum computing architecture is that of a *classical controller*, which is a classical computer that decides which quantum operations should be performed at each point in the computation. The classical controller is allowed to make these decisions by means of a randomized polynomial-time computation that can depend on the original input to the problem, any intermediate measurement outcomes and the operations chosen at previous steps.



(a) Interactions in the 2D NTC architecture: the grid lines indicate the two-qubit interactions which can be performed



(b) An example of concurrent interactions in the 2D NTC architecture: the components connected by the thick red edges indicate concurrent interactions and the thick red circles indicate single-qubit interactions

A special case of models of quantum computation that allow a classical controller is *one-way quantum computing* [95], which performs computations via a series of measurements on quantum states. The idea of using a classical controller to determine which operations to apply at each step is also implicit in the pre- and post-processing stages of Shor's algorithm [115], and is often assumed for fault-tolerant quantum computation. Since quantum operations are far more expensive than classical operations, we are primarily concerned with the depth of the quantum circuit and do not count the operations performed by the classical controller as long as they take polynomial time.

In this chapter, we study both the *classical-controller* kD *NTC* (kD CCNTC) architecture — a classical controller model where interactions are restricted to a kD grid — as well as the *non-adaptive* kD *NTC*[1] (NANTC) architecture where no classical controller is used and the

[1]The original NTC architecture described by Van Meter and Itoh [124] is in fact NANTC; however, we

operations applied cannot depend on intermediate measurement outcomes. The CCNTC model ignores the cost of offline computations performed by the classical controller and assumes that there are no classical locality restrictions. Since quantum computing technology is much less developed than classical computing technology, the clock rates of quantum computers are much lower than those of their classical counterparts. This makes ignoring the cost of classical computations a realistic assumption. Because quantum computers are already forced to be parallel devices in order to perform operations fault tolerantly [2], the total runtime of a quantum circuit is proportional to the depth of the corresponding quantum circuit. The restriction that interactions are between neighbors on a kD grid comes from the underlying physical device: in most technologies, only qubits that are spatially close can interact.

Another related architecture that is useful to keep in mind for the purpose of comparison is the *classical-controller abstract concurrent* (CCAC) architecture. This model of quantum computation allows the use of a classical controller and but places no restrictions on which pairs of qubits can interact. In other words, all pairs of qubits are considered to be neighbors. This is the abstract architecture in which most quantum algorithms are formulated.

5.1.1 Definitions

Before stating the main results of this chapter, we formally define models of computation and the measures of complexity that are required. Recall from Chapter 4 that the one- and two-qubit operations that can be performed by the hardware are called the *basic operations.* We assume that the basic operations are a *universal gate set* so that any one- or two-qubit unitary can be constructed from the basic operations. We also assume that the basic operations include measurement in the computational basis.

It is useful to distinguish between physical and logical timesteps. During each *physical timestep*, we can perform any set of disjoint basic operations. During a *logical timestep*,

prefer NANTC to avoid confusion with CCNTC where a classical controller is used.

we allow any set of disjoint t-qubit operations to be performed. In this chapter, we take $t = O(k)$ and assume k is constant.

Definition 5.1.1 (NANTC)**.** *In the* kD NANTC model*, computation is performed by applying a sequence of sets of basic operations $S_1, \ldots, S_d$ to the* kD *grid of qubits. We require that the operations in the set S_i are disjoint and are either single-qubit operations or two-qubit operations between neighbors in the* kD *grid. The sequence of sets of operations must be randomized polynomial-time computable from the size n of the input.*

In the models where a classical controller is present, the classical controller is invoked after each physical timestep to determine which operations to apply at the next step.

Definition 5.1.2 (CCAC)**.** *Let M be a randomized polynomial-time Turing machine that takes the input x and the measurement outcomes from the first i physical timesteps and outputs a set $M_1, \ldots, M_\ell$ of disjoint basic operations to apply to the qubits at the $i+1^{\text{th}}$ physical timestep. If no more physical timesteps are to be performed, then M outputs the special symbol $\boxdot$. Computation in the* CCAC model *is performed at physical timestep i by using M to compute the set of operations to apply and then applying them to the qubits.*

The CCNTC model is similar except that it also requires that two-qubit operations are only performed between neighbors on the kD grid.

Definition 5.1.3 (CCNTC)**.** *Let M be a randomized polynomial-time Turing machine that takes the input x and the measurement outcomes from the first i physical timesteps and outputs a set $M_1, \ldots, M_\ell$ of disjoint basic operations to be applied to the* kD *grid of qubits at the $i+1^{\text{th}}$ physical timestep. We require that each M_i is either a single-qubit operation or a two-qubit operation between neighbors in the* kD *grid. If no more physical timesteps are to be performed, then M outputs the special symbol $\boxdot$. Computation in the* CCNTC model *is performed at physical timestep i by using M to compute the set of operations to apply and then applying them to the* kD *grid of qubits.*

In this chapter, the machine M from Definitions 5.1.2 and 5.1.3 will be deterministic except for the pre- and post-processing stages of Shor's algorithm.

For the NANTC model, a *quantum circuit* is the sequence of basic operations $M_1, \ldots, M_\ell$ to be applied to the kD grid of qubits. For the CCAC and CCNTC models, a *quantum circuit* is described by the machine M from Definitions 5.1.2 and 5.1.3. We now define three standard measures of cost in these models.

Definition 5.1.4. *The* depth *of a quantum circuit is*

(a) d for the NANTC model where $S_1, \ldots, S_d$ is the sequence of operations from Definition 5.1.1 for an input of size n and

(b) $\max_{x\in\{0,1\}^n} \max_r d_{x,r}$ for the CCAC and CCNTC models where $d_{x,r}$ is the number of physical timesteps it takes for the machine M from Definitions 5.1.2 and 5.1.3 to output $\boxdot$ when the input is x and the random seed is r. The first max is taken is over all possible inputs x of length n and the second is over all possible random seeds r.

We note that the depth only changes by a constant factor if we use logical timesteps instead of physical timesteps in the above definition. This is due to our assumption that any operation performed in a logical timestep acts on at most $O(k) = O(1)$ qubits.

Definition 5.1.5. *The* size *of a quantum circuit is*

(a) $\sum_i |S_i|$ for the NANTC model where $S_1, \ldots, S_d$ is the sequence of operations from Definition 5.1.1 for an input of size n and

(b) $\max_{x\in\{0,1\}^n} \max_r s_{x,r}$ for the CCAC and CCNTC models where $S_{x,r}$ is the total number of operations applied when the input is x and the random seed is r. The first max is taken over all possible inputs x of length n and the second is over all possible random seeds r.

In the next definition, we assume that the qubits are indexed by $\mathbb{N}$ for the CCAC model.

Definition 5.1.6. *The* width *of a quantum circuit is* *(a) the size of the smallest hypercube that contains all of qubits acted on by operations in the sets S_i for the NANTC model where $S_1, \ldots, S_d$ is the sequence of operations from Definition 5.1.1 for an input of size n,*

(b) $\max_{x \in \{0,1\}^n} |A_x|$ *for the CCAC model where A_x is the smallest subset of* $\mathbb{N}$ *such that every qubit acted on is contained in A_x for input x and all random seeds r and*

(c) $\max_{x \in \{0,1\}^n} |A_x|$ *for the CCNTC model where A_x is the smallest hypercube in* $\mathbb{Z}^k$ *such that every qubit acted on is contained in A_x for input x and all random seeds r*

Typically, the depth is the most important metric to optimize since it is proportional to the amount of time required to execute the quantum operations. The width is also important since the number of qubits is currently quite limited but the size is largely irrelevant. Moreover, if parallelism is properly exploited then we expect the size to be roughly the depth times the width.

5.1.2 Results

In this subsection, we state the main results of this chapter. Our first result allows the standard *classical controller abstract concurrent* (CCAC) architecture to be simulated in the kD CCNTC architecture with constant factor overhead in the depth. We accomplish this using a 2D CCNTC teleportation scheme that allows arbitrary interactions on disjoint sets of qubits to be performed in constant depth. (See Chapter 4 for the basic idea behind quantum teleportation.)

Theorem 5.1.7. *Suppose that C is a CCAC quantum circuit with depth d, size s and width n. Then C can be simulated in $O(d)$ depth, $O(sn)$ size and n^2 width in the* 2D *CCNTC model.*

This result justifies the standard assumption that non-local interactions can be performed efficiently. Simulating each of the d timesteps from the CCAC circuit in the 2D CCNTC model requires an $O(n)$ time classical computation; this can be reduced to $O(\log n)$ time if the classical controller is a parallel device or if it includes a simple classical circuit. Since the clock speeds of classical devices are currently much faster than those of quantum devices, this overhead is not likely to be significant.

Corollary 5.1.8. *Let $\mathcal{E}$ be a quantum operation on n qubits. Let d_1 and d_2 be the minimum depths*[2] *required to implement $\mathcal{E}$ with error at most ϵ using $\mathsf{poly}(n)$ size and $\mathsf{poly}(n)$ width in the CCAC and kD CCNTC models respectively where $k \geq 2$. Then $d_1 = \Theta(d_2)$.*

It is possible to implement Shor's algorithm [115] in constant depth in the CCAC model [27] which implies that it can also be implemented in constant depth in the 2D CCNTC model.

Corollary 5.1.9. *Shor's algorithm can be implemented in constant depth, polynomial size and polynomial width in the* 2D *CCNTC model.*

Since controlled-U operations and fanouts can also be performed in constant depth and polynomial width in the CCAC model [57, 27, 121], we also have the following corollary.

Corollary 5.1.10. *Controlled-U operations with n controls and fanouts with n targets can be implemented in constant depth, $\mathsf{poly}(n)$ size and $\mathsf{poly}(n)$ width in the* 2D *CCNTC model.*

Our main technical result allows any subset of qubits to be reordered in constant depth. Theorem 5.1.7 follows from this as a corollary.

Theorem 5.1.11. *Suppose that we have an $n \times n$ grid where all qubits except those in the first column are in the state $|0\rangle$. Let $T \subseteq \{0, \ldots, n-1\}$ and let $\pi : T \to \{0, \ldots, n-1\}$ be a $1-1$ map such that for all $j \in T$ with $\pi(j) = 0$, $[0, j-1] \subseteq T$. Set $m = |\{j \in T \mid \pi(j) \neq 0\}|$. Then we can move each qubit at $(0, j)$ to $(\pi(j), 0)$ for all $j \in T$ in $O(1)$ depth, $O(mn)$ size and $(m+1)n \leq n^2$ width in the* 2D *CCNTC model.*

Previous work showed that any operation in the *Clifford group* can be implemented in constant depth in the one-way model [97]. In particular, this implies that the reordering of

[2]Here, we assume that there is a minimum depth required to implement $\mathcal{E}$ in the CCAC model when the size and width are $\mathsf{poly}(n)$.

Theorem 5.1.11 can be performed in constant depth in the CCNTC model. However, this can require a large number of qubits, since in the one-way model, qubits are not reused once they are measured. Therefore, since measurements are used to perform all computations in the one-way model, the number of qubits required is comparable to the number of gates.

Upper bounds for the depth of quantum circuits when converting between various architectures with no classical controller were previously studied by Cheung, Maslov and Severini [30]. Their results imply that the CCAC model can be simulated in the kD CCNTC model with $O(\sqrt[k]{n})$ factor depth overhead, $O(n)$ size overhead and no width overhead. In contrast to our results, their techniques are based on applying swap gates to move the interacting qubits next to each other and do not perform any measurements.

Implementations of Shor's algorithm in the kD CCNTC model with various superconstant depths were previously known for $k = 1$ and $k = 2$. Fowler, Devitt and Hollenberg [45] showed a 1D CCNTC circuit for Shor's algorithm which requires $O(n^3)$ depth, $O(n^4)$ size and $O(n)$ width where n is the number of bits in the integer which is being factored. Maslov [78] showed that any stabilizer circuit can be implemented in linear depth in the 1D CCNTC model, from which the result of Fowler, Devitt and Hollenberg [45] can be recovered. Kutin [68] gave a more efficient 1D CCNTC circuit which uses $O(n^2)$ depth, $O(n^3)$ size and $O(n)$ width. For the 2D CCNTC model, Pham and Svore [92] showed an implementation of Shor's algorithm in polylogarithmic depth, polynomial size and polynomial width.

Our result that controlled-U operations can be performed in constant depth, polynomial width and polynomial size in the CCNTC model was previously known to hold in the CCAC model. This line of work was started by Moore [86] who showed that parity and fanout are equivalent and posed the question of whether fanout has constant-depth circuits. Høyer and Špalek [57] proved that if fanout has constant-depth circuits then controlled-U operations can also be implemented in constant depth with inverse polynomial error. Raussendorf, Browne and Briegel [96] showed that any Clifford operation can be performed in constant depth on a one-way quantum computer while Browne, Kashefi and Predrix [27] proved that one-way quantum computation is equivalent to unitary quantum circuits with fanout. Combined with the aforementioned result of Høyer and Špalek [57], this implies that constant depth adaptive circuits for fanout can be used to implement controlled-U operations with inverse polynomial error in constant depth in the CCAC model. Takahashi and Tani [121] reduced the size of this circuit by a polynomial and made it exact.

In many technologies, measurements are much more costly than unitary operations. For this reason, we also consider the non-adaptive kD NANTC model. Here, there is no classical controller and the operations applied depend only on the size of the input and not on intermediate measurement outcomes. Our result in this model is a characterization of the complexity of controlled-U operations and fanouts.

Theorem 5.1.12. *The depth required for controlled-U operations with n controls and fanouts with n targets in the kD NANTC model is $\Theta(\sqrt[k]{n})$. Moreover, this depth can be achieved with size $\Theta(n)$ and width $\Theta(n)$.*

If the clock speeds of the quantum computer and its classical controller are comparable, then operations implemented using Theorem 5.1.12 are significantly faster than those implemented using Corollary 5.1.10. For this reason, Theorem 5.1.12 may become a better option as quantum computing technology matures.

The layout of the rest of this chapter is as follows. In Section 5.2, we review quantum teleportation and describe teleportation chains. In Section 5.3, we describe our 2D teleportation scheme and show that it allows arbitrary interactions to be implemented in constant depth in the 2D CCNTC model. In Section 5.4, we show an algorithm that implements controlled-U operations and fanouts for the kD NANTC model in depth $O(\sqrt[k]{n})$. In Section 5.5, we describe how our techniques can be applied to obtain kD NANTC quantum circuits for fanout with depth $O(\sqrt[k]{n})$. In Section 5.6, we prove a matching lower bound for a class of operations that includes controlled-U operations and fanouts.

5.2 Quantum teleportation chains

As we shall see, teleportation is a useful primitive that allows non-local interactions to be performed in a constant-depth circuit in the kD CCNTC model.

We briefly recall the essentials of the quantum teleportation procedure [21]. For a more detailed description, see Chapter 4. Recall that in quantum teleportation, Alice has a state $|\psi\rangle^S$ that she wishes to send to Bob and Alice and Bob share a Bell pair $|\Phi_\ell\rangle^{AB}$. After performing a Bell measurement on the registers S and A, Alice sends the measurement outcome k to Bob. Bob is then able to recover the state $|\psi\rangle^A$ by applying the *Pauli operation* $\sigma_\ell \sigma_k$ to his register B. The point of this procedure is that Alice has succeeded in sending a quantum state to Bob using only entanglement and classical communication. No quantum communication is necessary.

Let us now consider how quantum teleportation chains can be used in the the 1D CCNTC model model to teleport qubits arbitrary distances in constant depth. The underlying idea is very similar to the "wires for qubits" used in one-way quantum computation [95] and work by Copsey et al. on quantum architecture [36] . Suppose that we have a qubit in the state $|\psi\rangle^S$ along with m Bell states $\left|\Phi_{\ell_j}\right\rangle^{A_jB_j}$. These are arranged on a line so that the overall state is $|\psi\rangle^S \bigotimes_{j=1}^m \left|\Phi_{\ell_j}\right\rangle^{A_jB_j}$. Our goal is to move qubit S to B_m. One way to do this is to first teleport S to B_1 by performing a Bell measurement on SA_1. We then store the measurement outcome k_1 but do not apply the Pauli operation that would allow us to recover the state $|\psi\rangle$. From now on, we refer to this Pauli operation as *the correcting Pauli operation.* At this point, the state of B_1 is $\sigma_{\ell_1}\sigma_{k_1}|\psi\rangle$. Continuing this process, we obtain the state $\bigotimes_{j=1}^m \left|\Phi_{k_j}\right\rangle \prod_{j=m}^1 \left(\sigma_{\ell_j}\sigma_{k_j}\right)|\psi\rangle^{B_m}$. Since $\prod_{j=m}^1 \left(\sigma_{\ell_j}\sigma_{k_j}\right)$ is just a Pauli operation, we obtain the state $\bigotimes_{j=1}^m \left|\Phi_{k_j}\right\rangle |\psi\rangle^{B_m}$ in a single quantum operation. The crucial point here is that all of the Bell measurements are performed on disjoint pairs of qubits so they can all be done in parallel. Thus, we can perform a non-local interaction of arbitrary distance in constant depth. It is important to note that this is not possible without a classical controller since otherwise there is no way to compute the correcting Pauli operation.

5.3 *Depth complexity in the* kD *CCNTC model*

In this section, we show that an arbitrary set of CCAC interactions corresponding to basic operations can be performed in constant depth in the 2D CCNTC model. We assume that there are n qubits on which the interactions are to be performed and store these in the first column of a 2D $n \times n$ CCNTC grid. The qubit at location (i, j) is denoted by $q_{i,j}$. Since we must handle interactions between qubits that are not neighbors, we may as well assume that the original n qubits are stored in the first column $q_{0,0}, \ldots, q_{0,n-1}$ of qubits. The remaining columns are used as ancillas to implement teleportation chains. We teleport each of the n qubits horizontally to the right so that interacting pairs are in adjacent columns. Since these teleportations are on disjoint sets of qubits, they can be performed in parallel as in [95, 97, 123]. A second set of vertical teleportation chains is then used to move all the qubits down to the first row. At this point, the interacting qubits are neighbors so the interactions may be implemented directly. We then perform the reverse teleportations to move the qubits back to their original positions.

5.3.1 *An example of arbitrary interactions in the* 2D *CCNTC model*

We show an example in Figure 5.2. The desired interactions are shown in Figure 5.2a. The layout of the data qubits in the 2D grid is shown in Figure 5.2b; the ancilla qubits are used to implement the teleportation chains and are initially set to $|0\rangle$. We start by horizontally teleporting the qubits that interact to adjacent columns in Figure 5.2c where the teleportation chains are denoted by the dotted red arrows. The red double arrow indicates a swap operation; this is just a less expensive way of achieving the same result when the qubits are neighbors. The next step is to vertically teleport the data qubits down to the first row as shown in Figure 5.2d. Finally, all interacting qubits are now neighbors so we perform the desired interactions in Figure 5.2e. The final reverse teleportations are not shown but can be obtained by reversing the arrows in Figures 5.2c and 5.2d.

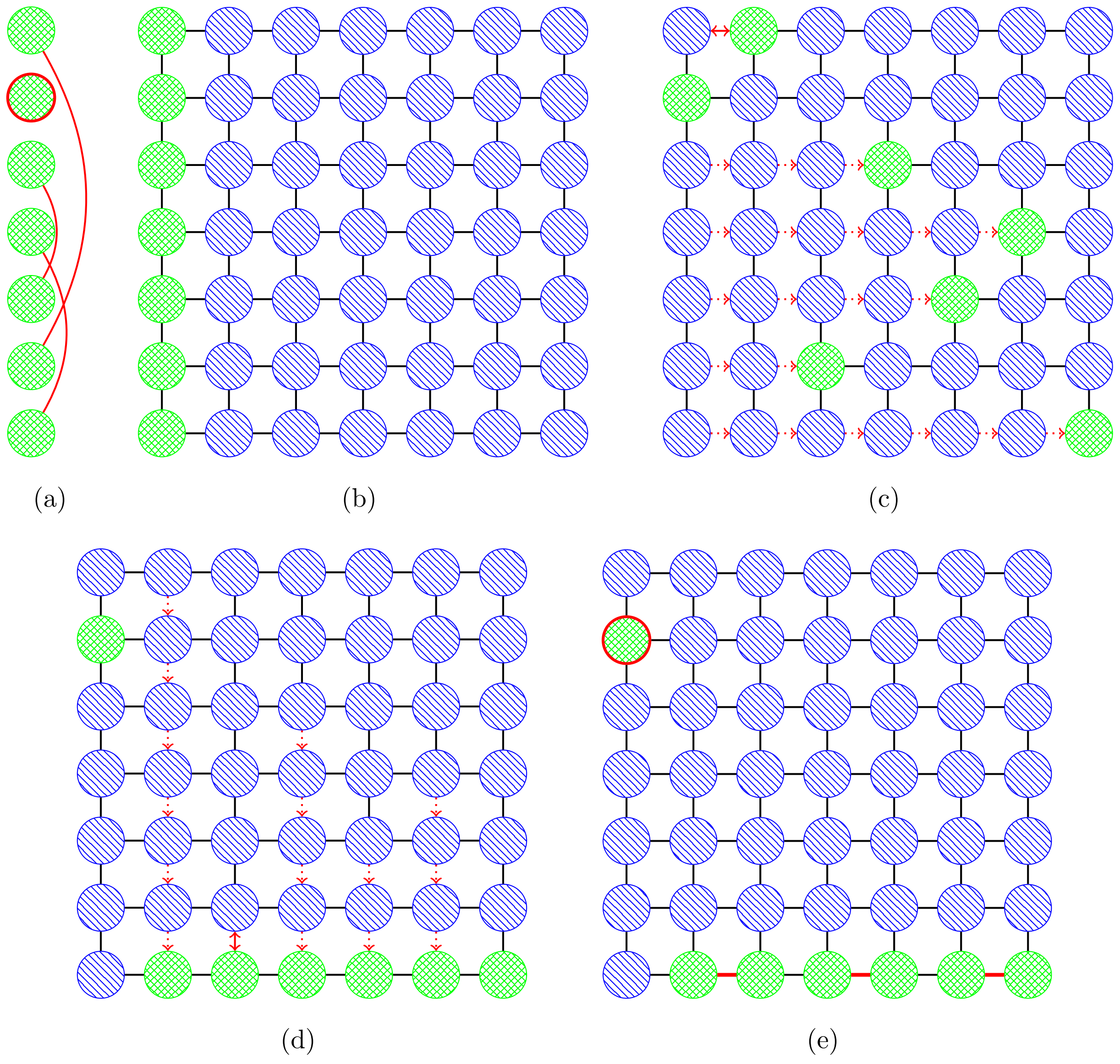


Figure 5.2: Performing an arbitrary set of interactions in the 2D CCNTC model. The qubits crosshatched green are the data qubits and the qubits shaded with diagonal downward blue lines are ancilla qubits

5.3.2 An algorithm for performing arbitrary interactions in the 2D *CCNTC model*

In order to define our algorithm, we first show how to perform an arbitrary reordering of the positions of the qubits in constant depth. We assume that there are n data qubits located in the first column of the $n \times n$ grid; the remaining qubits are in the state $|0\rangle$. We let $T \subseteq \{0, \ldots, n-1\}$ be a subset of row indexes on which a $1-1$ map $\pi : T \to \{0, \ldots, n-1\}$ is to be applied. This $1-1$ map describes where the qubits with row indexes in T are to be moved to on the x-axis. The reason that we specify T explicitly is because this allows us to only perform teleportations on qubits that have row indexes in T. If $|T| = o(n)$ then this can result in a circuit that has asymptotically smaller size. The reordering can be applied using Algorithm 1, which is based on the same technique as Figure 5.2. The notation teleport$(q_{i_1,j_1}, q_{i_2,j_2})$ where $i_1 = i_2$ or $j_1 = j_2$ means that a teleportation chain is applied to move the state of qubit at (i_1, j_1) along the line to (i_2, j_2).

Algorithm 1 The algorithm for performing an arbitrary reordering of a subset of the qubits in the 2D CCNTC model

```
Require: The $n$ data qubits are in the first column, $T \subseteq \{0, \ldots, n-1\}$ and $\pi : T \to \{0, \ldots, n-1\}$ is a $1-1$ map. For all $j \in T$ such that $\pi(j) = 0$, $\{k \in T^c \mid k < j\} = \emptyset$
Ensure: Each qubit at $(0, j)$ is moved to $(\pi(j), 0)$ for all $j \in T$
1: function REORDER($T$, $\pi$)
2:     for $j \in T$ do
3:         teleport$(q_{0,j}, q_{\pi(j),j})$
4:     end for
5:     for $j \in T$ do
6:         teleport$(q_{\pi(j),j}, q_{\pi(j),0})$
7:     end for
8: end function
```

Our main technical result follows immediately from Algorithm 1.

Theorem 5.1.11. *Suppose that we have an $n \times n$ grid where all qubits except those in the first column are in the state $|0\rangle$. Let $T \subseteq \{0, \ldots, n-1\}$ and let $\pi : T \to \{0, \ldots, n-1\}$ be a $1-1$ map such that for all $j \in T$ with $\pi(j) = 0$, $[0, j-1] \subseteq T$. Set $m = |\{j \in T \mid \pi(j) \neq 0\}|$. Then we can move each qubit at $(0, j)$ to $(\pi(j), 0)$ for all $j \in T$ in $O(1)$ depth, $O(mn)$ size and $(m+1)n \leq n^2$ width in the* 2D *CCNTC model.*

We note that the teleport operations in Algorithm 1 require an $O(n)$ time classical computation to determine the correcting Pauli matrix (see Section 5.2). Since this computation simply involves multiplying $O(n)$ Pauli matrices, it can be done more efficiently in $O(\log n)$ time by arranging the multiplications in a binary tree. The $O(\log n)$ runtime requires either that the classical controller is a parallel device or that it includes a special classical circuit for computing the correcting Pauli operation. Since classical operations are much faster than quantum operations on current devices, this overhead is unlikely to be a problem.

It is now straightforward to describe the algorithm for performing arbitrary interactions.We first note that an arbitrary set of interactions can be defined by disjoint one and two element subsets J_k of $\{0, \ldots, n-1\}$ and basic operations M_k where $1 \leq k \leq \ell$ and the values in J_k denote the qubits on which the operation M_k is to be applied. The pseudocode for performing arbitrary interactions in the 2D CCNTC model is shown in Algorithm 2.

The following theorem is a direct consequence of Algorithm 2.

Theorem 5.1.7. *Suppose that C is a CCAC quantum circuit with depth d, size s and width n. Then C can be simulated in $O(d)$ depth, $O(sn)$ size and n^2 width in the* 2D *CCNTC model.*

Recalling the discussion following Theorem 5.1.11, we see that each of the $O(d)$ timesteps requires an $O(n)$ time classical computation if the classical controller is a sequential device or a $O(\log n)$ time computation if it is parallel or includes a simple classical circuit. The time required to perform a single quantum operation is currently much longer than the time required to execute an instruction on a classical processor so this overhead is likely to be negligible.

Algorithm 2 The algorithm for performing arbitrary interactions in the 2D CCNTC model

```
Require: The $n$ inputs are in the first column, each $J_k$ is a disjoint one or two element
    subset of $\{0, \dots, n-1\}$, each $M_k$ is a basic operation and $|J_{k_1}| \leq |J_{k_2}|$ for $k_1 \leq k_2$
Ensure: The interactions specified by $J_k$ and $M_k$ are applied
 1: function INTERACT($J_1, \dots, J_\ell$, $M_1, \dots, M_\ell$)
 2:     $T \coloneqq ()$; $i \coloneqq 0$
 3:     for $k \coloneqq 1, \dots, \ell$ do
 4:         if $|J_k| = 1$ then
 5:             $i \coloneqq 1$
 6:         else
 7:             $\{j_1, j_2\} \coloneqq J_k$ where $j_1 < j_2$; $(\pi(j_1), \pi(j_2)) \coloneqq (i, i+1)$
 8:             Append the elements of $J_k$ to $T$
 9:             $i \coloneqq i + 2$
10:         end if
11:     end for
12:     Reorder$(T, \pi)$; $i \coloneqq 0$
13:     for $k \coloneqq 1, \dots, \ell$ do
14:         if $|J_k| = 1$ then
15:             $\{j\} \coloneqq J_k$
16:             Apply $M_k$ to $q_{0,j}$
17:             $i \coloneqq 1$
18:         else
19:             Apply $M_k$ to $q_{i,0}, q_{i+1,0}$
20:             $i \coloneqq i + 2$
21:         end if
22:     end for
23:     Perform the reverse teleportations to move the qubits back to their original positions
24: end function
```

The rest of our results for the kD CCNTC model follow from Theorem 5.1.7. Let $\mathcal{D}_n$ denote the set of all $n \times n$ density matrices. A general quantum operation is represented as a completely positive trace preserving (CPTP) map $\mathcal{E} : \mathcal{D}_n \to \mathcal{D}_n$. Obviously, any circuit in the 2D CCNTC model can also be applied when arbitrary interactions are allowed. The following corollary is immediate.

Corollary 5.1.8. *Let $\mathcal{E} : \mathcal{D}_n \to \mathcal{D}_n$ be a CPTP map and let $\epsilon \geq 0$. Let d_1 and d_2 be the minimum depths required to implement $\mathcal{E}$ with error at most ϵ in the CCAC and kD CCNTC models respectively where $k \geq 2$. Then $d_1 = \Theta(d_2)$.*

It is known that Shor's algorithm can be implemented in constant depth, polynomial size and polynomial width in the CCAC model [27] from which we obtain another corollary.

Corollary 5.1.9. *Shor's algorithm can be implemented in constant depth, polynomial size and polynomial width in the* 2D *CCNTC model.*

Because controlled-U operations and fanouts with unbounded numbers of control qubits or targets can be performed in constant depth, polynomial size and polynomial width in the CCAC model [57, 27, 121], we have the following result.

Corollary 5.1.10. *Controlled-U operations with n controls and fanouts with n targets can be implemented in constant depth,* $\mathsf{poly}(n)$ *size and* $\mathsf{poly}(n)$ *width in the* 2D *CCNTC model.*

5.4 Controlled operations in the kD NANTC model

In this section, we show how to control a single-qubit U operation by n controls using $O(\sqrt[k]{n})$ operations in the kD NANTC model. We start with an $m \times m$ grid; for reasons that will become clear later, we require that m is odd. The control qubits are placed such that they are not at adjacent grid points; the central 3×3 square has no controls except when $m = 3$. This is illustrated in Figures 5.3a, 5.4a, 5.5a and 5.6a for the cases where $m = 3$, $m = 5$, $m = 7$ and $m = 9$. Let c be the center of the grid which corresponds to the target qubit. The circuit works by considering each square ring in the grid with center c (i.e., a set of points in the grid that all have the same distance to the center under the ℓ_∞ norm). We start with the outermost such ring and propagate its control values into the next ring. At each such step, some of the control values are combined so that all the values can fit into the smaller ring. This continues until we reach a 3×3 ring at which point we apply a special sequence of operations to finish applying the controlled operation to the central qubit. We will show that each stage can be implemented in constant depth so the overall depth is $O(\sqrt{n})$.

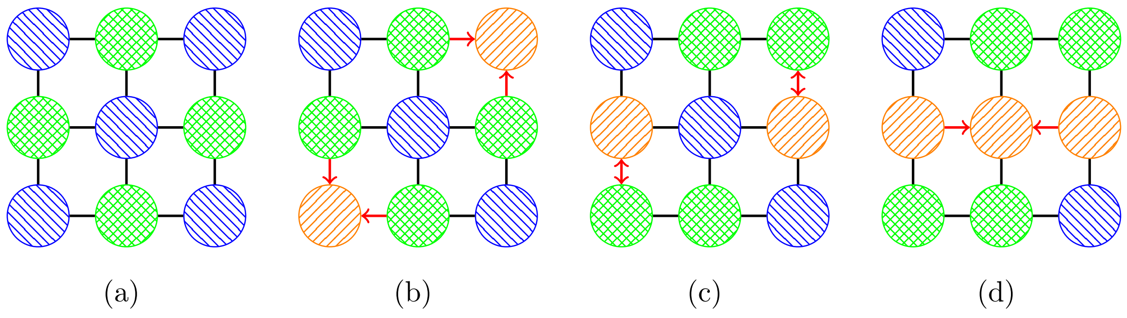


Figure 5.3: A controlled operation on a 3×3 grid. The qubits crosshatched green are the data qubits, the qubits shaded with diagonal upward orange lines are ancilla qubits which store intermediate data and the qubits shaded with diagonal downward blue lines are ancilla qubits which are currently unused.

5.4.1 The base case: the 3×3 *grid*

We now describe how this circuit works in greater detail. First, consider the case where $m = 3$. The grid starts as shown in Figure 5.3a; note that we do not force the central 3×3 square to be devoid of controls in this case since this is the entire grid. All ancilla qubits start in the state $|0\rangle$. We start by setting the lower left and upper right corner ancilla qubits to the ANDs of their neighboring controls as shown in Figure 5.3b. Both of these operations are disjoint, so this can be done in one logical timestep. The next step is to swap these two corner qubits with the vertical middle qubits so they can interact with the central target qubit; this is done in Figure 5.3c. Finally, we apply a U operation to the target qubit and control by the two middle qubits in Figure 5.3d.

At this point, the target qubit has the desired value; however, there are two other ancilla qubits in Figure 5.3d that must have their values uncomputed. This is done by applying the operations of Figures 5.3b–c in reverse order.

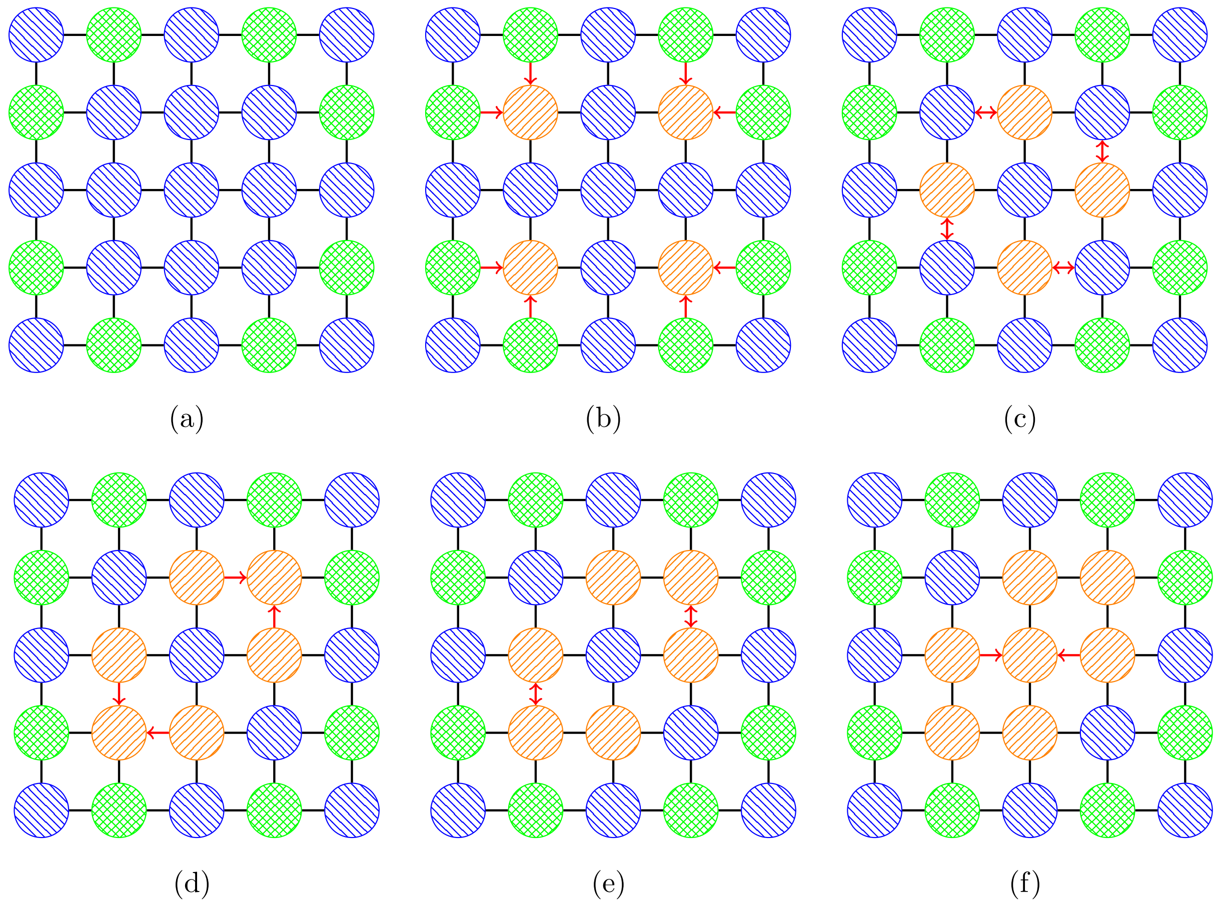


Figure 5.4: A controlled operation on a 5×5 grid. See Figure 5.3 for the meaning of the colors and shading used.

5.4.2 An example of the general case: the 5×5 *grid*

We now consider an example of the general case where $m = 5$ as shown in Figure 5.4a. The first step is to propagate the values of the outer ring inwards; since the inner ring is 3×3, there are no controls in the inner ring so this can be done as shown in Figure 5.4b. We then rotate the inner ring as in Figure 5.4c. At this point, the remaining operations to perform are the same as in the 3×3 case and are shown in Figures 5.4d–f. At this point the target qubit has the desired value so we uncompute the intermediate ancillas by applying the operations of Figures 5.4b–e in reverse order.

The same idea applies to an $m \times m$ grid except that when the inner rings have controls (i.e. for $m \geq 7$), the controls from the outer ring must be combined with those in the inner ring at the same time they are propagated inwards. See Section 5.7 for examples of the 7×7 and 9×9 cases.

5.4.3 An algorithm for controlled-U operations in $O(\sqrt{n})$ depth in the 2D NANTC model

We now present the algorithm used in Figures 5.3 – 5.6 for the general $m \times m$ grid. Consider an odd $m > 3$. We denote the coordinates of the qubits on this grid by (x, y) where $0 \leq x, y < m$. Let G be the set $\{0, \dots, m-1\}^2$ of all points on the grid and let $c = ((m-1)/2, (m-1)/2)$ be the central point. As discussed previously, the geometry induced by the ℓ_∞ norm is useful for reasoning about this grid. From now on, all distances in this subsection are understood to be with respect to the ℓ_∞ norm.

We will say that the k^{th} *ring* is the set of points that have distance $(m-1)/2 - k$ to c so the zeroth ring is outermost; we denote by $R_k = (r_0^k, \dots, r_{\ell_k}^k)$ the points of the k^{th} ring where r_0^k is the bottom left corner and the rest of the points are in clockwise order.

The ring R_k contains $4\left(\frac{m-1}{2} - k\right)$ controls so the entire grid has $n = 4\sum_{3<m-2k\leq m}\left(\frac{m-1}{2} - k\right) = (1/2)(m^2 - 9/2)$ controls for $m > 3$. In the case where $m = 3$, there are 4 controls. Thus, it is indeed the case that the depth is $O(\sqrt{n})$.

We denote by $q_{i,j}$ the value stored at the point (i, j) and assume the operation to apply to the target is U. The notation $\text{CU}(y, x_1, \dots, x_\ell)$ denotes applying a controlled-U operation to qubit y conditional on $x_1, \dots, x_\ell$. To apply a swap operation to qubits x and y, we write $\text{swap}(x, y)$. The pseudocode for the main algorithm is shown in Algorithm 3; the auxiliary functions are shown in Algorithms 4 and 5.

Algorithm 3 The algorithm for implementing a controlled-U operation on an $m \times m$ grid

```
Require: $m$ is odd
Ensure: A controlled-$U$ operation is applied to the target
 1: function CONTROL($m$)
 2:     $k := 0$
 3:     while $m - 2k \geq 3$ do
 4:         Control-Stage($k$)
 5:         $k := k + 1$
 6:     end while
 7:     Uncompute the intermediate ancillas by repeating all operations except for the final
    CU operation in reverse order
 8: end function
 9: function CONTROL-STAGE($k$)                    ▷ $k$ is the depth of the recursive call
10:     if $k > 0$ then
11:         Control-Clockwise($k$)
12:         Rotate($k$)
13:     end if
14:     if $m - 2k = 3$ then                        ▷ In this case, we have a $3 \times 3$ grid
15:         $q_{k,k} \leftarrow q_{k,k} \oplus q_{k,k+1} \wedge q_{k+1,k}$
16:         $q_{k+2,k+2} \leftarrow q_{k+2,k+2} \oplus q_{k+1,k+2} \wedge q_{k+2,k+1}$
17:         swap($q_{k,k}, q_{k,k+1}$)
18:         swap($q_{k+2,k+1}, q_{k+2,k+2}$)
19:         CU($q_{k+1,k+1}, q_{k,k+1}, q_{k+2,k+1}$)
20:     end if
21: end function
```

Algorithm 4 The CONTROL-CLOCKWISE operation

```
1: function CONTROL-CLOCKWISE($k$)
2:     $C = ((k,k),(k,m-k-1),(m-k-1,m-k-1),(m-k-1,k))$ ▷ The corners of $R_k$
3:     $D = ((0,1),(1,0),(0,-1),(-1,0))$ ▷ The directions to follow between the corners of $R_k$
4:     for $i \coloneqq 0,\dots,3$ do
5:         $i_- \coloneqq i-1 \mod 4$
6:         $i_+ \coloneqq i+1 \mod 4$
7:         $q_{C_i} \leftarrow q_{C_i} \oplus q_{C_i-D_i} \wedge q_{C_i+D_{i_-}}$ ▷ Compute the corner ancilla
8:         Let $s_0,\dots,s_{\ell_k/4}$ be the points in $R_k$ from $C_i$ to $C_{i_+}$ excluding $C_{i_+}$
9:         $j \coloneqq 2$
10:        while $j < \ell_k/4-1$ do ▷ Store the AND of two values in each ancilla in $L$ except for the last
11:            $q_{L_j} \leftarrow q_{L_j} \oplus q_{L_j-D_i} \wedge q_{L_j+D_{i_-}}$
12:            $j \coloneqq j+2$
13:        end while
14:        $p \coloneqq L_{\ell_k/4-1}$
15:        if $m-2k > 3$ then ▷ For the last ancilla, use three controls unless we have a $5\times 5$ grid
16:            $q_p \leftarrow q_p \wedge q_{p-D_i} \wedge q_{p+D_{i_-}} \wedge q_{p+D_i}$
17:        else
18:            $q_p \leftarrow q_p \wedge q_{p-D_i} \wedge q_{p+D_{i_-}}$
19:        end if
20:    end for
21: end function
```

Algorithm 5 The ROTATE operations

```
1: function ROTATE($k$)
2:     $i := 1$
3:     while $i \leq \ell_k$ do
4:         $i_+ := i + 1 \mod \ell_k$
5:         swap($q_{r^k_i}, q_{r^k_{i_+}}$)
6:         $i := i + 2$
7:     end while
8: end function
```

The following theorem is an immediate consequence of Algorithm 3.

Theorem 5.4.1. *Controlled-U operations with n controls have depth $O(\sqrt{n})$, size $O(n)$ and width $O(n)$ in the 2D NANTC model.*

5.4.4 Generalization to the kD NANTC model

In this section, we discuss how the circuit can be generalized to k dimensions. The algorithm works in the same way except the ring R_k is replaced by the grid points on the surface of the hypercube formed by the points at ℓ_∞ distance $(m-1)/2 - k$ from the center c of the grid. We proceed as before and propagate the controls on R_k into R_{k+1} until we obtain a grid of width 3. Since the number of controls on a kD grid of length m is $O(m^k)$, we obtain a circuit of depth $O(\sqrt[k]{n})$ for implementing a controlled-U operation with n controls. The constant depends on k, but we assumed that k is constant in Section 5.1. From this, we obtain the following result.

Theorem 5.4.2. *Controlled-U operations with n controls have depth $O(\sqrt[k]{n})$, size $O(n)$ and width $O(n)$ in the kD NANTC model.*

5.5 Fanout operations

In this section, we describe quantum circuits for fanout. In this case, we have a single control qubit and our goal is to XOR it into each of the target qubits. The construction of fanout circuits is adapted from Algorithm 3; the circuits are the same except that the qubit that was the target becomes the control qubit and qubits that were the controls become the targets. Let n be the number of targets. In the case of the circuit of Section 5.4, we simply apply all operations in reverse order and replace each Toffoli gate $y \leftarrow y \oplus x_1 \wedge \ldots \wedge x_n$ with a fanout operation $x_j \leftarrow x_j \oplus y$ for all $1 \leq j \leq n$. This yields a kD NANTC fanout circuit of depth $O(\sqrt[k]{n})$. We have shown the following.

Theorem 5.5.1. *fanouts to n targets have depth $O(\sqrt[k]{n})$, size $O(n)$ and width $O(n)$ in the kD NANTC model.*

5.6 Optimality

In this section, we prove that the depth, size and width of the circuits generated by Algorithm 3 (and its kD generalization) are optimal for the NANTC model. A similar lower bound for addition is discussed in [33]. These lower bounds hold regardless of where the controls and target qubits are located on the kD grid. They also hold for a more general class of operations that contains the controlled-U operations and fanouts.

Since each qubit is acted on by a constant number of operations in Algorithm 3, the size of the circuit is $O(n)$. This is clearly optimal since any circuit that implements a controlled operation must act on each of the controls.

Theorem 5.6.1. *Any NANTC quantum circuit that implements a non-trivial controlled-U operation with n controls has size $\Omega(n)$.*

The *trace norm* of a density matrix ρ (denoted $\|\rho\|_{\mathrm{tr}}$) is equal to $(1/2)\operatorname{tr}|\rho|$ (the $(1/2)$ factor ensures that $\|\rho - \sigma\|_1$ is the probability of distinguishing ρ and σ with the best possible measurement). Consider a general quantum operation $\mathcal{E} : \mathcal{D}_n \to \mathcal{D}_n$ represented as a CPTP map. We will use an operator version of the trace norm defined by $\|\mathcal{E}\|_{\mathrm{tr}} = \sup_{\rho \in \mathcal{D}} \|\mathcal{E}(\rho)\|_1$; if $\mathcal{E}_1$ and $\mathcal{E}_2$ are two CPTP maps then $\|\mathcal{E}_1 - \mathcal{E}_2\|_{\mathrm{tr}}$ is the probability of distinguishing between them on the worst possible input. Thus, it is a measure of how much these operations differ. We will also make use of the partial trace. If x is a qubit, then we will denote the partial trace over all qubits except x by $\mathrm{tr}_{\neg x} = \mathrm{tr}_{\mathbb{Z}^k \setminus \{x\}}$.

Controlled-U operations are special case of a more general class of operations.

Definition 5.6.2. *Let $\mathcal{E} : \mathcal{D}_n \to \mathcal{D}_n$ be a CPTP map. We say that $\mathcal{E}$ is* ϵ-input sensitive *if there exists a qubit y such that for $\Omega(n)$ qubits x, there exists a CPTP map $\mathcal{F} : \mathcal{D}_n \to \mathcal{D}_n$ acting only on x such that $\|\mathrm{tr}_{\neg y}(\mathcal{E}\mathcal{F} - \mathcal{E})\|_{\mathrm{tr}} \geq \epsilon$.*

Intuitively, an ϵ-input sensitive operation is a generalization of a Toffoli gate where modifying some input qubit x yields a different value on the output with probability ϵ. Similarly, we can define ϵ-output sensitive operations which are generalizations of fanout.

Definition 5.6.3. *Let $\mathcal{E} : \mathcal{D}_n \to \mathcal{D}_n$ be a CPTP map. We say that $\mathcal{E}$ is* ϵ-output sensitive *if there exists a qubit x such that for $\Omega(n)$ qubits y, there exists a CPTP map $\mathcal{F} : \mathcal{D}_n \to \mathcal{D}_n$ acting only on x such that $\|\mathrm{tr}_{\neg y}(\mathcal{E}\mathcal{F} - \mathcal{E})\|_{\mathrm{tr}} \geq \epsilon$.*

We say that $\mathcal{E}$ is ϵ-*sensitive* if it is ϵ-input or ϵ-output sensitive. A family $\{\mathcal{E} : \mathcal{D}_n \to \mathcal{D}_n\}$ of CPTP maps is ϵ-sensitive if every $\mathcal{E}_n$ is ϵ-sensitive. Our lower bounds will apply to all families of ϵ-sensitive operations. All proofs will be for the case of ϵ-input sensitive operations but the argument of ϵ-output sensitive operations is all but identical.

Theorem 5.6.4. *Let $\{\mathcal{E}_n : \mathcal{D}_n \to \mathcal{D}_n\}$ be a family of ϵ-sensitive operations. Then any family of kD NANTC circuits $\{C_n\}$ such that $\|\mathcal{E}_n - C_n\|_{\mathrm{tr}} < \epsilon/2$ for all n has size $\Omega(n)$.*

Proof. Suppose that C_n has size $o(n)$. Assume $\mathcal{E}_n$ is ϵ-input sensitive and choose a qubit y as in definition Definition 5.6.2 (the case where it is ϵ-output sensitive is very similar). There are $\Omega(n)$ qubits x such that there exists a CPTP map $\mathcal{F} : \mathcal{D}_n \to \mathcal{D}_n$ acting only on x such that $\|\mathrm{tr}_{\neg y}(\mathcal{E}_n\mathcal{F} - \mathcal{E}_n)\|_{\mathrm{tr}} \geq \epsilon$. For large n, there is such an x which is not acted on by C_n.

Then $\operatorname{tr}_{\neg y} C_n\mathcal{F} = \operatorname{tr}_{\neg y} C_n$. Now

$$\begin{aligned}\left\|\operatorname{tr}_{\neg y}(C_n - \mathcal{E}_n)\right\|_{\mathrm{tr}} &= \left\|\operatorname{tr}_{\neg y}(C_n\mathcal{F} - \mathcal{E}_n)\right\|_{\mathrm{tr}} &(5.1)\\ &\geq \left|\left\|\operatorname{tr}_{\neg y}(C_n\mathcal{F} - \mathcal{E}_n\mathcal{F})\right\|_{\mathrm{tr}} - \left\|\operatorname{tr}_{\neg y}(\mathcal{E}_n\mathcal{F} - \mathcal{E}_n)\right\|_{\mathrm{tr}}\right| &(5.2)\\ &> \epsilon/2 &(5.3)\end{aligned}$$

which is a contradiction. □

We call a controlled-U operation *non-trivial* if $U \neq I$. It is easy to prove the following.

Lemma 5.6.5. *Non-trivial controlled-U operations and fanouts are 1-sensitive.*

We now obtain a corollary of Theorem 5.6.4 of which Theorem 5.6.1 is a special case.

Corollary 5.6.6. *Let $\{\mathcal{E}_n : \mathcal{D}_n \to \mathcal{D}_n\}$ denote a family of controlled-U operations or fanouts. Any family of kD NANTC circuits $\{C_n\}$ such that $\|C_n - \mathcal{E}_n\|_{\mathrm{tr}} < 1/2$ has size $\Omega(n)$.*

This shows that the circuits generated by Algorithm 3 (and its kD generalization) have optimal size. Next, we will show that ϵ-sensitive kD NTC circuits have depth $\Omega(\sqrt[k]{n})$. For this we require the following easy lemma.

Lemma 5.6.7. *For any subset $S \subseteq \mathbb{Z}^k$ and any $x \in \mathbb{Z}^k$, there exists a subset $T \subseteq S$ of size $\Omega(|S|)$ such that for all $y \in T$, $\|x - y\|_1 = \Omega(\sqrt[k]{|S|})$.*

We are now ready to prove our depth lower bound.

Theorem 5.6.8. *Let $\{\mathcal{E}_n : \mathcal{D}_n \to \mathcal{D}_n\}$ be a family of ϵ-sensitive operations. Then any family of kD NANTC circuits $\{C_n\}$ such that $\|\mathcal{E}_n - C_n\|_{\mathrm{tr}} < \epsilon/2$ for all n has depth $\Omega(\sqrt[k]{n})$.*

Proof. Suppose $\{C_n\}$ has depth $t = o(\sqrt[k]{n})$. Assume that $\mathcal{E}_n$ is ϵ-input sensitive (the case where it is ϵ-output sensitive is very similar) and choose a qubit y as in Definition 5.6.2. There is a set S of $\Omega(n)$ qubits such that for each $x \in S$, there exists a CPTP map $\mathcal{F} : \mathcal{D}_n \to \mathcal{D}_n$ acting only on x with $\|\operatorname{tr}_{\neg y}(\mathcal{E}_n\mathcal{F} - \mathcal{E}_n)\|_{\mathrm{tr}} \geq \epsilon$. Let $c > 0$ be the hidden constant in the expression $\Omega(\sqrt[k]{|S|})$ from Lemma 5.6.7. For sufficiently large n, the depth of C_n is strictly less than $c\sqrt[k]{n}$. Let G_i be the set of disjoint one- and two-qubit operations that are performed at timestep $1 \leq i \leq t$ in C_n. For an operation $M \in G_i$, let us say that M is *active* if

(a) M acts non-trivially on y or

(b) there is an operation $M' \in G_j$ with $i < j \leq t$ such that M' is active and M and M' act non-trivially on a common qubit

Let us say that a qubit x *influences* y if there exists an active operation $M \in G_i$ that acts non-trivially on x. Suppose x influences y after t timesteps. Because all operations act on pairs of adjacent qubits, the ℓ_1 distance between x and y is at most t. By Lemma 5.6.7, there exists a subset T of S of size $\Omega(n)$ such that $\|x - y\|_1 \geq c\sqrt[k]{n}$ for all $x \in T$. Because $t < c\sqrt[k]{n}$, x does not influence y for $x \in T$. Let us fix some $x \in T$. Choosing a $\mathcal{F}$ acting only on x as in Definition 5.6.2, we have

$$\begin{aligned} \|\mathrm{tr}_{\neg y}(C_n - \mathcal{E}_n)\|_{\mathrm{tr}} &= \|\mathrm{tr}_{\neg y}(\mathcal{F}C_n - \mathcal{E}_n)\|_{\mathrm{tr}} && (5.4)\\ &\geq \left| \|\mathrm{tr}_{\neg y}(C_n\mathcal{F} - \mathcal{E}_n\mathcal{F})\|_{\mathrm{tr}} - \|\mathrm{tr}_{\neg y}(\mathcal{E}_n\mathcal{F} - \mathcal{E}_n)\|_{\mathrm{tr}} \right| && (5.5)\\ &> \epsilon/2 && (5.6) \end{aligned}$$

which is a contradiction. □

By Lemma 5.6.5, we obtain the following corollary.

Corollary 5.6.9. *Let $\{\mathcal{E}_n : \mathcal{D}_n \to \mathcal{D}_n\}$ denote a family of controlled-U operations or fanouts. Any family of kD NANTC circuits $\{C_n\}$ such that $\|C_n - \mathcal{E}_n\|_{\mathrm{tr}} < 1/2$ has depth $\Omega(\sqrt[k]{n})$.*

From Theorems 5.4.2 and 5.5.1 and Corollaries 5.6.6 and 5.6.9, we conclude that the circuits generated by Algorithm 3 and its kD generalization are optimal in their depth, size and width.

Theorem 5.1.12. *The depth required for controlled-U operations with n controls and fanouts with n targets in the kD NANTC model is $\Theta(\sqrt[k]{n})$. Moreover, this depth can be achieved with size $\Theta(n)$ and width $\Theta(n)$.*

5.7 More Examples

We now present the implementation of controlled-U operations in 7×7 and 9×9 2D NANTC grids. This is shown for $m = 7$ in Figure 5.5. As before, it is necessary to uncompute the intermediate ancillas by applying the operations of Figures 5.5b–g in reverse order. We also show the case where $m = 9$ in Figure 5.6. In this case, we apply the operations of Figures 5.6b–i in reverse order to uncompute the intermediate ancillas.

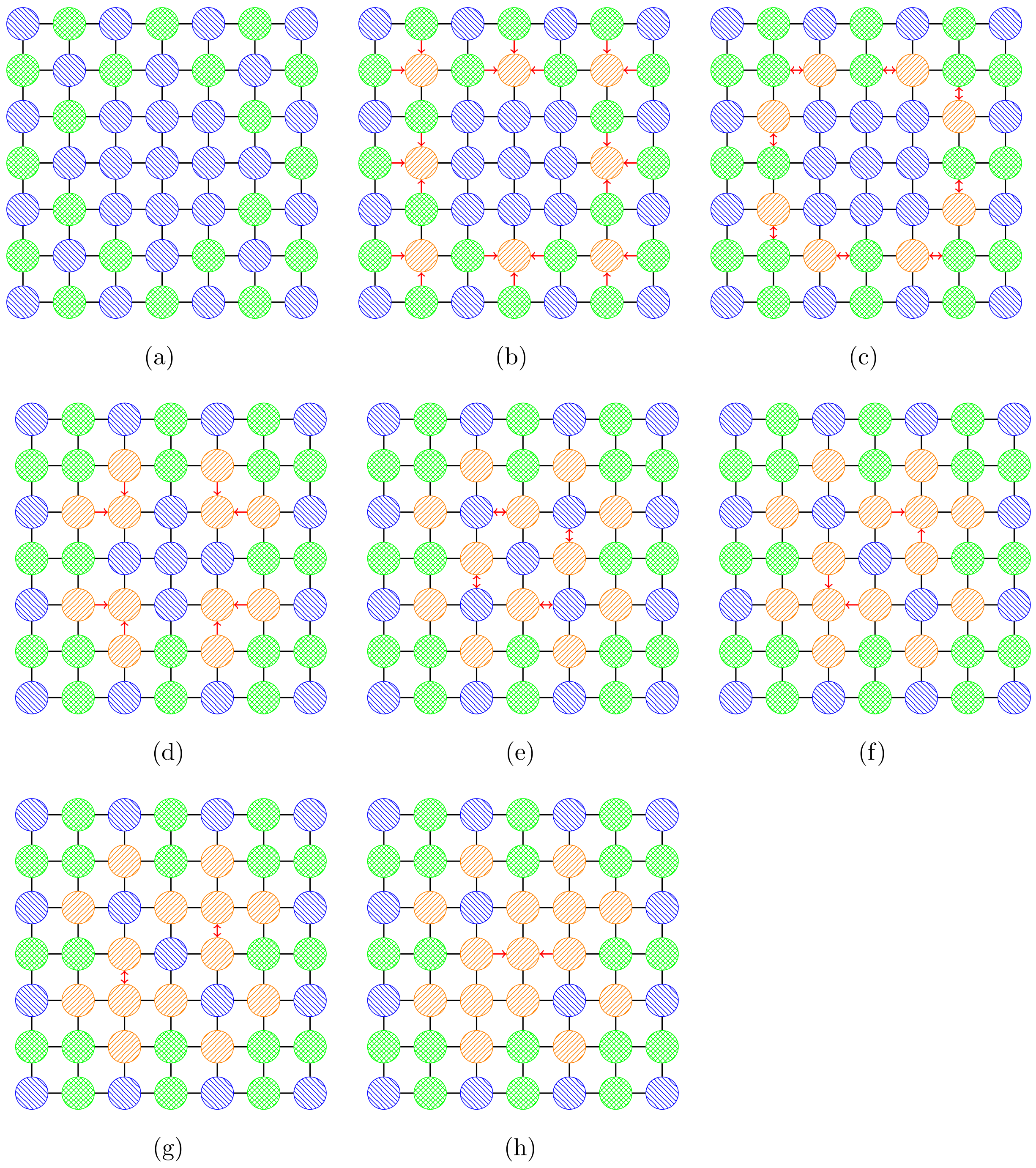
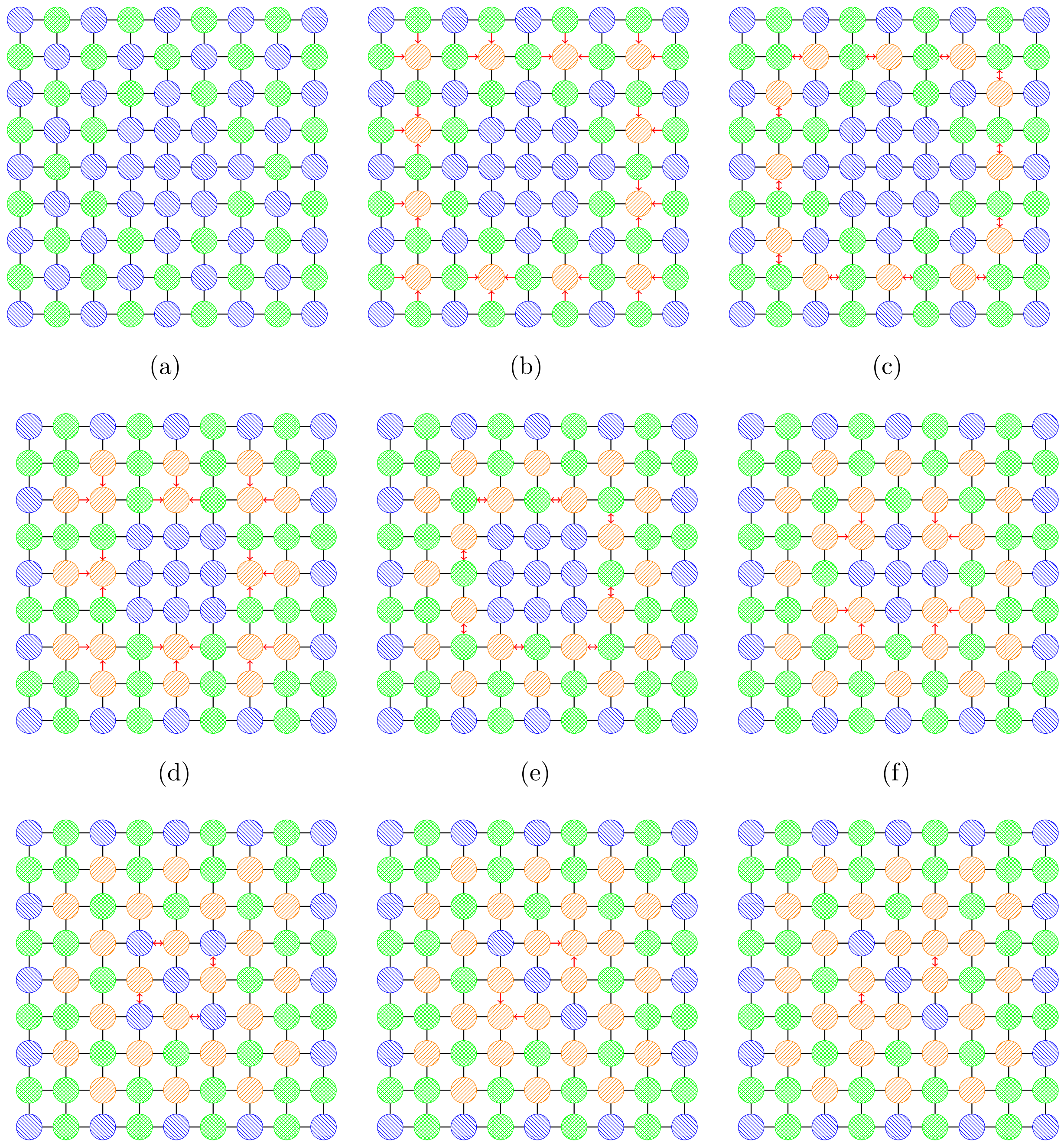


Figure 5.5: A controlled operation on a 7×7 grid. See Figure 5.3 for the meaning of the colors and shadings used.

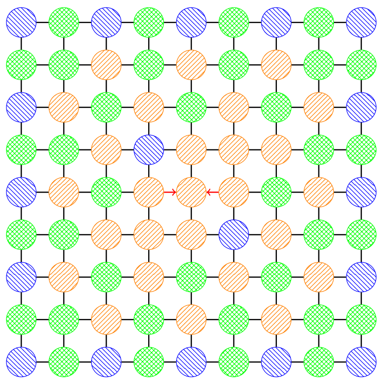


(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 5.6: A controlled operation on a 9 × 9 grid. See Figure 5.3 for the meaning of the colors and shadings used.



(j)

Figure 5.6: A controlled operation on a 9×9 grid (continued).

5.8 Conclusion

In this chapter, we saw that quantum teleportation can be used to implement quantum circuits with arbitrary interactions in a 2D architecture where only operations between neighboring qubits are allowed with only a constant factor increase in the depth. However, this comes at the cost of a quadratic increase in the width of the quantum circuit. Interestingly, we can show that this quadratic increase in width is necessary in some cases, so that the width requirements are essentially optimal. This result, along with methods that reduce the number of qubits required in certain cases, will be the subject of a future work.

Chapter 6
USELESSNESS AND INFINITY-VS-ONE SEPARATIONS

6.1 Introduction

Oracles are an important conceptual framework for understanding quantum speedups. They may represent subroutines whose code we cannot usefully examine, or an unknown physical system whose properties we would like to estimate. When used by a quantum computer, the most general form of an oracle is a possibly noisy quantum operation that can be applied to an n-qubit input. However, oracles this general have no obvious classical analogue, which makes it difficult to compare the ability of classical and quantum computers to efficiently interrogate oracles. This was the original motivation of the *standard oracle model*, in which f is a function from $[N] = \{1, \ldots, N\}$ to $\{0, 1\}$, and the oracle $\mathcal{O}_f$ acts for a classical computer by mapping x, y to $x, y \oplus f(x)$, and for a quantum computer as a unitary that maps $|x, y\rangle$ to $|x, y \oplus f(x)\rangle$. One way to justify the standard oracle model is that if there is a (not necessarily reversible) classical circuit computing f, then $\mathcal{O}_f$ can be simulated by computing f, XORing the answer onto the target, and uncomputing f.

In this chapter, we consider other forms of oracles that are more general than the standard oracle model, but nevertheless permit comparison between classical and quantum query complexities. Meyer and Pommersheim [83] generalized the standard model by letting A be a deterministic classical algorithm. The oracle then maps each basis state $|x, y\rangle$ to $\left|x, \pi_{A(x)}(y)\right\rangle$ where each $\pi_{A(x)}$ is a permutation. We further generalize the model by replacing A with a randomized classical algorithm. The random coins used by A are internal to the oracle and cannot be accessed externally. We call this concept an *oracle with internal randomness.* Note that even if A takes no input, the oracle can still be interesting since it may apply different permutations depending on its internal coin flips.

Oracles with internal randomness correspond naturally to the situation in which a (quantum or classical) computer seeks to determine properties of a device that acts in a noisy or otherwise non-deterministic manner. One simple example is an oracle that "misfires", i.e. when queried, the oracle does nothing with probability p and responds according to the standard oracle model with probability $1 - p$. This model was considered in [99], which found, somewhat surprisingly, that the square-root advantage of Grover search disappears (i.e. there is an $\Omega(N)$ quantum query lower bound for computing the OR function) for any constant $p > 0$.

The rest of this chapter is divided into two parts. First, we explore various examples of oracles with internal randomness that demonstrate the power of the model. We will see that in some cases (e.g. Theorems 6.3.1 and 6.3.2), this can even result in problems solvable with one quantum query that are completely unsolvable using classical queries.

In the second part, we consider the question of when oracle problems can be solved with any nontrivial advantage; i.e. a probability of success better than could be obtained by simply guessing the answer according to the prior distribution. For an example of when such advantage is *not* possible, consider the parity function on N bits. If these bits are drawn from the uniform distribution, then any classical algorithm making $\leq N - 1$ queries—or any quantum algorithm making $\leq \frac{N}{2} - 1$ queries—will not be able to guess the parity with any nontrivial advantage. In Section 6.4, we consider the problem of when some number of queries are *useless* for solving an oracle problem. Informally, our main result is roughly that k quantum queries are useless if and only if $2k$ classical queries are useless (this is formalized in Theorem 6.4.7). However, a subtlety arises in our theorem when oracles have internal randomness, in that the $2k$ classical queries need to be considered as k pairs, each of which uses a separate sample from the internal randomness of the oracle.

In the unbounded-error query complexity regime, similar results were obtained 15 years ago by Farhi, Goldstone, Gutman and Sipser [42] for the case of the parity function. More recently, Montanaro, Nishimura and Raymond [85] proved a similar result for any binary function f, using techniques that do not readily generalize to non-binary f. One direction of the special case of our result for *deterministic permutative oracles* was proved by Meyer and Pommersheim [82]. Our proof is arguably simpler and more operational. We introduce an analogue of gate teleportation [50] for oracles by showing that oracles can be (a) encoded into states analogous to Choi-Jamiołkowski states, and (b) retrieved from those states with an exponentially small, but heralded, success probability (i.e. the procedure outputs a flag that tells us whether it succeeded or failed). We expect that this characterization will be useful for future study of query complexity in the regime where any nonzero advantage is sought.

Finally, our encoding can be used to construct infinity-vs-one separations from *any* separation between classical and quantum uselessness (see Theorems 6.5.1 and 6.5.2).

6.2 Conventions for oracles

Throughout this chapter, we deal with oracles that have either one or two inputs. Single-input oracles are those which simply apply an operation to the input. When an oracle has two inputs, we call the first of these the *control* and the second the *target* in analogy with controlled operations. The control is never modified by the oracle but the target is transformed depending on the state of the control.

6.3 Examples of infinity-vs-one query-complexity separations

In this section, we discuss problems that can be solved using a single quantum query but cannot be solved classically even with an unlimited number of queries. Such a separation is far stronger even than exponential separations. To achieve such infinity-vs-one separations, it is necessary (but not sufficient) for the oracle to have internal randomness, since otherwise one could simulate the quantum algorithm classically with exponential overhead. The key point is that internal randomness effectively causes a different oracle to be used for each query so such a simulation is not possible in this case.

6.3.1 Distinguishing involutions with no fixed points from cycles of length at least three

Our first example of an infinity-vs-one separation is given by the problem of distinguishing involutions from cycles of length at least three. Define

$$\mathrm{INV} = \left\{\pi \in S_N \;\middle|\; \pi^2 = 1 \text{ and } \pi x \neq x \text{ for all } x \in [N]\right\}$$

This is the set of involutions in S_N with no fixed points. Let

$$\mathrm{CYC} = \{\pi \in S_N \mid \pi \text{ is a cycle of length } N\}$$

For any nonempty subset S of S_N, define $\mathcal{O}_S$ to be the oracle with a control $x \in [N]$ and a target $y \in [N]$ that acts according to Algorithm 6.

Algorithm 6 The oracle for the problem of distinguishing involutions with no fixed points from cycles of length at least three

```
1: Select $\pi \in S$ uniformly at random
2: Compute $\pi(x)$ where $x$ is the value of the control
3: Add $\pi(x)$ to the target $y$ modulo $N$
```

Theorem 6.3.1. *When $N \geq 3$, classical algorithms with unbounded error cannot solve the problem of distinguishing cycles from involutions with no fixed points using any number of queries.*

Proof. In this problem, an oracle $\mathcal{O}_S$ is given which is either $\mathcal{O}_{\mathrm{INV}}$ or $\mathcal{O}_{\mathrm{CYC}}$; the problem is to determine which of these is the case. Consider querying the oracle when the control is x. Then $\pi(x)$ is a uniformly random value in $[N] \setminus \{x\}$ for both cases so this problem cannot be solved by a classical algorithm. Since multiple classical queries to this oracle will also be uncorrelated by the above argument, it follows that no classical algorithm can distinguish involutions from cycles. □

However, when $N \geq 3$, the problem can be solved by a quantum algorithm using a single query to the oracle as shown in Algorithm 7.

Algorithm 7 The quantum algorithm for distinguishing involutions with no fixed points from cycles

```
1: Prepare the state $\frac{1}{\sqrt{N}}\sum_{x=1}^{N}|x\rangle$
2: Apply $\mathcal{O}_S$ to obtain the state $\frac{1}{\sqrt{N}}\sum_{x=1}^{N}|x,\pi(x)\rangle$
3: Apply the swap test to $\frac{1}{\sqrt{N}}\sum_{x=1}^{N}|x,\pi(x)\rangle$
4: if the swap test outputs “symmetric” then return “INV”
5: else return “CYC”
6: end if
```

We now show that the above algorithm effectively counts the number of transpositions in an arbitrary permutation which is sufficient to distinguish involutions from cycles. Our proof relies on the swap test [28] which provides a way of estimating the absolute value of the inner product of two quantum states. See Chapter 4 for a description of the swap test.

Theorem 6.3.2. *Quantum algorithms can solve the problem of distinguishing cycles from involutions with no fixed points using a single query with one-sided error* $1/2$ *when* $N \geq 3$.

Proof. Consider a general state ρ^{AB} on two identical systems A and B. Then applying the swap test to this system (where the swap exchanges A and B) outputs 0 with probability $\Pr(0) = \frac{1+\operatorname{tr}\rho^{AB}F}{2}$ where F is a swap operation. Applying this formula to the state $\frac{1}{\sqrt{N}}\sum_{x=1}^{N}|x,\pi(x)\rangle$, the probability of observing 0 is

$$\Pr(0) = \frac{1+(1/N)\sum_{xy}\langle\pi(y)|x\rangle\langle y|\pi(x)\rangle}{2} \tag{6.1}$$

$$= \frac{1+(1/N)\left|\{(x,y)\mid \pi(x)=y \text{ and } \pi(y)=x\}\right|}{2} \tag{6.2}$$

Since $N \geq 3$, this probability is $1/2$ if $\pi \in$ CYC and is 1 if $\pi \in$ INV. □

Hence, there is an infinity-vs-one separation in the unbounded-error classical and quantum query complexities for this problem. This analysis can also be applied to obtain an algorithm for estimating the number of transpositions in any permutation.

6.3.2 An infinity-vs-$\Theta(n)$ separation for a modification of Simon's problem

We now show how to modify Simon's problem [116] to obtain an infinity-vs-$\Theta(n)$ separation between the classical and quantum query complexities. Recall that for Simon's problem, we are given oracle access to a function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ and $f(x) = f(y)$ if and only if $x = y + a$ for some fixed element $a \in \mathbb{Z}_2^n$ and our task is to determine a. Classically, exponentially many queries are required; however, quantumly at each step we learn a vector that is orthogonal to a so that the expected number of queries required is $\Theta(n)$. The crucial point here is that this algorithm will return a vector orthogonal to a for any f that is constant and distinct on the cosets $\{x, x + a\}$, so if f changes between calls to the oracle and a does not, then the quantum algorithm will not be affected.

Our randomized oracle is defined as follows. Fix some unknown $a \in \mathbb{Z}_2^n$. Then construct an oracle $\mathcal{O}_a : |x\rangle |y\rangle \mapsto |x\rangle |y + f(x)\rangle$ where $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ is selected uniformly at random at each call subject to the constraint that $f(x) = f(y)$ if and only if $x = y + a$. The problem is then to determine a.

Classically, this cannot be done since each query to the oracle results in a random number; however, the quantum algorithm still requires only $\Theta(n)$ queries.

6.3.3 An infinity-vs-one separation for the hidden linear structure problem

Beaudrap, Cleve and Watrous [38] introduced the hidden linear structure problem where we are given a blackbox that performs the mapping $|x\rangle |y\rangle \mapsto |x\rangle |\pi(y + sx)\rangle$ where $\pi \in S_q$ and $s \in GF(q)$ for $q = 2^n$. The problem is to find s. By extending quantum Fourier transforms to $GF(q)$, Beaudrap, Cleve and Watrous [38] show that this problem can be solved exactly using a single quantum query but classical algorithms require $\Omega(\sqrt{q})$ queries to determine s. They are able to achieve such a query complexity separation by using a non-standard (but still deterministic) oracle model. In the 10 years since their paper, it is still an open question whether such separations are possible in the standard oracle model.

We propose the following randomized variant of their oracle problem. Fix some (unknown) $s \in GF(q)$. Then define the oracle by $\mathcal{O}_s : |x\rangle |y\rangle \mapsto |x\rangle |\pi(y + sx)\rangle$ where π is selected uniformly at random for each query. The goal is still to determine s. Since the quantum algorithm only uses one query it is unaffected by this change; however, classically the output of the oracle is completely random at each query so we obtain an infinity-vs-one separation.

The three separations shown are examples of a more general phenomenon in which randomness can be used to amplify a modest quantum-vs-classical query separation into an unbounded one. We defer this discussion to Section 6.5.

6.4 Uselessness for oracles with internal randomness

We now turn to the general problem of when some number of queries are *useless* for solving an oracle problem. Equivalently we can ask when it is possible to answer an oracle problem with any positive advantage over guessing.

To define oracle problems, we use a slightly more compact notation than in previous sections. An oracle π is defined by a collection of permutations $\pi_{x,r} \in S_M$, where $x \in [N]$ is input by the algorithm, and r is the internal randomness which is distributed according to R. Overloading notation, we say that if the oracle is queried k times, then $\boldsymbol{r} = (r_1, \ldots, r_k)$ is distributed according to R^k, which may not necessarily be an i.i.d. distribution.

To describe the problem we want to solve, we follow the notation of Meyer and Pommersheim [83] while adding internal randomness to the oracle. We are promised that our oracle belongs to a set C, which in general may be a strict subset of all functions from $[N] \times \text{supp}(R)$ to S_M. The set C is partitioned into sets $\{C_j\}$, and our goal is to determine which C_j contains π. By an abuse of notation, we say that C is our oracle problem. Queries are made to an oracle $\mathcal{O}_\pi$ which acts by $|x\rangle |y\rangle \mapsto |x\rangle |\pi_{x,r}(y)\rangle$.

The oracle problem C is a worst-case problem for which we demand that algorithms work well for all choices of $\pi \in C$. However, we also consider average-case problems in which π is distributed according to a known distribution μ. The resulting oracle problem is denoted (C, μ).

Before stating our own results, we describe the main result of [82]. In their model there is no internal randomness, so the action of the oracle is simply $\pi_x \in S_M$ for each $x \in [N]$. If $\boldsymbol{x} = (x_1, \ldots, x_k)$ and $\boldsymbol{y} = (y_1, \ldots, y_k)$, then define $\pi_{\boldsymbol{x}}(\boldsymbol{y}) = (\pi_{x_1}(y_1), \ldots, \pi_{x_k}(y_k))$. Their result may then be stated as follows.

Definition 6.4.1 (Classical uselessness[82]). *k classical queries are* useless *for the oracle problem (C, μ) if for all $\boldsymbol{x} \in [N]^k$, $\boldsymbol{y} \in [M]^k$, $\boldsymbol{z} \in [M]^k$ and j,* $\Pr(\pi \in C_j \mid \pi_{\boldsymbol{x}}(\boldsymbol{y}) = \boldsymbol{z}) = \Pr(\pi \in C_j)$*, where π is distributed according to μ.*

Definition 6.4.2 (Quantum uselessness[82]). *k quantum queries are* useless *for the oracle problem (C, μ) if for any k-query quantum algorithm run on any initial state and any POVM measurement $\{M_s\}$ which is made on the output of the algorithm,* $\Pr(\pi \in C_j \mid s) = \Pr(\pi \in C_j)$ *for all j and s, where π is distributed according to μ.*

We pause to briefly comment on the connection to unbounded-error query complexity. Unbounded-error query complexity typically refers to binary problems, i.e. when C is partitioned into C_0, C_1 and the goal is to determine which one π belongs to with success probability $> 1/2$. In this case, the statement that k (quantum or classical) queries are useless for (C, μ) (for some μ) is equivalent to the unbounded-error query complexity of C being $> k$. This is stated precisely and proved in Section 6.8.

The main result of [82] is the following theorem.

Theorem 6.4.3 (Classical uselessness implies quantum uselessness[82]). *For any deterministic oracle problem (C, μ), if $2k$ classical queries are useless then k quantum queries are useless.*

We will give an alternate proof of this theorem, establish a converse, and generalize it to oracles with internal randomness.

6.4.1 Definitions of Classical Uselessness

In order to characterize uselessness for oracles with internal randomness, we first need to extend the definitions to this case. As above, we define $\pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}) = (\pi_{x_1,r_1}(y_1), \ldots \pi_{x_k,r_k}(y_k))$.

One natural definition of uselessness in this setting is that a classical algorithm ignorant of the oracle's internal randomness should not be able to gain any nontrivial advantage in learning which C_j contains π.

Definition 6.4.4 (Weak classical uselessness)**.** *If (C, μ) is an oracle problem, then k classical queries are* weakly useless *if for all $\boldsymbol{x} \in [N]^k$, $\boldsymbol{y}, \boldsymbol{z} \in [M]^k$ and j, $\Pr(\pi \in C_j \mid \pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}) = \boldsymbol{z}) = \Pr(\pi \in C_j)$, where π and $\boldsymbol{r}$ are distributed according to μ and R^k.*

It is easy to see that if $2k$ classical queries are weakly useless then k quantum queries need not be useless since Algorithm 7 is a counterexample. The proper classical analog of quantum uselessness is obtained by allowing k pairs of classical queries each of which share a seed.

A much stronger definition of uselessness would be to allow the classical algorithm to see, or equivalently to choose, the internal random bits used by the oracle.

Definition 6.4.5 (Strong classical uselessness)**.** *If (C, μ) is an oracle problem, then k classical queries are* strongly useless *if for all $\boldsymbol{x} \in [N]^k$, $\boldsymbol{y}, \boldsymbol{z} \in [M]^k$ and all possible values $\boldsymbol{r} \in \operatorname{supp}(R^k)$,*

$$\Pr(\pi \in C_j \mid \pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}) = \boldsymbol{z}) = \Pr(\pi \in C_j) \tag{6.3}$$

for all j, where π is distributed according to μ.

We will see later that strong classical uselessness for $2k$ queries is sufficiently powerful to imply quantum uselessness for k queries. However, it is in fact too strong, so the definition must be weakened as follows.

Definition 6.4.6 (Pairwise classical uselessness)**.** *If (C, μ) is an oracle problem, then $2k$ classical queries are* pairwise useless *if for all $\boldsymbol{x}, \boldsymbol{x}' \in [N]^k$, $\boldsymbol{y}, \boldsymbol{y}', \boldsymbol{z}, \boldsymbol{z}' \in [M]^k$ and j, $\Pr(\pi \in C_j \mid \pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}) = \boldsymbol{z}, \pi_{\boldsymbol{x}',\boldsymbol{r}}(\boldsymbol{y}') = \boldsymbol{z}') = \Pr(\pi \in C_j)$, where π and $\boldsymbol{r}$ are distributed according to μ and R^k.*

This definition ensures that each pair of query values (x_i, x_i') shares the same random seed r_i. We will see later that this corresponds precisely (in the unbounded error setting) to the power of quantum queries, because the density matrix resulting from a quantum query depends on only one random seed, while the different row and column indices interrogate two different choices of x, y.

It is important to note that weak classical uselessness and pairwise classical uselessness are not comparable: there exist problems that satisfy weak classical uselessness but not pairwise classical uselessness and vice versa. Section 6.3.1 gives an example where two classical queries are weakly useless but not pairwise useless. For an example of a problem where two classical queries are not weakly useless but are pairwise useless, let C be the set of all balanced binary functions on $\{0,1\}$ and let f be chosen uniformly at random from C. Consider the task of determining the function implemented by the oracle that acts for the i^{th} query by $|x\rangle \mapsto |x \oplus f(r_i)\rangle$ where r_i is the i^{th} random seed; let r_1 be uniformly distributed in $\{0,1\}$ and let $r_i = 0$ for $i \geq 2$. Clearly, two classical queries with the random seeds r_1 and r_2 determine f. However, two classical queries that share the random seed r_1 yield no useful information.

It is easy to show that uselessness does not depend on the distribution $\mu(\pi \in C_j)$ over the classes provided the probability of each class is positive. However, it does depend on the conditional distribution of the oracle within each class. Consider the problem of determining the parity of a binary function $f : [N] \to \{0,1\}$; by tweaking the conditional distribution of f for each parity, we can cause $f(1)$ to be equal to the parity of f with high probability so a single query to f wouldn't be useless. On the other hand, if the conditional distribution for f were uniform, $N - 1$ classical queries would be useless.

6.4.2 Uselessness results

Our main result in this section is the following equivalence:

Theorem 6.4.7. *For any oracle problem (C, μ), k quantum queries are useless if and only* *if $2k$ classical queries are pairwise useless.*

For deterministic oracles, weak, pairwise and strong classical uselessness are all the same. In this case, Theorem 6.4.7 can be simplified to the following strengthening of Theorem 6.4.3.

Corollary 6.4.8. *For any deterministic oracle problem (C, μ), k quantum queries are useless if and only if $2k$ classical queries are useless.*

6.4.3 Encoding oracles in states

In this section, we will prove Theorem 6.4.7. Our strategy will be to show that in the unbounded-error setting, the optimal algorithms make a series of fixed queries and then measure the resulting states. The key ingredient is to show that oracles can be encoded in states in a way that is perfectly efficient in terms of queries (i.e. one oracle call creates one state, and one state simulates one oracle call), albeit at a cost of producing the output "I don't know" most of the time. We define these encodings first for deterministic oracles.

Definition 6.4.9. *Let $\mathcal{O}_\pi$ be a deterministic permutation oracle that maps $|x, y\rangle \in \mathbb{C}^N \otimes \mathbb{C}^M$ to $|x, \pi_x(y)\rangle$. Then define the* encoding *of π to be $|\psi_\pi\rangle = \frac{1}{\sqrt{NM}} \sum_{x\in[N],y\in[M]} |x\rangle^X |y\rangle^Y |\pi_x(y)\rangle^Z$. Here X, Y, Z label different registers for notational convenience.*

Clearly one use of $\mathcal{O}_\pi$ allows the creation of one copy of $|\psi_\pi\rangle$; simply prepare the state $\frac{1}{\sqrt{NM}} \sum_{x,y} |x\rangle^X |y\rangle^Y |y\rangle^Z$ and apply $\mathcal{O}_\pi$ to registers XZ. We will see shortly that one copy of $|\psi_\pi\rangle$ can in turn simulate one use of $\mathcal{O}_\pi$, albeit with a very high, but heralded, failure probability. Before proving this result, we show how Definition 6.4.9 generalizes to oracles with internal randomness.

Definition 6.4.10. *Let $\mathcal{O}_\pi$ be an oracle whose action is defined by $\mathcal{O}_\pi(|x\rangle\langle x'| \otimes |y\rangle\langle y'| = \mathsf{E}_{r\sim R} |x\rangle\langle x'| \otimes |\pi_{x,r}(y)\rangle\langle \pi_{x',r}(y')|$. For each r, define the deterministic oracle $\mathcal{O}_{\pi,r}$ by $\mathcal{O}_{\pi,r}|x, y\rangle = |x, \pi_{x,r}(y)\rangle$ and define the* encoding *for fixed r to be $|\psi_{\pi,r}\rangle = \frac{1}{\sqrt{NM}} \sum_{x\in[N],y\in[M]} |x\rangle^X |y\rangle^Y |\pi_{x,r}(y)\rangle^Z$.*

Now we define encodings of oracles with randomness.

Definition 6.4.11. *If $\mathcal{O}_\pi$ is an oracle with internal randomness, then define the* encoding *of $\mathcal{O}_\pi$ to be $\rho_\pi = \mathsf{E}_r \psi_{\pi,r}$.*

For convenience, we use the standard convention mentioned in Section 4.1 that $\psi = |\psi\rangle\langle\psi|$. The utility of considering encodings comes from the following operational equivalence.

Theorem 6.4.12. *(a) One use of $\mathcal{O}_\pi$ can create one copy of ρ_π.*

(b) It is possible to consume one copy of ρ_π and simulate $\mathcal{O}_\pi$ with success probability $1/NM^2$. The simulation outputs a classical flag indicating success or failure.

In both cases, the run time required is linear in the number of qubits, i.e. $O(\log NM)$.

We point out that in the simulation, failure destroys not only the encoding, but also the state input to the oracle. Nevertheless, this simulation is enough to distinguish the case when k queries are useless from the case when they are not.

Additionally, Theorem 6.4.12 is stated implicitly in terms of a distribution R. In the case of k queries correlated according to R^k, we have the following variant:

Theorem 6.4.13. *(a) k uses of $\mathcal{O}_\pi$ can create $\rho_\pi^k = \mathsf{E}_{\boldsymbol{r}\sim R^k}[\psi_{\pi,r_1} \otimes \cdots \otimes \psi_{\pi,r_k}]$.*

(b) It is possible to consume ρ_π^k and simulate k uses of $\mathcal{O}_\pi$ with success probability $1/N^k M^{2k}$, again with a flag indicating success or failure.

As a corollary, for correlated internal randomness in the unbounded-error scenario, we can permit algorithms to make the k oracle calls in any order. We will only prove Theorem 6.4.13, since it subsumes Theorem 6.4.12.

Proof. To create ρ_π^k, we simply apply $\mathcal{O}_\pi$ k times to $\left(\frac{1}{\sqrt{NM}} \sum_{x,y} |x\rangle^X |y\rangle^Y |y\rangle^Z\right)^{\otimes k}$.

For the second reduction, suppose we are given a copy of ρ_π^k and would like to apply $\mathcal{O}_\pi$ to simulate the ith query of some algorithm. If we condition on $\boldsymbol{r}$, then ρ_π^k becomes the state $\psi_{\pi,r_1} \otimes \cdots \otimes \psi_{\pi,r_k}$. We will use the ith component of this state to simulate our query.

Suppose we want to simulate the action of $\mathcal{O}_{\pi,r_i}$ on the state $|x'\rangle^{X'}|y'\rangle^{Y'}$. Define

$$A = \sum_{x\in[N]} |x\rangle^{X}\langle x,x|^{XX'} \otimes \frac{1}{\sqrt{M}} \sum_{y\in[M]} \langle y,y|^{YY'}$$

Since $AA^\dagger = \sum_x |x\rangle\langle x| = I_N$, it follows that $A^\dagger A \le I_{N^2M^2}$ and $\{A, \sqrt{I - A^\dagger A}\}$ comprise a valid collection of measurement operators. Our simulation will apply this measurement, with outcome A labeled success, and $\sqrt{I - A^\dagger A}$ labeled failure.

Upon outcome $\sqrt{I - A^\dagger A}$, the algorithm declares failure. If this occurs at any step of a multi-query algorithm, then the algorithm should guess j according to the *a priori* distribution μ. Thus, for the purposes of determining whether the algorithm outperforms the best guessing strategy, it then suffices to consider only the cases when outcome A occurs.

Upon outcome A, the state $|\psi_{\pi,r_i}\rangle^{XYZ}|x'\rangle^{X'}|y'\rangle^{Y'}$ is mapped to the (unnormalized) state $\frac{1}{\sqrt{NM^2}}|x'\rangle^{X}|\pi_{x,r_i}(y')\rangle^{Z}$. Since the normalization is independent of the input, this means that A occurs with probability $1/NM^2$ regardless of the input state. Conditioned on this outcome, the resulting map is precisely the action of $\mathcal{O}_{\pi,r_i}$.

The overall algorithm succeeds when each of the k queries succeeds. Since each query succeeds with probability $1/NM^2$, the overall algorithm succeeds with probability $1/N^kM^{2k}$. □

Armed with our notion of encoding, it is straightforward to characterize quantum uselessness.

Corollary 6.4.14. *Define $\sigma_j = \mathsf{E}_{\pi\in C_j}\rho_\pi^k$. Then k quantum queries are useless if and only if all the σ_j are the same.*

Proof. By Theorem 6.4.13, any k-query algorithm can WLOG create ρ_π^k, resulting in the state σ_j if π is drawn randomly from C_j. The algorithm then proceeds to determine which σ_j it holds, using no further oracle queries. If all the σ_j are equal, then it can learn nothing about j. Conversely, if some σ_j is different from the others, then there is a measurement that will be able to guess j with positive advantage. □

To conclude the proof of Theorem 6.4.7, observe that the quantity on the LHS of Definition 6.4.6 is $\operatorname{tr}\left(|\boldsymbol{x}\rangle\langle\boldsymbol{x'}| \otimes |\boldsymbol{y}\rangle\langle\boldsymbol{y'}| \otimes |\boldsymbol{z}\rangle\langle\boldsymbol{z'}|\right) \mathsf{E}_{\pi\in C_j}\mu(\pi)\rho_\pi^k = \operatorname{tr}\left(|\boldsymbol{x}\rangle\langle\boldsymbol{x'}| \otimes |\boldsymbol{y}\rangle\langle\boldsymbol{y'}| \otimes |\boldsymbol{z}\rangle\langle\boldsymbol{z'}|\right)\sigma_j$ which will be independent of j for all $\boldsymbol{x}, \boldsymbol{x'}, \boldsymbol{y}, \boldsymbol{y'}, \boldsymbol{z}, \boldsymbol{z'}$ if and only if all of the σ_j are identical. Since we can simulate this measurement using $2k$ classical queries, the application of Corollary 6.4.14, completes the proof of Theorem 6.4.7.

Theorem 6.4.15. *Suppose that for some oracle problem k classical queries are weakly useless but k quantum queries are not useless. Then there exists an oracle problem in which this separation holds where the oracle acts by bitwise XOR.*

Proof. Consider an oracle problem (C, μ) for which k classical queries are weakly useless but k quantum queries are not useless. The oracle acts by $\mathcal{O} : |x\rangle|y\rangle \mapsto |x\rangle|\pi_{x,r_i}(y)\rangle$ on the i^{th} call. We can define a new oracle $\mathcal{O}' : |x\rangle|y\rangle|z\rangle \mapsto |x\rangle|y\rangle|z \oplus \pi_{x,r_i}(y)\rangle$. Our new oracle $\mathcal{O}'$ can be used to prepare the encoding for $\mathcal{O}$ so k queries to $\mathcal{O}'$ can simulate any quantum algorithm that uses k queries to $\mathcal{O}$. Classically, $\mathcal{O}'$ can be simulated using $\mathcal{O}$ so we conclude that k classical queries to $\mathcal{O}'$ are weakly useless. □

6.5 Amplifying separations

We now leverage our results to obtain a general method of amplifying any separation between classical and quantum uselessness. Let (C, μ) be an oracle problem where C is partitioned into $\{C_i\}$ and r_j is the j^{th} random seed. For each $\pi \in C$, we have an oracle $\mathcal{O}_\pi$. Suppose that k classical queries are weakly useless but k quantum queries are not useless. Let us define the oracle $\mathcal{O}_i : |x_1\rangle \cdots |x_k\rangle |y_1\rangle \cdots |y_k\rangle \mapsto |x_1\rangle \cdots |x_k\rangle |\pi_{x_1,r_1}(y_1)\rangle \cdots |\pi_{x_k,r_k}(y_k)\rangle$ where π is selected from C_i according to μ (this is done independently for each query), $\boldsymbol{r}$ is distributed according to R^k and a fresh random seed $\boldsymbol{r}$ is used for every query to $\mathcal{O}_i$. Consider the problem of determining i where the oracle $\mathcal{O}_i$ is given with probability $\mu(\pi \in C_i)$.

Theorem 6.5.1. *Any number of classical queries to the oracle $\mathcal{O}_i$ is weakly useless for determining i.*

Proof. Clearly, a single query to $\mathcal{O}_i$ is equivalent to k queries to the original oracle which are weakly useless by assumption. We conclude that a single classical query to the new oracle is weakly useless. We now show that ℓ classical queries are weakly useless for any $\ell \geq 1$. Let $\boldsymbol{x}_j \in [N]^k$, $\boldsymbol{y}_j, \boldsymbol{z}_j \in [M]^k$ and let each $\boldsymbol{r}_j$ be sampled independently from R^k where $1 \leq j \leq \ell$. We must prove that

$$\Pr(i \mid \pi^j_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = \boldsymbol{z}_j, j = 1, \ldots, \ell) = \Pr(i) \tag{6.4}$$

where each π^j is sampled independently from C_i according to μ. This condition is equivalent to

$$\Pr(\pi^j_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = z_j, j = 1, \ldots, \ell \mid i) = \Pr(\pi^j_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = \boldsymbol{z}_j, j = 1, \ldots, \ell) \tag{6.5}$$

Note that by construction,

$$\Pr(\pi^j_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = z_j, j = 1, \ldots, \ell \mid i) = \prod_j \Pr(\pi^j_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = \boldsymbol{z}_j \mid i)$$

By our assumption that k classical queries to the original oracle are weakly useless, we have that $\Pr(i \mid \pi_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = \boldsymbol{z}_j) = \Pr(i)$ or equivalently $\Pr(\pi_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = \boldsymbol{z}_j \mid i) = \Pr(\pi_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = \boldsymbol{z}_j)$. Therefore,

$$\begin{aligned}
\Pr(\pi^j_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = \boldsymbol{z}_j, j = 1,\ldots,\ell) &= \sum_i \Pr(\pi^j_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = z_j, j = 1,\ldots,\ell \mid i) \Pr(i) && (6.6)\\
&= \sum_i \left(\prod_j \Pr(\pi^j_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = z_j \mid i) \right) \Pr(i) && (6.7)\\
&= \sum_i \left(\prod_j \Pr(\pi^j_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = z_j) \right) \Pr(i) && (6.8)\\
&= \prod_j \Pr(\pi^j_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = z_j) && (6.9)\\
&= \prod_j \Pr(\pi^j_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = z_j \mid i) && (6.10)\\
&= \Pr(\pi^j_{\boldsymbol{x}_j,\boldsymbol{r}_j}(\boldsymbol{y}_j) = z_j, j = 1,\ldots,\ell \mid i) && (6.11)
\end{aligned}$$

which is the desired result. □

We conclude that no matter how many classical queries are made to $\mathcal{O}_i$, no information is obtained about i. On the other hand, we have the following result:

Theorem 6.5.2. *A single quantum query to $\mathcal{O}_i$ is* not *useless for determining i.*

Proof. One can use a single quantum query to $\mathcal{O}_i$ to construct the state ρ^k_π as described in Theorem 6.4.13. Applying Theorem 6.4.13, this state may be used to guess i with higher probability than random guessing since k quantum queries are not useless. □

Thus, we have constructed an infinity-vs-one separation in unbounded-error classical and quantum query complexities from an arbitrary initial separation. One can also construct an infinity-vs-one separation in the bounded-error setting from an arbitrary separation in the unbounded setting; the construction is straightforward and we defer the details to Appendix 6.7.

6.6 *Alternate proofs of uselessness*

In this section, we present alternate proofs of various uselessness theorems. These proofs do not rely on the idea of encoding oracles into states, but instead give direct arguments, so they are more self-contained, although also longer. First we prove that pairwise classical uselessness implies quantum uselessness.

Proof. The proof is an extension of the technique used by Meyer and Pommersheim [82]. Suppose that $2k$ classical queries are pairwise useless. Consider an oracle π that acts by $\mathcal{O}_\pi^i : |x, y, z\rangle \mapsto |x, \pi_{x,r_i}, z\rangle$ for the i^{th} query. (The first two registers are the usual input and output registers for the oracle, while the third register is for auxilliary computations by the algorithm in between oracle calls.) Note that, as before, the r_i variables may obey an arbitrary joint distribution so different queries are not necessarily independent. Consider an arbitrary k-query quantum algorithm with initial state ρ_0 and POVM $\{M_s\}$. For the i^{th} query, the algorithm queries the oracle and then applies an arbitrary unitary transformation U_i. This yields the final state

$$\rho_\pi = U_k \mathcal{O}_\pi^k \dots U_1 \mathcal{O}_\pi^1 \rho_0 \mathcal{O}_\pi^{1\dagger} U_1^\dagger \dots \mathcal{O}_\pi^{k\dagger} U_k^\dagger \tag{6.12}$$

Let us fix the random seed used for the i^{th} query as r_i. The final state is then

$$\rho_{\pi,\boldsymbol{r}} = U_k P_{r_k} \dots U_1 P_{r_1} \rho_0 P_{r_1}^\dagger U_1^\dagger \dots P_{r_k}^\dagger U_k^\dagger \tag{6.13}$$

where P_{r_i} denotes the permutative action $|x, y, z\rangle \mapsto |x, \pi_{x,r_i}(y), z\rangle$ of the oracle when the random seed is fixed to r_i. Let A be a matrix, $L = (x, y, z)$ and $L' = (x', y', z')$. Then

$$\left(P_{r_i} A P_{r_i}^\dagger\right)_{L,L'} = \left\langle x, \pi_{x,r_i}^{-1}(y), z\right| A \left|x', \pi_{x',r_i}^{-1}(y'), z'\right\rangle \tag{6.14}$$

$$= A_{\pi_{\cdot,r_i}(L),\pi_{\cdot,r_i}(L')} \tag{6.15}$$

where $\pi_{\cdot,r_i}(L) = (x, \pi^{-1}_{x,r_i}(y), z)$. Then the state after the $i+1^{\text{th}}$ query (for the fixed values $\boldsymbol{r}$ of the seeds) is

$$\rho_{i+1,\boldsymbol{r}} = U_{i+1} P_{r_{i+1}} \rho_{i,\boldsymbol{r}} P^{\dagger}_{r_{i+1}} U^{\dagger}_{i+1} \tag{6.16}$$

so that the matrix elements are

$$(\rho_{i+1,\boldsymbol{r}})_{L,L'} = \sum_{K,K'} (U_{i+1})_{L,K} (\rho_{i,\boldsymbol{r}})_{\pi_{\cdot,r_{i+1}}(K),\pi_{\cdot,r_{i+1}}(K')} (U^{\dagger}_{i+1})_{K',L'} \tag{6.17}$$

This value is a function of L, L', $\pi_{\cdot,r_{i+1}}(K)$ and $\pi_{\cdot,r_{i+1}}(K')$. Therefore, the final state $\rho_{\pi,\boldsymbol{r}} = \rho_{k,\boldsymbol{r}}$ may be written as

$$\rho_{\pi,\boldsymbol{r}} = \sum_{I} Q_I(\pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}), \pi_{\boldsymbol{x}',\boldsymbol{r}}(\boldsymbol{y}')) \tag{6.18}$$

where $I = (L_1, \ldots, L_k, L'_1 \ldots, L'_k)$. Let $\mathsf{E}_{\pi|\pi\in C_j}$ denote the expectation over π according to the distribution $\Pr(\pi \mid \pi \in C_j)$. Then for any j,

$$\begin{aligned}
\mathsf{E}_{\pi|\pi\in C_j} \rho_{\pi,\boldsymbol{r}} &= \sum_{I} \mathsf{E}_{\pi|\pi\in C_j} Q_I(\pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}), \pi_{\boldsymbol{x}',\boldsymbol{r}}(\boldsymbol{y}')) && (6.19)\\
&= \sum_{I} \sum_{\boldsymbol{w},\boldsymbol{w}'} \mathsf{E}_{\pi|\pi\in C_j} Q_I(\pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}), \pi_{\boldsymbol{x}',\boldsymbol{r}}(\boldsymbol{y}'))[\pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}) = \boldsymbol{w}, \pi_{\boldsymbol{x}',\boldsymbol{r}}(\boldsymbol{y}') = \boldsymbol{w}'] && (6.20)\\
&\text{where } \boldsymbol{w} = (w_1, \ldots, w_k) \text{ and } \boldsymbol{w}' = (w'_1, \ldots, w'_k)\\
&= \sum_{I} \sum_{\boldsymbol{w},\boldsymbol{w}'} Q_I(\boldsymbol{w}, \boldsymbol{w}') \mathsf{E}_{\pi|\pi\in C_j} [\pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}) = \boldsymbol{w}, \pi_{\boldsymbol{x}',\boldsymbol{r}}(\boldsymbol{y}') = \boldsymbol{w}'] && (6.21)\\
&= \sum_{I} \sum_{\boldsymbol{w},\boldsymbol{w}'} Q_I(\boldsymbol{w}, \boldsymbol{w}') \Pr(\pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}) = \boldsymbol{w}, \pi_{\boldsymbol{x}',\boldsymbol{r}}(\boldsymbol{y}') = \boldsymbol{w}' \mid \pi \in C_j) && (6.22)\\
& && (6.23)
\end{aligned}$$

Taking the expectation over the random seeds $\boldsymbol{r}$,

$$\mathsf{E}_{\pi|\pi\in C_j}\mathsf{E}_{\boldsymbol{r}}\rho_{\pi,\boldsymbol{r}} = \sum_I \sum_{\boldsymbol{w},\boldsymbol{w}'} Q_I(\boldsymbol{w},\boldsymbol{w}')\mathsf{E}_{\boldsymbol{r}} \Pr(\pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}) = \boldsymbol{w}, \pi_{\boldsymbol{x}',\boldsymbol{r}}(\boldsymbol{y}') = \boldsymbol{w}' \mid \pi \in C_j) \tag{6.24}$$

$$= \sum_I \sum_{\boldsymbol{w},\boldsymbol{w}'} Q_I(\boldsymbol{w},\boldsymbol{w}') \Pr(\pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}) = \boldsymbol{w}, \pi_{\boldsymbol{x}',\boldsymbol{r}}(\boldsymbol{y}') = \boldsymbol{w}' \mid \pi \in C_j) \tag{6.25}$$

$$= \sum_I \sum_{\boldsymbol{w},\boldsymbol{w}'} Q_I(\boldsymbol{w},\boldsymbol{w}') \Pr(\pi \in C_j \mid \pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}) = \boldsymbol{w}, \pi_{\boldsymbol{x}',\boldsymbol{r}}(\boldsymbol{y}') = \boldsymbol{w}') \tag{6.26}$$

$$\cdot \frac{\Pr(\pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}) = \boldsymbol{w}, \pi_{\boldsymbol{x}',\boldsymbol{r}}(\boldsymbol{y}') = \boldsymbol{w}')}{\Pr(\pi \in C_j)} \tag{6.27}$$

$$= \sum_I \sum_{\boldsymbol{w},\boldsymbol{w}'} Q_I(\boldsymbol{w},\boldsymbol{w}') \Pr(\pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}) = \boldsymbol{w}, \pi_{\boldsymbol{x}',\boldsymbol{r}}(\boldsymbol{y}') = \boldsymbol{w}') \tag{6.28}$$

by pairwise classical uselessness

$$= \mathsf{E}_{\pi}\mathsf{E}_{r} \sum_I \sum_{\boldsymbol{w},\boldsymbol{w}'} Q_I(\boldsymbol{w},\boldsymbol{w}')[\pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}) = \boldsymbol{w}, \pi_{\boldsymbol{x}',\boldsymbol{r}}(\boldsymbol{y}') = \boldsymbol{w}'] \tag{6.29}$$

$$= \mathsf{E}_{\pi}\mathsf{E}_{r} \sum_I Q_I(\pi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{y}), \pi_{\boldsymbol{x}',\boldsymbol{r}}(\boldsymbol{y}')) \tag{6.30}$$

$$= \mathsf{E}_{\pi}\mathsf{E}_{\boldsymbol{r}}\rho_{\pi,\boldsymbol{r}} \tag{6.31}$$

$$\tag{6.32}$$

Defining $\rho_\pi = \mathsf{E}_{\boldsymbol{r}}\rho_{\pi,\boldsymbol{r}}$, this may be written as

$$\mathsf{E}_{\pi|\pi\in C_j}\rho_\pi = \mathsf{E}_\pi\rho_\pi \tag{6.33}$$

Note that for a random $\pi \in C$, the state after running the algorithm is $\mathsf{E}_\pi\rho_\pi$ and for a random $\pi \in C_j$ the state is $\mathsf{E}_{\pi|\pi\in C_j}\rho_\pi$. Now, consider the probability that $\pi \in C_j$ given the measurement outcome s. We have

$$\Pr(\pi \in C_j \mid s) = \frac{\Pr(s \mid \pi \in C_j)\Pr(\pi \in C_j)}{\Pr(s)} \tag{6.34}$$

$$= \frac{\operatorname{tr} M_s \mathsf{E}_{\pi|\pi\in C_j}\rho_f}{\operatorname{tr} M_s \mathsf{E}_\pi\rho_\pi} \Pr(\pi \in C_j) \tag{6.35}$$

$$= \Pr(\pi \in C_j) \tag{6.36}$$

as claimed. $\square$

Next, we prove that quantum uselessness implies classical uselessness, but in the special case of standard oracles that act via XOR but with internal randomness. Specifically, consider an oracle that acts by $\mathcal{O}_f^i : |x, y, z\rangle \mapsto |x, y \oplus f(x, r_i), z\rangle$ for the i^{th} query. As before, we allow the r_i variables to be drawn from an arbitrary joint distribution.

Proof. Suppose that k quantum queries are useless. This means that for any POVM $\{M_s\}$ and quantum algorithm run on any initial state ρ_0, $\Pr(f \in C_j \mid s) = \Pr(f \in C_j)$ for all j. Since $\Pr(f \in C_j \mid s) = \frac{\Pr(s|f \in C_j)\Pr(f \in C_j)}{\Pr(s)}$, this implies that

$$\Pr(s \mid f \in C_j) = \Pr(s) \tag{6.37}$$

for all j. Let us choose the initial state

$$\rho_0 = \left(\frac{1}{N} \sum_{x,x'} |x\rangle \langle x'| \otimes |0\rangle \langle 0| \right)^{\otimes k} \tag{6.38}$$

and the algorithm defined by the unitary operator $\bigotimes_{i=1}^k \mathcal{O}_f^i$. The result of running the algorithm assuming a particular function f and fixed seeds $\boldsymbol{r}$ is then

$$\begin{aligned} \rho_{f,\boldsymbol{r}} &= \left(\bigotimes_{i=1}^k \mathcal{O}_f^i \right) \rho_0 \left(\bigotimes_{i=1}^k \mathcal{O}_f^i \right)^\dagger && (6.39) \\ &= \frac{1}{N^k} \bigotimes_{i=1}^k \sum_{x,x'} |x, f(x, r_i)\rangle \langle x', f(x', r_i)| && (6.40) \end{aligned}$$

For a particular function f, the state after running the algorithm is

$$\begin{aligned} \rho_f &= \mathsf{E}_{\boldsymbol{r}} \rho_{f,\boldsymbol{r}} && (6.41) \\ &= \frac{1}{N^k} \mathsf{E}_{\boldsymbol{r}} \bigotimes_{i=1}^k \sum_{x,x'} |x, f(x, r_i)\rangle \langle x', f(x', r_i)| && (6.42) \end{aligned}$$

Now

$$\Pr(s) = \mathsf{E}_f \Pr(s \mid f) \tag{6.43}$$
$$= \operatorname{tr} M_s \mathsf{E}_f \rho_f \tag{6.44}$$
$$= \operatorname{tr} M_s \rho_C \tag{6.45}$$

Similarly,

$$\Pr(s \mid f \in C_j) = \mathsf{E}_{f|f\in C_j} \Pr(s \mid f) \tag{6.46}$$
$$= \operatorname{tr} M_s \mathsf{E}_{f|f\in C_j} \rho_f \tag{6.47}$$
$$= \operatorname{tr} M_s \rho_{C_j} \tag{6.48}$$

Since $\Pr(s \mid f \in C_j) = \Pr(f \in C_j)$, this implies that

$$\operatorname{tr} M_s(\rho_{C_j} - \rho_C) = 0 \tag{6.49}$$

for *all* POVMs $\{M_s\}$ which means that

$$\rho_{C_j} = \rho_C \tag{6.50}$$
$$\frac{1}{N^k}\mathsf{E}_{\boldsymbol{r}}\mathsf{E}_{f|f\in C_j}\bigotimes_{i=1}^{k}\sum_{x,x'}|x, f(x,r_i)\rangle\langle x', f(x',r_i)| = \frac{1}{N^k}\mathsf{E}_{\boldsymbol{r}}\mathsf{E}_{f}\bigotimes_{i=1}^{k}\sum_{x,x'}|x, f(x,r_i)\rangle\langle x', f(x',r_i)| \tag{6.51}$$

Equating the $((x_1, y_1, \ldots, x_k, y_k), (x'_1, y'_1, \ldots, x'_k, y'_k))$ elements of these matrices, we have that

$$\mathsf{E}_{\boldsymbol{r}}\mathsf{E}_{f|f\in C_j}[f(\boldsymbol{x},\boldsymbol{r}) = \boldsymbol{y}, f(\boldsymbol{x}',\boldsymbol{r}) = \boldsymbol{y}'] = \mathsf{E}_{\boldsymbol{r}}\mathsf{E}_{f}[f(\boldsymbol{x},\boldsymbol{r}) = \boldsymbol{y}, f(\boldsymbol{x}',\boldsymbol{r}) = \boldsymbol{y}'] \tag{6.52}$$
$$\mathsf{E}_{\boldsymbol{r}}\Pr(f(\boldsymbol{x},\boldsymbol{r}) = \boldsymbol{y}, f(\boldsymbol{x}',\boldsymbol{r}) = \boldsymbol{y}' \mid f \in C_j) = \mathsf{E}_{\boldsymbol{r}}\Pr(f(\boldsymbol{x},\boldsymbol{r}) = \boldsymbol{y}, f(\boldsymbol{x}',\boldsymbol{r}) = \boldsymbol{y}') \tag{6.53}$$
$$\Pr(f(\boldsymbol{x},\boldsymbol{r}) = \boldsymbol{y}, f(\boldsymbol{x}',\boldsymbol{r}) = \boldsymbol{y}' \mid f \in C_j) = \Pr(f(\boldsymbol{x},\boldsymbol{r}) = \boldsymbol{y}, f(\boldsymbol{x}',\boldsymbol{r}) = \boldsymbol{y}') \tag{6.54}$$

Applying Bayes' rule, we have

$$\Pr(f \in C_j \mid f(\boldsymbol{x}, \boldsymbol{r}) = \boldsymbol{y}, f(\boldsymbol{x}', \boldsymbol{r}) = \boldsymbol{y}') = \frac{\Pr(f(\boldsymbol{x}, \boldsymbol{r}) = \boldsymbol{y}, f(\boldsymbol{x}', \boldsymbol{r}) = \boldsymbol{y}' \mid f \in C_j) \Pr(f \in C_j)}{\Pr(f(\boldsymbol{x}, \boldsymbol{r}) = \boldsymbol{y}, f(\boldsymbol{x}', \boldsymbol{r}) = \boldsymbol{y}')} \tag{6.55}$$

$$= \Pr(f \in C_j) \tag{6.56}$$

which is precisely the definition of pairwise classical uselessness in the case of oracles that act by XOR. □

Combining this with Theorem 6.4.7, we have the following result

Corollary 6.6.1. *For any oracle problem (C, μ) in the standard model with internal randomness, k quantum queries are useless if and only if $2k$ classical queries are pairwise useless*

Since pairwise classical uselessness is equivalent to classical uselessness when f is deterministic, we have the following corollary.

Corollary 6.6.2. *If k quantum queries are useless for an oracle problem (C, μ) in the standard model, then $2k$ classical queries are useless.*

6.7 Bounded-error infinity-vs-one separations

We now show how to obtain an infinity-vs-one separation in the bounded-error regime from an arbitrary separation between the classical and quantum uselessness. Consider the oracle $\mathcal{O}_i$ as defined above. By Theorem 6.5.2, there exists a single-query quantum algorithm A, a POVM $\{M_s\}$ and an i' such that for some s, $\Pr(i = i' \mid s) > \Pr(i = i')$. Equivalently,

$$\Pr(s \mid i = i') > \Pr(s) \tag{6.57}$$

$$\Pr(s \mid i = i')(1 - \Pr(i = i')) > \Pr(s \mid i \neq i') \Pr(i \neq i') \tag{6.58}$$

$$\Pr(s \mid i = i') > \Pr(s \mid i \neq i') \tag{6.59}$$

$$\Pr(s \mid i = i') = \Pr(s \mid i \neq i') + \epsilon \tag{6.60}$$

for some $\epsilon > 0$. Consider the problem of deciding if $i = i'$ by querying $\mathcal{O}_i$. By running A some large number of times T and using majority voting and Chernoff bounds, we may decide if $i = i'$ with bounded error. Although T may be quite large, the gap is large since it is a separation between an infinite number of classical queries and a finite number of classical queries.

Corollary 6.7.1. *The bounded-error quantum query complexity of deciding if $i = i'$ using $\mathcal{O}_i$ is finite.*

By Theorem 6.5.1, $\Pr(i \mid \pi^j_{\boldsymbol{x}_j, \boldsymbol{r}_j}(\boldsymbol{y}_j) = z_j, j = 1, \ldots) = \Pr(i)$ for all $\ell \geq 1$ and $\boldsymbol{x}_j \in [N]^k$, $\boldsymbol{y}_j, \boldsymbol{z}_j \in [M]^k$. Thus, $\Pr(i = i' \mid \pi^j_{\boldsymbol{x}_j, \boldsymbol{r}_j}(\boldsymbol{y}_j) = z_j, j = 1, \ldots) = \Pr(i = i')$ and $\Pr(i \neq i' \mid \pi^j_{\boldsymbol{x}_j, \boldsymbol{r}_j}(\boldsymbol{y}_j) = z_j, j = 1, \ldots) = \Pr(i \neq i')$ so ℓ queries are weakly useless for deciding if $i = i'$.

Corollary 6.7.2. *Any number of classical queries to the oracle $\mathcal{O}_i$ is weakly useless for deciding if $i = i'$; thus no classical algorithm can decide if $i = i'$ with unbounded error no matter how many queries are made.*

We can construct a new oracle $\mathcal{O}'_i$ that simulates T queries to $\mathcal{O}_i$ using an independent random seed for each query. From this we obtain the following.

Corollary 6.7.3. *The bounded-error quantum query complexity of deciding if $i = i'$ using $\mathcal{O}'_i$ is* 1*.*

Corollary 6.7.4. *Any number of classical queries to the oracle $\mathcal{O}'_i$ is weakly useless for deciding if $i = i'$; thus no classical algorithm can decide if $i = i'$ with unbounded error no matter how many queries are made.*

Thus, we have constructed an infinity-vs-one separation between the bounded-error quantum query complexity and the unbounded-error classical query complexity from an arbitrary initial separation. This comes at the price of large inputs for the constructed oracle.

6.8 Relation between uselessness and unbounded query complexity

In this section, we define *binary* oracle problems to be those where our goal is to output a single bit, or equivalently, where C is partitioned into only two sets C_0, C_1, and our goal is to determine whether $\pi \in C_0$ or $\pi \in C_1$.

Proposition 6.8.1. *Let C be a binary oracle problem. Then the unbounded quantum (resp. classical) query complexity of C is $> k$ if and only if there exists a distribution μ with $\mu(C_0) = \mu(C_1) = 1/2$ such that k quantum (resp. classical) queries are useless for (C, μ).*

Equivalently we could demand that $0 < \mu(C_0) < 1$ because reweighting 0-inputs and 1-inputs does not affect the uselessness properties of a distribution. (The same does *not* hold for changing the probabilities within the class of 0-inputs or 1-inputs.) However, we need to avoid the trivial case in which a distribution is useless because the answer is already known perfectly from the prior distribution μ.

Proof. The "if" direction is easy. If such a μ exists, then by the definition of uselessness, no algorithm can achieve success probability $> 1/2$ with $\leq k$ queries.

For the converse, we use Yao's minimax principle, which states that there exists a distribution μ for which no k-query algorithm can achieve success probability $> 1/2$. Since it is always possible to achieve success probaiblity $\max(\mu(C^{-1}(0)), \mu(C^{-1}(1)))$ by guessing, we must also have $\mu(C^{-1}(0)) = \mu(C^{-1}(1)) = 1/2$. □

A natural generalization of unbounded-error query complexity to non-binary problems would be to define success as guessing the right answer with probability $> \max_j \mu(C_j)$. In this case, uselessness is now a strictly stronger statement whenever μ is such that $\mu(C_j)$ is not the same for each j. To see this, let ν be the distribution over π obtained after making some number of queries. Uselessness states that $\mu(C_j) = \nu(C_j)$ for each j, whereas unbounded-error query complexity depends only on whether $\max_j \mu(C_j) = \max_j \nu(C_j)$.

Chapter 7

A QUANTUM ALGORITHM FOR TREE ISOMORPHISM

7.1 Introduction

The problem of deciding if two graphs are isomorphic has many practical applications such as searching for an unknown molecule in a chemical database [65], verification of hierarchical circuits [129] and generating application specific instruction sets [35]. As we mentioned in Chapter 2, graph isomorphism is not known to be solvable in polynomial time despite a great deal of effort to develop an efficient algorithm. These efforts culminated in Luks' discovery in 1983 of a $2^{O(\sqrt{n \log n})}$ time classical algorithm [18, 16], which has not been improved since.

Another approach this problem which has received much attention is that of quantum algorithms for graph isomorphism. However, no super-polynomial quantum speedups are known even for special cases of the graph isomorphism problem.

One of the major approaches to developing efficient quantum algorithms for the graph isomorphism problem is the hidden subgroup problem (HSP), which is the basis for many super-polynomial quantum speedups including Shor's algorithm for factoring [115] and several others [39, 116, 38].

Unfortunately, a string of negative results has shown that is increasingly unlikely that the hidden subgroup approach will work for graph isomorphism. While it was shown that there exists a quantum measurement that can solve the HSP on the symmetric group [41], it is not known if this measurement can be implemented efficiently and there is evidence that it cannot be. This line of work was started by Moore, Russell and Schulman's proof [87] that strong Fourier sampling (the standard approach to HSPs) is ineffective for the symmetric group. Hallgren, Moore, Rotteler, Russell and Sen showed the stronger result [54] that the measurement for the HSP over the symmetric group must involve entanglement over

$\Omega(n \log n)$ coset states. Finally, Moore, Russell and Sniady proved [88] that the sieve methods used to obtain a quantum speedup for the dihedral HSP [98, 67, 66] cannot significantly outperform the best classical algorithms known for graph isomorphism [18, 16].

Because of these results, other approaches to quantum algorithms for graph isomorphism are of great interest. One of the most promising is the state preparation approach [3], which aims to prepare a complete invariant state that represents the isomorphism class of the graph. This complete invariant state corresponds to the superposition of all permutations of the graph. Since the sets of permutations of two graphs coincide if they are isomorphic and are disjoint if they are non-isomorphic, the complete invariant states for two isomorphic graphs are equal and the complete invariant states for two non-isomorphic graphs are orthogonal. Because the swap test [28] provides a means of distinguishing orthogonal states, the problem of testing isomorphism of two graphs reduces to the ability to prepare complete invariant states. Unfortunately, it is not currently known how to prepare complete invariant states for classes of graphs that are considered difficult classically.

In this chapter, we take a small step towards a state-preparation based algorithm for graph isomorphism by developing a quantum algorithm for rooted tree isomorphism. By considering all possible roots in one of the trees, it is also possible to efficiently decide if two unrooted trees are isomorphic. Although tree isomorphism can be decided in linear time on a classical computer [4], our goal is to make progress on the state preparation approach to graph isomorphism rather than to provide a speedup over classical algorithms for tree isomorphism.

Shor observed[1] that an isomorphism testing algorithm can be used to prepare a complete invariant state. Combined with the linear time classical algorithm from [4], this implies that complete invariant states can be efficiently prepared for trees. However, the resulting algorithm uses the classical algorithm as a subroutine to solve all of the isomorphism problems that it encounters. Consequently, this approach does not seem likely to be useful for classes

[1]Aram Harrow (personal communication).

of graphs for which we do not have efficient classical algorithms.

By contrast, the quantum algorithm for tree isomorphism shown in this chapter relies on techniques that are fundamentally quantum and our techniques are potentially useful for quantum algorithms for more general classes of graphs. All of the runtimes we will give correspond to the depth in the CCAC model introduced in Chapter 5. We concern ourselves only with the runtime and ignore the width and size of the quantum circuits involved.

Our algorithm is based on an efficient solution to what we call the *quantum state symmetrization problem*. In the basic formulation of this problem, we are given a collection of mutually orthogonal states $\{|\psi_i\rangle \mid 1 \leq i \leq \ell\}$ and a permutation group G of degree ℓ and must compute the superposition

$$\frac{1}{\sqrt{|G|}} \sum_{\pi \in G} \bigotimes_{i=1}^{\ell} |\psi_{\pi(i)}\rangle$$

of all permutations of $\bigotimes_{i=1}^{\ell} |\psi_i\rangle$ by elements of G. We accomplish this using *strong generating sets* for permutation groups [117].

Because our tree isomorphism algorithm will need to apply our state symmetrization procedure to states that correspond to subtrees of possibly differing sizes, we need an efficient algorithm for a more general version of the state symmetrization problem that allows the ψ_i's to have different sizes. However, we show that our algorithm for state symmetrization can be generalized to account for this difficulty. (We shall define this more general version of the state symmetrization problem in the next section, but the precise definition is unimportant for the purposes of the present discussion.)

Our isomorphism algorithm then works roughly as follows. Let T be the rooted tree for which we wish to prepare the complete invariant state. We recursively compute the complete invariant state for each subtree corresponding to a child of the root of T. Our idea is then to apply our state symmetrization algorithm to remove all information about the order in which these subtrees appear in T. However, there is a difficulty: some of the subtrees may be isomorphic and will therefore have the same complete invariant state, which makes it impossible to apply our state symmetrization procedure. Fortunately, we are able to overcome this difficulty by adding extra information to the states corresponding to each subtree. This makes different isomorphic subtrees correspond to distinct states; however, after applying our state symmetrization procedure to the states corresponding to the subtrees, we nonetheless still obtain a state for T that depends only on its isomorphism class. In this way, we obtain a recursive procedure for preparing a complete invariant state for a rooted tree T.

7.2 A quantum algorithm for state symmetrization

Our first step is to develop an efficient quantum algorithm for the state symmetrization problem. In addition to being used as a subroutine in our algorithm for tree isomorphism, the state symmetrization problem is related to graph isomorphism. Let $|\psi_1\rangle, \dots, |\psi_n\rangle$ be a sequence of orthonormal states and let G be a subgroup of S_n. The problem is to prepare the state $\frac{1}{\sqrt{|G|}} \sum_{\pi \in G} \bigotimes_{i=1}^{n} \left|\psi_{\pi(i)}\right\rangle$. Consider a graph X. As mentioned in the previous section, if it were possible to efficiently prepare the complete invariant state

$$|X\rangle = \sqrt{\frac{|Aut(X)|}{n!}} \sum_{\pi \in S_n / Aut(X)} |X^{\pi}\rangle$$

then we could solve the graph isomorphism problem efficiently using the swap test [28]. The crucial difference between the state symmetrization problem and the state preparation approach to graph isomorphism is that in the former, symmetrization is performed over a sequence of orthonormal states and it is not clear that graph isomorphism can be cast in this framework.

First, we define the more general version of the state symmetrization problem as promised in Section 7.1. For this, we need a generalized notion of orthogonal states.

Definition 7.2.1. *Let $d_i \in \mathbb{N}$ and $|\psi_i\rangle \in \mathbb{C}^{d_i}$ for each $1 \le i \le \ell$ and let G be a permutation group of degree ℓ. Then the collection $\{|\psi_i\rangle \mid 1 \le i \le \ell\}$ of states is* G-symmeterizable *if there is a unitary matrix V that can be implemented in $\mathsf{poly}(\log \sum_{i=1}^{\ell} d_i)$ time that takes a permutation $\bigotimes_{i=1}^{\ell} \left|\psi_{\pi(i)}\right\rangle$ where $\pi \in G$ and outputs $|\pi\rangle = \bigotimes_{i=1}^{\ell} |\pi(i)\rangle$.*

If a collection of states is G-symmeterizable for every permutation group G of degree ℓ, then we say that it is *symmeterizable.*

Before showing our algorithm for symmetrizing collections of G-symmeterizable states later in Subsection 7.2.3, we consider two special cases of Definition 7.2.1 that are relevant to our quantum algorithm for tree isomorphism and also motivate Definition 7.2.1.

7.2.1 Collections of efficiently-preparable orthonormal states

The first special case of symmeterizable states is the situation where we have a collection $\left\{|\psi_i\rangle \in \mathbb{C}^d \;\middle|\; 1 \le i \le \ell\right\}$ of orthogonal states of the same dimension, along with unitary matrices U_i (that can be implemented in $\mathsf{poly}(d\ell)$ time) such that $|\psi_i\rangle = U_i\,|0\rangle$ for each $1 \le i \le \ell$. In order to prove that the collection $\left\{|\psi_i\rangle \in \mathbb{C}^d \;\middle|\; 1 \le i \le \ell\right\}$ of orthonormal states is G-symmeterizable for any permutation group G, we require the following lemma.

Lemma 7.2.2. *Suppose that $\left\{|\psi_i\rangle \in \mathbb{C}^d \;\middle|\; 1 \le i \le \ell\right\}$ is a collection of orthonormal states of dimension d where each $|\psi_i\rangle = U_i\,|0\rangle$ for some unitary U_i that can be implemented in time t_i. Let $t = \sum_{i=1}^{\ell} t_i$. Then we can implement a unitary U in $4t + O(1)$ time such that each $|\psi_i\rangle = U\,|i\rangle$.*

Proof. Suppose that the initial state is the computational basis state $|j\rangle$. We start by adding a second register initialized to $|0\rangle$ to obtain $|j\rangle\,|0\rangle$. Then, for each $1 \le i \le \ell$, we apply a U_i operation to the second register that is controlled by i on the first register. This yields the state $|j\rangle\,|\psi_j\rangle$. Let $\widetilde{\mathrm{CU_i}}$ denote XORing i into the first register and then applying a U_i operation to the first register where both operations are controlled by 0 on the second register. For each $1 \le i \le \ell$, we then apply the operation

$$(I \otimes U_i) \cdot \widetilde{\mathrm{CU_i}} \cdot (I \otimes U_i)^\dagger$$

to the two registers. Since the above operation effectively sets the first register to $|0\rangle$ and then applies a U_i operation that is controlled by $|\psi_i\rangle$ on the second register, this yields the state $|\psi_j\rangle\,|\psi_j\rangle$.

Now, we need to uncompute the second register. Let $\widehat{\mathrm{CU_i}}$ denote applying a U_i operation to the second register that is controlled by 0 on the first register. By performing the operation

$$(U_i \otimes I) \cdot \widehat{\mathrm{CU_i}}^{\dagger} \cdot (U_i \otimes I)^{\dagger}$$

on the two registers for each $1 \leq i \leq \ell$, we obtain $|\psi_j\rangle\,|0\rangle$. From this, we can get the desired state $|\psi_j\rangle$ by discarding the constant register $|0\rangle$. Thus, we can efficiently implement an unitary U such that each $|\psi_i\rangle = U\,|i\rangle$. The complexity claimed follows by noting that Toffoli gates with an arbitrary number of controls can be implemented in $O(1)$ time [57, 27, 121]. □

It follows that the collection of orthonormal states $\left\{|\psi_i\rangle \in \mathbb{C}^d \;\middle|\; 1 \leq i \leq \ell\right\}$ is symmeterizable.

Corollary 7.2.3. *Let $\left\{|\psi_i\rangle \in \mathbb{C}^d \;\middle|\; 1 \leq i \leq \ell\right\}$ be a collection of orthonormal states of dimension d where each $|\psi_i\rangle = U_i\,|0\rangle$ and each U_i can be implemented in time t_i. Let $t = \sum_{i=1}^{\ell} t_i$. Then we can implement a unitary V in $4t + O(1)$ time such that each $V\left(\bigotimes_{i=1}^{\ell} |\psi_{\pi(i)}\rangle\right) = \bigotimes_{i=1}^{\ell} |\pi(i)\rangle$.*

Proof. Define $V = (U^{\dagger})^{\otimes \ell}$ where U is as in Lemma 7.2.2. The result then follows immediately from Lemma 7.2.2. □

7.2.2 Delimited orthonormal states

We call the collections of symmeterizable states that arise in our algorithm for tree isomorphism *delimited orthonormal states.* Essentially, a collection of delimited orthonormal states is one in which each state starts with a special *separator state* that can be distinguished from the other parts of all of the states in the collection. Given a quantum state that corresponds to a permutation of the states in the collection, we can then use the separator states to find the boundaries between the elements of the permutation. This allows us to compute the permutation.

Before we give a formal definition of delimited orthonormal states, some additional notation is necessary. In what follows, we shall deal with spaces of the form $(\mathbb{C}^5)^{\otimes n}$. Thus, our qudits[2] are 5-dimensional. We do this because it allows us to maintain two separate systems of binary numbers that are mutually orthogonal as well as a special fifth basis state that is used to pad states of different lengths. Let us denote the basis states by $|0\rangle$, $|1\rangle$, $|\bar{0}\rangle$, $|\bar{1}\rangle$ and $|\Box\rangle$. For a natural number j, we let $|j\rangle$ denote the binary representation of j using $|0\rangle$ and $|1\rangle$ and $|\bar{j}\rangle$ denote the binary representation of j using $|\bar{0}\rangle$ and $|\bar{1}\rangle$. Let $C = \langle |0\rangle, |1\rangle, |\bar{0}\rangle, |\bar{1}\rangle\rangle$. We are now ready to give our definition.

Definition 7.2.4. *Let each $|\psi_i\rangle \in C^{\otimes n_i}$ for $1 \le i \le \ell$. Then $\{|\psi_i\rangle \mid 1 \le i \le \ell\}$ is a collection of* delimited orthonormal quantum states *if the following hold.*

(a) There exists a separator state $|\phi\rangle \in \operatorname{span}\{|\bar{0}\rangle, |\bar{1}\rangle\}$ *for some $m \in \mathbb{N}$ such that, for each $1 \le i \le \ell$, $|\psi_i\rangle = |\phi\rangle \otimes \left|\tilde{\psi_i}\right\rangle$ for some state $\left|\tilde{\psi_i}\right\rangle \in \operatorname{span}\{|0\rangle, |1\rangle\}$*

(b) If $n_i = n_j$, then $\langle \psi_i | \psi_j \rangle = [i = j]$

(c) For each $1 \le i \le \ell$, there exists a unitary matrix U_i that can be implemented in time t_i such that $|\psi_i\rangle = U_i |0\rangle$

The idea behind this definition is that given a permutation $\bigotimes_{i=1}^{\ell} \left|\psi_{\pi(i)}\right\rangle$ of the states $\{|\psi_i\rangle \mid 1 \le i \le \ell\}$, we can take advantage of the separator state $|\phi\rangle$ to find the beginning and end of each state $\left|\psi_{\pi(i)}\right\rangle$. We then pad those $\left|\psi_{\pi(i)}\right\rangle$ that contain fewer than $\max_{i=1}^{\ell} n_i$ qudits by appending copies of the state $|\Box\rangle$. This results in a collection of orthonormal states that are all of the same dimension so we can then apply Lemma 7.2.2 to recover $|\pi\rangle$.

Lemma 7.2.5. *Suppose that $\{|\psi_i\rangle \in C^{\otimes n_i} \mid 1 \le i \le \ell\}$ is a collection of delimited orthonormal states where each $|\psi_i\rangle = U_i |0\rangle$ and each unitary U_i can be implemented in t_i time. Let $|\phi\rangle \in C^{\otimes m}$ be a separator state such that there exists a unitary matrix that can be implemented in $O(\log m)$ time that maps $|0\rangle$ to $|\phi\rangle$. Let $n = \sum_{i=1}^{\ell} n_i$ and $t = \sum_{i=1}^{\ell} t_i$. Let $\Delta = n_{\max} - n_{\min}$ where $n_{\max} = \max_{i=1}^{\ell} n_i$ and $n_{\min} = \min_{i=1}^{\ell} n_i$. Then we can implement a unitary V in $4t + O(\ell n \Delta + \ell n (\log^* \log \ell) \log n)$ time such that, for any $\pi \in S_\ell$, $V\left(\bigotimes_{i=1}^{\ell} \left|\psi_{\pi(i)}\right\rangle\right) = \left(\bigotimes_{i=1}^{\ell} |\pi(i)\rangle\right) |0\rangle^{\otimes r}$ where r is chosen so that V is a square matrix.*

[2]A qudit is like a qubit, but can have any number of dimensions. A d-dimensional qudit can always be simulated by $O(\log d)$ qubits.

In the following proof we sometimes say that we set a register to a value for the sake of brevity. Of course, one cannot set an arbitrary state to a second arbitrary state on a quantum computer. However, we only use this terminology for new registers that are initialized to $|0\rangle$.

Proof. To show how to implement V, suppose that the input state is

$$\bigotimes_{i=1}^{\ell} \left|\psi_{\pi(i)}\right\rangle^{Q_i} \tag{7.1}$$

for some unknown $\pi \in S_\ell$ where we have used the letter Q_i to label i^{th} register. Now let $\left|\hat{\psi}_i\right\rangle = |\psi_i\rangle \otimes |\square\rangle^{\otimes(n_{\max}-n_i)}$ for each $1 \leq i \leq \ell$. Let R be a unitary such that $R|0\rangle = |\square\rangle$. Since R is a single qudit operation, it can be implemented in constant time. Letting $\hat{U}_i = U_i \otimes R^{\otimes(n_{\max}-n_i)}$, we see that $\hat{U}_i|0\rangle = \left|\hat{\psi}_i\right\rangle$ and $\hat{U}_i$ can be implemented in $\hat{t}_i = \max\{t_i, O(1)\} = t_i + O(1)$ time.

Our goal is to transform (7.1) into

$$\bigotimes_{i=1}^{\ell} \left|\hat{\psi}_{\pi(i)}\right\rangle^{\hat{Q}_i} \tag{7.2}$$

where we have used the letter $\hat{Q}_i$ instead of Q_i to label the i^{th} register since its size is now potentially different. Since $\left|\hat{\psi}_i\right\rangle \in (\mathbb{C}^5)^{\otimes n_{\max}}$ for each $1 \leq i \leq \ell$, we can apply Lemma 7.2.2 to recover $|\pi\rangle$. The first step in performing this transformation is to compute the index at which the i^{th} state $\psi_{\pi(i)}$ starts for each i. We accomplish this by appending ℓ registers initialized to $|0\rangle$ to obtain the state

$$\left(\bigotimes_{i=1}^{\ell} \left|\psi_{\pi(i)}\right\rangle^{Q_i}\right)\left(\bigotimes_{i=1}^{\ell} |0\rangle^{C_i}\right)$$

We then seek to store the index of the 5-valued qudit that each $\left|\psi_{\pi(i)}\right\rangle$ starts at in register C_i. Clearly, the first qudit of $\left|\psi_{\pi(1)}\right\rangle$ has index 1 in (7.1). To compute the index of the first qudit in $\bigotimes_{i=1}^{\ell} \left|\psi_{\pi(i)}\right\rangle$ for each $2 \leq i \leq \ell$, we proceed as follows.

For each $2 \leq j \leq n$, we compute the number c'_j of qudits of index less than j at which a copy of the separating state $|\phi\rangle$ starts. This is possible because $|\phi\rangle$ is in span $\{|\bar{0}\rangle, |\bar{1}\rangle\}^{\otimes\lceil\log m+1\rceil}$, while each $\left|\psi'_{\pi(i)}\right\rangle$ is in span $\{|0\rangle, |1\rangle\}^{\otimes n_i}$ where $\left|\psi_{\pi(i)}\right\rangle = |\phi\rangle \left|\psi'_{\pi(i)}\right\rangle$. (Recall that $\langle x|\bar{y}\rangle = 0$ for all x and y.) The value c'_j is then stored in a register labelled by C'_j. Since addition of two r-bit numbers can be done in $O(\log^* r)$ time [27, 122] and Toffoli gates with arbitrary numbers of controls can be performed in constant time [57, 27, 121], this takes $O((\log^* \log \ell) \log j)$ time for each j. We then set register i to j if the separating state $|\phi\rangle$ starts at index j and $c'_j = i - 1$. This can be done using a constant number of Toffoli gates. Since the above steps must be done for all $2 \le i \le \ell$ and $2 \le j \le n$, the total time for this step is $O(\ell n (\log^* \log \ell) \log n)$. Letting $k(i)$ be the index of the qudit at which $\left|\psi_{\pi(i)}\right\rangle$ starts for each $1 \le i \le \ell$, the state becomes

$$\left(\bigotimes_{i=1}^{\ell} \left|\psi_{\pi(i)}\right\rangle^{Q_i}\right)\left(\bigotimes_{i=1}^{\ell} |k(i)\rangle^{C_i}\right)$$

after uncomputing and discarding the registers labelled by $C_{(i,j)}$.

Now that the values $k(i)$ are available, we transform each state $\left|\psi_{\pi(i)}\right\rangle$ into $\left|\hat{\psi}_{\pi(i)}\right\rangle$ as follows. First, we append $\ell n_{\max} - n$ qudits initialized to $|\square\rangle$ to obtain the state

$$\left(\bigotimes_{i=1}^{\ell} \left|\psi_{\pi(i)}\right\rangle^{Q_i}\right)\left(\bigotimes_{i=1}^{\ell} |k(i)\rangle^{C_i}\right)\left(\bigotimes_{i=1}^{\ell} \left|\square^{n_{\max}-n_i}\right\rangle^{P_i}\right)$$

in $O(1)$ time.

Then, for each $1 \le i \le \ell$ and each $1 \le j \le n$, we apply controlled swaps conditioned on register C_i being in the state $|j\rangle$ (i.e. $k(i) = j$) to move the qudits in register P_i immediately to the right of register Q_i. This can be done in $O(\Delta)$ time for each i and j. The total time needed for this step is therefore $O(\ell n \Delta)$. The state then becomes

$$\left(\bigotimes_{i=1}^{\ell} \left|\hat{\psi}_{\pi(i)}\right\rangle^{\hat{Q}_i}\right)\left(\bigotimes_{i=1}^{\ell} |k(i)\rangle^{C_i}\right)$$

Note that the P_i registers are no longer present since they have combined with the Q_i registers to create the $\hat{Q}_i$ registers.

We are now almost ready to apply Lemma 7.2.2. First, we need to uncompute and discard the C_i registers. To accomplish this, we note that $k(i)$ is equal to $(i-1)n_{\max} - s(i) + 1$ where

$s(i)$ is the number of $|\square\rangle$ states at qudits with indexes less than $(i-1)n_{\max}+1$. For each $1 \le i \le \ell$, we compute $s(i)$ and store it in a new register labelled by S_i. Computing all of the values $s(i)$ takes $O(\ell(\log^* \log \ell n_{\max}) \log \ell n_{\max})$ time. All of the C_i registers can then be uncomputed in $O(\log^* \log \ell n_{\max})$ time after which they can be discarded. The S_i registers can then be uncomputed and discarded in $O(\ell(\log^* \log \ell n_{\max}) \log \ell n_{\max})$. The total time required for all of the steps in this stage is then $O(\ell(\log^* \log \ell n_{\max}) \log \ell n_{\max})$. After this is done, our state is

$$\bigotimes_{i=1}^{\ell} \left| \hat{\psi}_{\pi(i)} \right\rangle^{\hat{Q}_i}$$

as in (7.2). By applying Lemma 7.2.2, we obtain the state

$$|\pi\rangle = \bigotimes_{i=1}^{\ell} |\pi(i)\rangle$$

in $4t$ time. Appending $|0\rangle^{\otimes r}$, we obtain the desired state

$$|\pi\rangle\, |0\rangle^{\otimes r}$$

Adding up the complexity for each step above, we find that the overall time complexity is $4t + O(\ell n \Delta + \ell n (\log^* \log \ell) \log n)$ as claimed. $\square$

One immediate consequence of Lemma 7.2.5 is that delimited states are weakly orthogonal. This fact yields powerful primitives for state symmetrization that form the core of our quantum algorithm for tree isomorphism.

7.2.3 The algorithm for performing symmetrization

In this subsection, we show an algorithm for symmetrizing collections of symmeterizable states. Since we already know that orthogonal states and delimited states are symmeterizable, this implies algorithms for symmetrizing collections of orthogonal and delimited states as well. Recall the definition of complete left transversals from Section 3.1.

Lemma 7.2.6. *Let $\{|\psi_i\rangle \mid 1 \le i \le \ell\}$ be a collection of G-symmeterizable states where $d_i \in \mathbb{N}$ and $|\psi_i\rangle \in \mathbb{C}^{d_i}$ for each $1 \le i \le \ell$. Assume that the unitary matrix V that maps each permutation $\bigotimes_{i=1}^{\ell} |\psi_{\pi(i)}\rangle$ where $\pi \in G$ to $|\pi\rangle = \bigotimes_{i=1}^{\ell} |\pi(i)\rangle$ can be implemented in time t'. Further suppose that we are given a complete left transversal R_i for each quotient $G_{(\ell,\dots,i)}/G_{(\ell,\dots,i-1)}$ where $i > 1$. Then we can prepare the state*

$$\frac{1}{\sqrt{|G|}} \sum_{\pi \in G} \bigotimes_{i=1}^{\ell} |\psi_{\pi(i)}\rangle$$

in $t' + O(\ell \log \ell)$ time.

First, we note that the requirement that the complete left transversals R_i are given as part of the input can be easily satisfied. This is because there are efficient algorithms [117, 15, 14, 114] for computing a strong generating set (defined in Section 3.6). Given a strong generating set, it is then easy to recover a set of complete transversals. Thus the assumption that we are given the complete left transversals is for convenience only and can be eliminated. Throughout this chapter, we often write expressions of the form $\pi = \pi_1 \cdots \pi_\ell$ where π and π_i are permutations for each $1 \le i \le \ell$. The notation π_i refers to a permutation while $\pi(i)$ refers to the image of i under the permutation π.

Proof. Every element of G can be expressed uniquely as a product $\pi_\ell \cdots \pi_2$ where each $\pi_i \in R_i$. For convenience, we let $G_i = G_{(\ell,\dots,i)}$ for each $1 < i \le \ell$. Our plan is to create a superposition of all permutations over the group G and then transform it into the superposition of all permutations of the state $\bigotimes_{i=1}^{\ell} |\psi_i\rangle$.

Since we cannot directly create the superposition of all permutations in G, the first step is to prepare the superposition

$$\bigotimes_{i=2}^{\ell} \frac{1}{\sqrt{|R_i|}} \sum_{\pi_i \in R_i} |\pi_i\rangle = \frac{1}{\sqrt{|G|}} \sum_{\substack{\pi = \pi_\ell \cdots \pi_2 \\ \pi_i \in R_i}} \bigotimes_{i=2}^{\ell} |\pi_i\rangle$$

of all permutations in G represented in terms of the complete left transversals R_i. This can be done in $O(\log \ell)$ time.

Next, we need to convert each state $\bigotimes_{i=1}^{\ell} |\pi_i\rangle$ in the superposition into the state $|\pi\rangle = \bigotimes_{i=1}^{\ell} |\pi(i)\rangle$ so that we can use the unitary matrix $V^\dagger$ to obtain the desired state. To do this, it suffices to show that we can efficiently compose two permutations since $\pi = \pi_\ell \cdots \pi_2$. Now, if $\rho, \sigma \in S_\ell$, then $(\rho\sigma)(i) = j$ if and only if there exists $k \in [\ell]$ such that $\sigma(i) = k$ and $\rho(k) = j$. This observation yields a quantum circuit for computing the composition of two permutations in $O(\ell)$ time. Thus, we can compose $\ell - 1$ permutations in $O(\ell \log \ell)$ time. Adding another set of registers, this allows us to obtain the state

$$\frac{1}{\sqrt{|G|}} \sum_{\substack{\pi=\pi_\ell\cdots\pi_2 \\ \pi_i \in R_i}} \left(\bigotimes_{i=2}^{\ell} |\pi_i\rangle \right) |\pi\rangle$$

in $O(\ell \log \ell)$ time where $|\pi\rangle = \bigotimes_{i=1}^{\ell} |\pi(i)\rangle$.

Now, if ρ_ℓ and σ_ℓ are distinct elements of R_ℓ, then $\rho_\ell^{-1}\sigma_\ell \notin G_{\ell-1}$ so $\rho_\ell(\ell) \neq \sigma_\ell(\ell)$. Thus, each element of R_ℓ may be identified with a number in $[\ell]$. This implies that if we are given $\pi \in G$, we can compute the unique π_ℓ such that $\pi = \pi_2 \cdots \pi_\ell$ where each $\pi_i \in R_i$ in $O(1)$ time using controlled operations. Thus, we can compute the state

$$\frac{1}{\sqrt{|G|}} \sum_{\substack{\pi=\pi_2\cdots\pi_\ell \\ \pi_i \in R_i}} \left(\bigotimes_{i=2}^{\ell-1} |\pi_i\rangle \right) |\pi\rangle \left|\pi_\ell^{-1}\pi\right\rangle$$

by uncomputing π_ℓ in $O(\log \ell)$ time. By continuing in this manner, we obtain the state

$$\frac{1}{\sqrt{|G|}} \sum_{\substack{\pi=\pi_2\cdots\pi_\ell \\ \pi_i \in R_i}} |\pi\rangle \left|\pi_2 \cdots \pi_\ell^{-1}\pi\right\rangle$$

in $O(\ell \log \ell)$ time.

The permutation $\pi_2 \cdots \pi_\ell^{-1}\pi$ must fix each $2 \leq i \leq \ell$ so it follows that $\pi_2 \cdots \pi_\ell^{-1}\pi$ is the identity permutation. Therefore, we can uncompute $\pi_2 \cdots \pi_\ell^{-1}\pi$ as well to obtain in $O(1)$ time

$$\frac{1}{\sqrt{|G|}} \sum_{\substack{\pi=\pi_2\cdots\pi_\ell \\ \pi_i \in R_i}} |\pi\rangle = \frac{1}{\sqrt{|G|}} \sum_{\pi \in G} |\pi\rangle$$

Finally, we apply the unitary $V^\dagger$ which yields the desired state

$$\frac{1}{\sqrt{|G|}} \sum_{\pi \in G} \bigotimes_{i=1}^{\ell} \left|\psi_{\pi(i)}\right\rangle$$

in time t.

Adding up the complexities for each step, we obtain an overall runtime of $t + O(\ell \log \ell)$ as claimed. $\square$

By combining Lemma 7.2.6 with Corollary 7.2.3 and Lemma 7.2.5, we obtain two useful corollaries.

Corollary 7.2.7. *Let $\{|\psi_i\rangle \in \mathbb{C}^d \mid 1 \leq i \leq \ell\}$ be a collection of orthonormal states of dimension d where each $|\psi_i\rangle = U_i |0\rangle$ and each U_i can be implemented in t_i time. Let G be a permutation group of degree ℓ and assume that we are given a complete left transversal R_i for each quotient $G_{(\ell,\ldots,i)}/G_{(\ell,\ldots,i-1)}$ where $i > 1$. Then we can prepare the state*

$$\frac{1}{\sqrt{|G|}} \sum_{\pi \in G} \bigotimes_{i=1}^{\ell} |\psi_{\pi(i)}\rangle$$

in $4t + O(\ell \log \ell)$ time where $t = \sum_{i=1}^{\ell} t_i$.

The next corollary forms the core of our quantum algorithm for tree isomorphism.

Corollary 7.2.8. *Suppose that $\{|\psi_i\rangle \in C^{\otimes n_i} \mid 1 \leq i \leq \ell\}$ is a collection of delimited orthonormal states where each $|\psi_i\rangle = U_i |0\rangle$ and each unitary U_i can be implemented in t_i time. Let $|\phi\rangle \in C^{\otimes m}$ be a separator state such that there exists a unitary matrix that can be implemented in $O(\log m)$ time that maps $|0\rangle$ to $|\phi\rangle$. Let $n = \sum_{i=1}^{\ell} n_i$ and $t = \sum_{i=1}^{\ell} t_i$. Let $\Delta = n_{\max} - n_{\min}$ where $n_{\max} = \max_{i=1}^{\ell} n_i$ and $n_{\min} = \min_{i=1}^{\ell} n_i$. Let G be a permutation group of degree ℓ and assume that we are given a complete left transversal R_i for each quotient $G_{(\ell,\ldots,i)}/G_{(\ell,\ldots,i-1)}$ where $i > 1$. Then we can prepare the state*

$$\frac{1}{\sqrt{|G|}} \sum_{\pi \in G} \bigotimes_{i=1}^{\ell} |\psi_{\pi(i)}\rangle$$

in $4t + O(\ell n \Delta + \ell n (\log^* \log \ell) \log n)$ time.

7.3 A quantum algorithm for tree isomorphism

In this section, we give our algorithm for tree isomorphism and analyze its complexity. We use 5-valued qudits with the same notational conventions as in Subsection 7.2.2. The basis state $|\square\rangle$ is not used here as it is reserved for the internal implementation of Corollary 7.2.8. The idea behind the algorithm is to recursively compute a complete invariant state $|T_i\rangle$ for each subtree T_i rooted at a child of the root. This collection of states is not delimited. However, we can modify it to make it delimited. An application of Corollary 7.2.8 then yields a complete invariant state $|T\rangle$ for the rooted tree T.

Theorem 7.3.1. *Let T be a rooted tree with n nodes. Then we can compute in $O(n^5)$ time a complete invariant state $|T\rangle$ such that if T' is another rooted tree with n nodes, then $\langle T|T'\rangle = [T \cong T']$.*

Proof. We show by induction that there is a unitary matrix U such that $U|0\rangle = |T\rangle$.

If $n = 1$, then we define $|T\rangle = |0\rangle$. Otherwise, let $T_1, \ldots, T_\ell$ be the subtrees of T rooted at the children of the root. We recursively compute a complete invariant state $|T_i\rangle = U_i|0\rangle$ for each T_i. Let m be the depth of T and let $|\phi\rangle$ be the state $|\bar{m}\rangle$ with the correct number of $|\bar{0}\rangle$ qudits prepended so that it uses exactly $\lceil \log n + 1 \rceil$ qudits. We then prepend $|\phi\rangle$ to each invariant state $|T_i\rangle$. Since the state $|\phi\rangle$ is not used in the states $|T_i\rangle$, this almost gives us a collection of delimited states. However, it may be the case that $T_i \cong T_j$ for some $i \neq j$, in which case we have $|T_i\rangle = |T_j\rangle$, so that $|T_i\rangle$ and $|T_j\rangle$ are not orthogonal.

We can correct this by prepending the state $|k(i)\rangle$ to each state $|T_i\rangle$ where $k(i)$ is the number of trees $T_j \cong T_i$ where $j < i$. We accomplish this as follows. For reasons of efficiency that will become clear in the complexity analysis, we handle the subtrees that have a number of nodes distinct from all others separately. To this end, we let v be the vector of all indexes $1 \leq i \leq \ell$ such that $|\{T_j \mid |T_i| = |T_j|, 1 \leq j \leq \ell\}| = 1$ listed in order of increasing $|T_i|$.

Let CP be the unitary matrix that acts on a pair of registers by adding one to the second register if all of the qudits in the first register are in the state $|0\rangle$. Then $(U_i \otimes I) \cdot \mathrm{CP} \cdot (U_i^\dagger \otimes I)$ adds one to the second register if the first register is in the state $|T_i\rangle$. For each $i \notin v$, we apply this operation to each $|T_j\rangle$ such that $j < i$ and T_j and T_i have the same number of nodes. In this way, each state $|T_i\rangle$ with $i \notin v$ is transformed into $|\phi\rangle \left|\hat{k}(i)\right\rangle |T_i\rangle$ where $\left|\hat{k}(i)\right\rangle$ is the state $|k(i)\rangle$ padded with the right number of zeros to ensure that it uses exactly $\lceil \log n \rceil$ qudits. Each state $|T_i\rangle$ with $i \in v$ is transformed into $|\phi\rangle |0\rangle^{\lceil \log n \rceil - 1} |1\rangle |T_i\rangle$. This gives us a collection of delimited orthonormal states.

We apply Corollary 7.2.8 to obtain the state

$$\left|\hat{T}\right\rangle = \frac{1}{\sqrt{(\ell - |v|)!}} \sum_{\pi \in S_{(\ell - |v|)}} \bigotimes_{i \notin v} \left(|\phi\rangle \left|\hat{k}(\pi(i))\right\rangle \left|T_{\pi(i)}\right\rangle \right)$$

We then define

$$|T\rangle = \left(\bigotimes_{i \in v} |\phi\rangle |0\rangle^{\lceil \log n \rceil - 1} |1\rangle |T_i\rangle \right) \left|\hat{T}\right\rangle$$

Since each state $|T_i\rangle$ is a complete invariant for $|T_i\rangle$, it follows by induction that $|T\rangle$ is a complete invariant for $|T\rangle$. This step takes time $\sum_{i \in v} t_i + 5 \sum_{i \notin v} t_i + O(\ell n \Delta + \ell n (\log^* \log \ell) \log n)$. Accounting for the time required for the other two steps, we find that the total time required to prepare $|T\rangle$ is

$$\sum_{i \in v} t_i + \sum_{i \notin v} (5 + 2\ell_i) t_i + O(n^3) \tag{7.3}$$

where each ℓ_i of the number of subtrees T_j with $j < i$ that have the same size as T_i.

All that remains is to analyze the complexity. Let $f(n)$ denote the time required to compute $|T\rangle$ when T has n nodes. We will prove by induction on n that $f(n) \leq c_1 n^{c_2}$ where c_1 and c_2 are positive constants to be determined. Before we can show the recurrence, we need to introduce some notation. Let $P(n-1)$ denote the set of all vectors of natural numbers that sum to $n-1$. For any $\vec{n} \in P(n-1)$, let $\kappa_i(\vec{n}, x)$ denote the number of elements of $\vec{n}$ at indexes less than i that are equal to $x \in \mathbb{N}$. Let $V(\vec{n})$ denote the set of indexes $1 \leq i \leq |\vec{n}|$ such that n_i occurs only at once index in $\vec{n}$.

Then we have

$$\begin{aligned}
f(n) &\leq \max_{\vec{n}\in P(n-1)} \left\{ \sum_{i\in V(\vec{n})} f(n_i) + \sum_{i\notin V(\vec{n})} (5+2\kappa_i(\vec{n},n_i))f(n_i) \right\} + c_3n^3 \text{ for some } c_3 > 0 \\
&\leq \max_{\vec{n}\in P(n-1)} \left\{ \sum_{i\in V(\vec{n})} f(n_i) + \sum_{i\notin V(\vec{n})} (5+2\kappa_i(\vec{n},n_i))f(n_i) \right\} + c_3n^3 \\
&\leq \max_{\vec{n}'\in P(n-1)} \left\{ \sum_{k\in\vec{n}'} \max_{\substack{a,b\in\mathbb{N}\\ ab=k}} \{g(a)f(b)\} \right\} + c_3n^3
\end{aligned}$$

where

$$g(a) = \begin{cases} 1 & \text{if } a = 1 \\ a^2+4a & \text{if } a \geq 2 \end{cases}$$

We claim that $g(a)f(b) \leq c_1k^{c_2}$ where $k = ab$ if $c_2 \geq 5$. In the case where $a = 1$, $g(a)f(b) = f(k)$ and $f(k) \leq c_1k^{c_2}$ by induction. Therefore, we assume that $a \geq 2$. Then

$$\begin{aligned}
g(a) &= a^2+4a \\
&< 5a^2 \\
&< a^5
\end{aligned}$$

Therefore,

$$\begin{aligned}
g(a)f(b) &< a^5f(b) \\
&\leq c_1a^5\left(\frac{k}{a}\right)^{c_2} \\
&\leq c_1k^{c_2}
\end{aligned}$$

if $c_2 \geq 5$, as we claimed. Consequently, since $(n-1)^c \leq n^c - \Omega(n^{c-1})$ for $c \geq 1$, we have

$$\begin{aligned}
f(n) &< c_1(n-1)^{c_2} + c_3n^3 \\
&\leq c_1n^{c_2} + c_3n^3 - c_1c_4n^{c_2-1} \text{ for some } c_4 > 0 \\
&\leq c_1n^{c_2}
\end{aligned}$$

if $c_1 \geq c_3/c_4$. Since $c_1 > 0$ and $c_2 > 0$ are constants that we can choose while c_3 is an absolute constant and c_4 is a constant that depends only on c_2, it follows that $f(n) = O(n^5)$. $\square$

7.4 Conclusion

In this chapter, we showed that complete invariant states for trees can be prepared in $O(n^5)$ time on a quantum computer. Our primitive for symmetrizing collections of delimited orthonormal states seems powerful and may be of independent interest.

A few open problems still remain. First, it seems unlikely that $\Omega(n^5)$ time is really necessary for preparing complete invariant states for trees. Our goal was merely to obtain polynomial time and we did not attempt to optimize the polynomial. Our analysis is probably not tight and can likely be modified to get a better upper bound. However, preparing complete invariant states for trees in nearly linear time (which seems like the correct runtime) will likely require enhancements to the underlying algorithm as well.

A second question is whether the methods developed in this chapter can be leveraged to test isomorphism of more complicated types of graphs. Of particular interest are graphs that generalize trees, such as the cone graphs.

Part II

ISOMORPHISM TESTING

Chapter 8
THE COLOR AUTOMORPHISM PROBLEM

8.1 Introduction

In this chapter, we survey a group-theoretic problem known as the *color automorphism problem* (which we will define later). This problem is important for several reasons. As we shall see, algorithms for color automorphism can be used to obtain algorithms for testing isomorphism of bounded-degree graphs, which are used in our group isomorphism algorithms in Chapters 10 – 12. (However, these later chapters only depend on the statements of Theorems 8.4.5 and 8.4.7 and do not rely on the proof methods introduced in the present chapter.) Together with Zemlyachenko's lemma (which provides a method for reducing the degree of a graph), an algorithm for bounded-degree graph isomorphism is one of the two main ingredients in the best algorithm for general graph isomorphism that is currently known.

The first algorithm for the color-automorphism problem was devised by Luks [76], and was efficient enough to yield the first algorithm for testing isomorphism of graphs of constant degree. Subsequently, Luks improved [18, 16] his algorithm to the point where it implies that isomorphism of graphs of degree at most d can be tested in $n^{O(d/\log d)}$ time.

In this chapter, we cover algorithms for the color-automorphism problem and their application to the graph isomorphism problem. In Section 8.2, we cover the basics of group actions. We discuss two algorithms for permutation groups that are needed in the color automorphism algorithm in Section 8.3. We describe Luks' original algorithm for color automorphism [76] and subsequent improvements [16] in Section 8.4. Because it is required for Zemlyachenko's lemma which is needed for the algorithm for general graph isomorphism, we describe the Weisfeiler-Lehman (WL) algorithm in Section 8.5. We then cover Zemlyachenko's lemma and the best algorithm [18, 16] known for general graph isomorphism in

Section 8.6. We conclude with open questions and possibilities for further improvements in Section 8.7.

8.2 Group actions

In this section, we cover the basics of *group actions*, which are a generalization of permutation groups. For basic background on groups and permutation groups, see Chapter 3.

Given a group G and a set Ω, we say that G acts on Ω if there is a homomorphism $\phi : G \to \mathrm{Sym}(\Omega)$ such that each $g \in G$ acts on Ω by the associated permutation $\phi(g)$. For $g \in G$ and $\alpha \in \Omega$, we denote the action of g on α as $g(\alpha)$ or $g\alpha$. An action is *faithful* if $\ker \phi = \{1\}$ and in this case G is isomorphic to a subgroup of $\mathrm{Sym}(\Omega)$. An action is *transitive* if for all $\alpha, \beta \in \Omega$ there exists $g \in G$ such that $g\alpha = \beta$; if an action is intransitive then Ω may be partitioned into a set of *orbits* $\Omega_1, \ldots, \Omega_m$ such that the restriction of the action of G to any orbit is transitive (we remark here that each $\Omega_i = G\alpha_i = \{g\alpha_i \mid g \in G\}$ for any $\alpha_i \in \Omega_i$). For transitive G, a nonempty subset $\Delta \subseteq \Omega$ is called a *block* if for all $g \in G$, $\Delta \cap g\Delta$ is either empty or equal to Δ. In this case, we call the set of images of Δ under the action of G (which partitions Ω) a *block system.* A *trivial block system* is the unit partition or the discrete partition. A block system is *minimal* if no partition of Ω which more coarse is also a nontrivial block system. (In other words, one cannot join blocks to obtain another nontrivial block system.) An action is *primitive* if it is transitive and has no nontrivial block system. We see that if an action of G is transitive then its action on any minimal block system is primitive. Moreover, if H is the subgroup of G which stabilizes some minimal block system then G/H is primitive and acts faithfully on the blocks.

When the action in question is obvious from the context, we shall sometimes refer to transitivity and other properties of group actions as if they were properties of the group. For example, for a permutation group G on Ω we might say that G is transitive. This would mean that the action of applying the permutations to Ω is transitive.

We can define stabilizers for group actions in the same way that we did for permutation groups in Section 3.6. For $\alpha \in \Omega$, we denote the subgroup of G which fixes α by G_α. For a subset $\Delta \subseteq \Omega$, the *setwise stabilizer* of Δ (denoted G_Δ) is the subgroup which sends every element of Δ back into Δ. A subset $\Delta \subseteq \Omega$ is called G-stable if $G = G_\Delta$. The *pointwise stabilizer* of Δ (denoted $G_{(\Delta)}$) is the subgroup of G which fixes every element of Δ.

Let us choose $\alpha \in \Omega$. There is a close relationship between the set $\mathcal{B}$ of all blocks for a group action and the set $\mathcal{S}$ of all subgroups of G that contain G_α.

Theorem 8.2.1. *(cf. [40]) The map $\Psi : \mathcal{B} \to \mathcal{S} : \Delta \mapsto G_{\{\Delta\}}$ is an order-preserving bijection (with respect to set inclusion) and its inverse is $\Phi : \mathcal{S} \to \mathcal{B} : H \mapsto Ha$.*

This implies that G is primitive if and only if G_α is a maximal subgroup of G. It follows that every primitive p-group is of order p and that every primitive Abelian group is cyclic of prime order.

8.3 Permutation-group algorithms

The color-automorphism algorithms that we will discuss in this chapter rely on two basic permutation group algorithms. The first of these is capable of computing the kernel of a homomorphism $\phi : G \to H$ between two permutation groups. First, we remark that ϕ can be compactly specified by its action on a generating set S of G. In order to compute $\ker \phi$, we consider the subgroup $K = \langle \{(\pi, \phi(\pi))\}_{\pi \in S} \rangle$ of S_{m+n} where m and n are the degrees of G and H. By computing the pointwise stabilizer $K_{(m+n,\ldots,m+1)}$, we obtain the group $\ker \phi \times \{\iota\}$ from which we can easily compute $\ker \phi$.

We shall also need an algorithm for computing a minimal block system of a transitive permutation group $G \leq \mathrm{Sym}(\Omega)$. This may be done recursively by giving an algorithm (cf. [76, 18]) for computing the smallest block in which a pair $\alpha, \beta \in \Omega$ are contained. To compute the smallest such block, we consider the graph whose nodes correspond to Ω and whose edges correspond to the images under G of the set $\{\alpha, \beta\}$. The smallest block containing $\{\alpha, \beta\}$ is then the connected component in which $\{\alpha, \beta\}$ is contained. We consider all possible such pairs and choose one which results in a nontrivial block system. We then continue the process recursively on this block system; the result is a minimal block system for G.

8.4 Bounded-degree graph isomorphism

In this section, we will show how to decide isomorphism of graphs of bounded degree [7, 76, 18, 16]. The algorithm works by computing the subgroup $\mathrm{Aut}_e(X)$ which setwise fixes the edge e of the connected graph X. Since we represent all groups in terms of their generating sets, computing a group means we compute a generating set for that group. This suffices to decide isomorphism of connected graphs of bounded degree. To show this, consider two graphs X and Y and choose edges $e_X = \{a_1, b_1\}$ and $e_Y = \{a_2, b_2\}$ in each of these graphs. We delete these edges from the graphs and add two new nodes c_1 and c_2; we then connect each a_i and b_i to c_i and draw an edge between c_1 and c_2. Denoting the resulting graph by Z, we compute $\mathrm{Aut}_{\{c_1,c_2\}}(Z)$. The original graphs X and Y are isomorphic if and only if every generating set of this group contains a permutation which swaps c_1 and c_2 for some choice of the edges e_X and e_Y. By fixing the choice of e_X and repeating the reduction for each possible choice of e_Y, we can decide isomorphism of connected graphs of bounded degree. It is easy to see that this allows us to decide isomorphism of arbitrary graphs of bounded degree since we can split them into their connected components and determine which components are isomorphic. We shall therefore focus on computing the group $\mathrm{Aut}_e(X)$ where X is a connected graph of bounded degree.

The algorithm for testing isomorphism of graphs of bounded degree is based on the tower of subgroups approach that was introduced by Babai [8] in his algorithm for deciding isomorphism of graphs of bounded color class. The algorithm for bounded-degree graphs is particularly elegant when formulated in terms of the *color automorphism problem* [76]. Here, we are given a set Ω and a permutation group G. Each element of Ω has an associated color and our goal is to compute the subgroup $C_\Omega(G)$ of G that maps each element of Ω to some other element which has the same color. It is clear that this problem is at least as hard as *graph automorphism* where one must compute the group of automorphisms of the graph. Since it is known that the automorphism problem is equivalent to the isomorphism problem under Turing reductions [79] (cf. [23, 56]), we see that the color automorphism problem is GI-hard. However, this does not seem to be useful for obtaining an efficient algorithm for graph isomorphism since the color automorphism problem on this group appears to difficult. In fact, it is known that at least one version of the corresponding canonization problem is NP-hard [18].

8.4.1 Reduction to the color automorphism problem

The key contribution of Luks' paper [76] is his Turing reduction from testing isomorphism of graphs of degree at most d to color automorphism problems for groups in the class Γ_{d-1} on sets of size at most $\binom{n}{d}$. Here, Γ_d is the set of all permutation groups whose non-Abelian composition factors are isomorphic to subgroups of S_d. This class of groups is relevant because of the following result.

Theorem 8.4.1 (Babai, Cameron and Pálfy [10]). *Let G be a primitive permutation group of degree n in Γ_d. Then $|G| \leq n^{w(d)}$ where $w(d) = O(d \log d)$.*

Let $X = (V, E)$ be a connected graph of degree at most d. As we shall see in the next subsection, this result allows us to solve the color automorphism problem for a permutation group in Γ_d acting on a set of size n in $n^{w(d-1)+O(1)}$ time. For now, we reduce computing $\mathrm{Aut}_e(X)$ to a linear number of color automorphism problems for groups in Γ_{d-1} acting on sets of size at most $\binom{n}{d}$.

We remark that [10] was published after Luks' original result, which relied on different techniques. However, we prefer the use of Theorem 8.4.1 since it yields a more efficient and less complicated algorithm. We now show Luks' reduction from computing $\mathrm{Aut}_e(X)$ to the color automorphism problem. Our presentation also uses ideas from [7].

We define $X_r = (V_r, E_r)$ to be the subgraph of X which consists of those nodes and edges that are located on paths of length at most r that include e. However, the automorphism groups of the graphs X_r still do not have enough structure so we use another trick. Let $Y_{r+1} = (V_{r+1}, F_{r+1})$ be the graph obtained from X_{r+1} by removing those edges between nodes in $V_{s+1} \setminus V_s$ for $1 \leq s \leq r$. We let $Y = (V, F) = Y_t$ where t is the largest value such that $X_t \neq X_{t-1}$. We can think of computing $\mathrm{Aut}_e(X)$ as a color automorphism problem for the group $\mathrm{Aut}_e(Y)$ acting on the set of all two element subsets of V where each subset is colored "edge" if it corresponds to an edge in $E \setminus F$ and "non-edge" otherwise. Thus, we can compute $\mathrm{Aut}_e(X)$ efficiently from $\mathrm{Aut}_e(Y)$ assuming that $\mathrm{Aut}_e(Y)$ is in Γ_{d-1} (which we show later).

Our plan is to show how to compute $\mathrm{Aut}_e(Y_{r+1})$ given $\mathrm{Aut}_e(Y_r)$. To start, we note that Y_1 consists of just the edge e so $\mathrm{Aut}_e(Y_1)$ consists of the identity and the permutation which swaps the endpoints of e. Then clearly, $\mathrm{Aut}_e(Y_1) \in \Gamma_{d-1}$. To compute $\mathrm{Aut}_e(Y_{r+1})$ given $\mathrm{Aut}_e(Y_r)$, we consider the homomorphism $\phi_r : \mathrm{Aut}_e(Y_{r+1}) \to \mathrm{Aut}_e(Y_r)$ which outputs the restriction of each automorphism in $\mathrm{Aut}_e(Y_{r+1})$ to V_r. The kernel of ϕ_r is clearly the subgroup of $\mathrm{Aut}_e(Y_{r+1})$ that pointwise fixes V_r. Let $\mathcal{A}_r$ be the set of all subsets of V_r of at most d elements and define $\rho_r : V_{r+1} \setminus V_r \to \mathcal{A}_r$ to map each vertex in $V_{r+1} \setminus V_r$ to the set of nodes to which it is connected in Y_{r+1}. Since Y_{r+1} has no edges connecting the nodes in $V_{r+1} \setminus V_r$ to each other, an automorphism of Y_{r+1} that fixes V_r can map a node in $V_{r+1} \setminus V_r$ to any other node with the same set of neighbors. Then $\ker \phi_r$ is the direct product $\times_{A \in \mathcal{A}_r} \mathrm{Sym}(\rho_r^{-1}(A))$, which we can easily compute. Now by induction, $\mathrm{Aut}_e(Y_r)$ is in Γ_{d-1} so $\mathrm{Im}\, \phi_r$ is also in Γ_{d-1}. Since $|\rho^{-1}(A)| \leq d - 1$ for any $A \in \mathcal{A}_r$, $\ker \phi_r$ is in Γ_{d-1} which implies that $\mathrm{Aut}_e(Y_{r+1})$ is in Γ_{d-1} since $\mathrm{Aut}_e(Y_{r+1})/\ker \phi_r \cong \mathrm{Im}\, \phi_r$ by the first isomorhpism theorem.

To finish computing $\mathrm{Aut}_e(Y_{r+1})$, we note that $\mathrm{Im}\, \phi_r$ is the subgroup of automorphisms of Y_r that can be extended to automorphisms of Y_{r+1}. Our next goal is to compute $\mathrm{Im}\, \phi_r$. By definition, all edges in $F_{r+1} \setminus F_r$ are from V_r to $V_{r+1} \setminus V_r$. Thus, an automorphism $\pi \in \mathrm{Aut}_e(Y_r)$ extends to an automorphism of Y_{r+1} if and only if it sends each $A \in \mathcal{A}_r$ to some $B \in \mathcal{A}_r$ such that the number of nodes in $V_{r+1} \setminus V_r$ that have A as their neighborhood in Y_{r+1} is equal the number of nodes that have B as their neighborhood. Equivalently, π must stabilize the sets $\mathcal{A}_{r,s} = \{A \in \mathcal{A}_r \mid |\rho^{-1}(A)| = s\}$ for all $1 \leq s \leq d-1$. If we think of $\mathrm{Aut}_e(Y_r)$ as acting on $\mathcal{A}_r$ and color each $A \in \mathcal{A}_r$ according to the set $\mathcal{A}_{r,s}$ in which it is contained, then this is a color automorphism problem on a set of size at most $\binom{n}{d}$ for a group in Γ_{d-1} so we can compute

$\operatorname{Im}\phi_r$. For each generator $\pi \in \operatorname{Im}\phi_r$, we extend π to $\sigma \in \operatorname{Aut}_e(Y_{r+1})$ as follows. For each $A \in \mathcal{A}_r$ and $\pi[A]$, we consider the nodes $\rho^{-1}(A)$ and $\rho^{-1}(\pi[A])$ that have A and $\pi[A]$ as their neighborhoods in Y_{r+1}. If $A = \pi[A]$ then we define σ on $\rho^{-1}(A) = \rho^{-1}(\pi[A])$ by an arbitrary permutation. Otherwise, $\rho^{-1}(A)$ and $\rho^{-1}(\pi[A])$ are disjoint and we define σ on $\rho^{-1}(A)$ by an arbitrary bijection to $\rho^{-1}(\pi[A])$. We continue constructing the extension σ of π in this way by considering subsets $A \in \mathcal{A}_r$ until σ is defined on all of V_{r+1}. It is clear that $\phi_r(\sigma) = \pi$.

The preimages of different generators of $\operatorname{Im}\phi_r$ are representatives of cosets which generate the factor group $\operatorname{Aut}_e(Y_{r+1})/\ker\phi_r$. It follows that the generators of $\ker\phi_r$ together with a preimage of each generator of $\operatorname{Im}\phi_r$ generates $\operatorname{Aut}_e(Y_{r+1})$. This allows us to compute $\operatorname{Aut}_e(Y_{r+1})$ given $\operatorname{Aut}_e(Y_r)$. Thus, we can compute $\operatorname{Aut}_e(Y)$ by induction. We then compute $\operatorname{Aut}_e(X)$ from $\operatorname{Aut}_e(Y)$ as described above.

8.4.2 An algorithm for the color automorphism problem

In this subsection, we present Luks' algorithm for the color automorphism problem. It is useful to consider a slightly more general version of the color automorphism problem. Here, we are given a coset σG where $\sigma \in \operatorname{Sym}(\Omega)$ and $G \leq \operatorname{Sym}(\Omega)$ and a G-stable subset $\Delta \subseteq \Omega$. The goal is to compute $C_\Delta(\sigma G) = \{\pi \in \sigma G \mid \forall \alpha \in \Delta, \pi\alpha \sim \alpha\}$ where $\alpha \sim \beta$ means that $\alpha, \beta \in \Omega$ have the same color. (We remark that a coset σG can be represented efficiently by a representative σg and a generating set for G.) It is easy to show that $C_\Delta(\sigma G)$ is either empty or a left coset of $C_\Delta(G)$. This means that the output $C_\Delta(\sigma G)$ can always be represented compactly. It follows from the definition that if $\Omega = \Delta_1 \cup \Delta_2$, then $C_\Omega(\sigma G) = C_{\Delta_1}(C_{\Delta_2}(\sigma G))$. Also, if $\sigma G = \bigcup_i \sigma\tau_i H$ where each $\tau_i \in G$ and $H \leq G$ then $C_\Delta(\sigma G) = \bigcup_i C_\Delta(\sigma\tau_i H)$.

Given an instance $C_\Delta(\sigma G)$ of the color automorphism problem, we first check if $|\Delta| = 1$. In this case, we return σG if σ respects the color of Δ and $\emptyset$ otherwise (recall that Δ is G-stable). If $|\Delta| > 1$, we test if G is intransitive on Δ. In this case, we can partition Δ into two nonempty G-stable subsets Δ_1 and Δ_2. We then break the problem up into two smaller problems by writing $C_\Delta(\sigma G) = C_{\Delta_1}(C_{\Delta_2}(\sigma G))$. The third case occurs when G is transitive on Δ. In this case, we compute a minimal block system $\Delta_1, \ldots, \Delta_m$ for the action of G on

Δ. We then compute the subgroup H that setwise stabilizes each block Δ_i. We note that in general, computing the setwise stabilizer is GI-hard. However, in this case H is the kernel of the homomorphism $\phi : G \to \mathrm{Sym}(\{\Delta_1, \ldots, \Delta_m\})$ which maps each element of G to its induced action on the blocks and we have already explained how to compute the kernels of homomorphisms of permutation groups in Section 8.3. We proceed by computing a complete set of representatives $\tau_1, \ldots, \tau_k$ for the cosets in G/H. Then $C_\Delta(\sigma G) = \bigcup_{i=1}^k C_\Delta(\sigma\tau_i H)$; however, we need to express $C_\Delta(\sigma G)$ as a subcoset of σG. We can do this by computing each $C_\Delta(\sigma\tau_i H) = \rho_i C_\Delta(H)$. Since we already argued that $C_\Delta(\sigma G)$ is a subcoset of σG, it follows that $C_\Delta(\sigma G) = \rho_1 \langle C_\Delta(H), \left\{\rho_1^{-1}\rho_i \;\middle|\; 2 \le i \le k\right\}\rangle$, which has the desired form.

Now let us analyze the running time in terms of the size n of the set Ω and the class Γ_d which contains the group G. The only issue is that when G is transitive on Δ, the size of the set in the sub-problems is not immediately reduced. However, since in that case each block Δ_i is stabilized by H, the case of the algorithm that tests for intransitivity will break each problem $C_\Delta(\sigma\tau_i H)$ into m smaller problems on each of the blocks. Thus, we have that for each n, at least one of the following inequalities is satisfied.

$$T(n) \le \sum_{i=1}^{m} T(n_i) + \mathsf{poly}(n) \text{ where } \{n_1, \ldots, n_m\} \text{ is an integer partition of } n$$

$$T(n) \le m^{w(d-1)+1} T(n/m) + \mathsf{poly}(n) \text{ where } m \text{ is a divisor of } n$$

One can easily verify that $T(n) = n^{w(d-1)+O(1)}$ satisfies both inequalities and is therefore an upper bound on the runtime.

Theorem 8.4.2 (Babai and Luks [18]). *Let G be a permutation group in Γ_d acting on a colored set Ω of size n. Then for any $\sigma \in \mathrm{Sym}(\Omega)$, $C_\Omega(\sigma G)$ can be computed in $n^{w(d-1)+O(1)}$ time.*

Combining this result with Luks' reduction from bounded-degree graph isomorphism to the color automorphism problem, we obtain the following theorem.

Theorem 8.4.3 (Luks [76] (cf. [7])). *Isomorphism of graphs of degree at most d can be tested in $n^{O(d^2 \log d)}$ time.*

In fact this result can be slightly generalized. For this, we need to review some basic definitions for graphs. A *colored graph* is a graph that associates each vertex with a given color. Two colored graphs are isomorphic if there is a bijection between their vertex sets that respects the edges and maps each node to a node of the same color. Since the set of nodes has size n, one can handle colored graphs as well by simply solving an additional color automorphism problem. This does not increase the runtime.

Theorem 8.4.4 (Luks [76] (cf. [7])). *Isomorphism of colored graphs of degree at most d can be tested in $n^{O(d^2 \log d)}$ time.*

8.4.3 Faster algorithms for graphs of bounded degree

Although Theorem 8.4.4 is impressive, it is desirable to obtain a more efficient algorithm. The $d \log d$ factor in the exponent comes from the color automorphism algorithm while the second d factor comes from solving the color automorphism problem on sets of size at most n^d. Thus, if we could improve the reduction to color automorphism to only use sets of size $n^{O(1)}$, we would obtain an $n^{O(d \log d)}$ algorithm. This can be accomplished using a clever trick [18]. Let us define the graphs X_r and Y_r as before. We note that the color automorphism problem that we must solve to compute $\mathrm{Aut}_e(X)$ given $\mathrm{Aut}_e(Y)$ is on a set of size at most n^2. The sets of size at most n^d arise when we compute $\mathrm{Aut}_e(Y_{r+1})$ from $\mathrm{Aut}_e(Y_r)$ via a color automorphism problem on all d element subsets of V_r. We accomplish this using a different reduction to the color automorphism problem.

Intuitively, the proof works by noting that we can think of an automorphism of a graph either as a permutation of the vertices that respects the edges or a permutation of the edges that induces a well-defined permutation of the vertices. The improved reduction works by lifting to an action on the edges that contains all of the automorphisms and then selecting only those permutations of the edges that correspond to permutations of the vertices.

More formally, for each node $x \in V_r$, we define $d_r(x)$ to be the number of edges from x to nodes in $V_{r+1} \backslash V_r$. We see that a necessary (but not sufficient) condition for an automorphism $\pi \in \mathrm{Aut}_e(Y_r)$ to extend to an automorphism of Y_{r+1} is that $d_r(\pi(x)) = d_r(x)$ for all $x \in V_r$. Thus, we start by computing the subgroup H_r of $\mathrm{Aut}_e(Y_r)$ that respects d_r; note that this is a color automorphism problem on a set of size at most n. We then extend H_r to a group K_r that acts on the edges in Y_{r+1} from V_r to $V_{r+1} \setminus V_r$ by allowing edges that share a point in V_r to be permuted in all possible ways (note that since we already restricted to only those automorphisms that respect the degrees of the nodes, this group is in Γ_{d-1}). Moreover, K_r contains all permutations of the edges that correspond to automorphisms of Y_{r+1}. The problem now is that we cannot immediately map K_r back to a group of permutations of V_{r+1}. We resolve this by computing the subgroup L_r of K_r that maps every pair of edges from V_r to $V_{r+1} \setminus V_r$ that have a common endpoint in $V_{r+1} \setminus V_r$ to another pair of edges with the same property. We do this by considering the action of K_r on all pairs of edges from V_r to $V_{r+1} \setminus V_r$; we then color each pair that shares a common endpoint in $V_{r+1} \setminus V_r$ "red" and all other pairs "blue." Since the set for this color automorphism problem is of size at most n^4, we can find L_r efficiently. By definition, L_r can also be thought of as an action on V_{r+1} and it is easy to show that $L_r = \mathrm{Aut}_e(Y_{r+1})$. Since all of the sets on which we solve the color automorphism problem now have size at most n^4, we obtain the following theorem.

Theorem 8.4.5 (Babai and Luks [18]). *Isomorphism of colored graphs of degree at most d can be tested in $n^{O(d \log d)}$ time.*

8.4.4 Further speedups

In this subsection, we shall sketch how to obtain the $n^{O(d/\log d)}$ algorithm [16] for graphs of degree at most d (which is the best result to date). The *socle* (denoted $\mathrm{soc}(G)$) of G is the subgroup generated by the minimal normal subgroups of G. Let G be a primitive group of degree n which is in Γ_d and consider $\mathrm{soc}(G)$. Using the classification of finite simple groups, one can show that either (a) G has a Sylow p-subgroup P of index at most $n^{O(d/\log d)}$ or

(b) the socle is isomorphic to a direct product of alternating groups of degree at most d. In the first case, the Sylow p-subgroup can be found efficiently using an algorithm due to Kantor [60] or a more specialized algorithm from Luks' original paper [76]. Once this group has been obtained, one can proceed using techniques similar to those discussed previously. The more difficult case is when the socle is a direct product of alternating groups. We shall describe the speedup only in the special case where the socle is isomorphic to a single alternating group. We shall also assume for simplicity that $\mathrm{soc}(G)$ acts as the alternating group on Ω. This is not true in general since an isomorphism between two groups need not respect their permutation domains (this is the difference between an isomorphism and a *permutation isomorphism*). However, both of these assumptions can be eliminated using more complex versions of the techniques described here [16].

The first step is to pass to the socle of G. This can be done at essentially zero cost since one can show [16] that the index of the socle in G is at most $n^{O(\log d)}$. We know that the socle is transitive since G is primitive. We divide Ω into two halves Δ_1 and Δ_2 arbitrarily and compute the setwise stabilizer $\mathrm{soc}(G)_{\Delta_1}$. Since the index of this group in $\mathrm{soc}(G)$ is at most 2^n, this can be done in time $\mathsf{poly}(n)2^n$ using more specialized algorithms for permutation groups [15] (in fact even the more general methods introduced earlier in this chapter suffice with worse constants in the exponent of the final runtime). This allows us to pass from $\mathrm{soc}(G)$ to the group $\mathrm{soc}(G)_{\Delta_1}$ at the cost of increasing the number of problems by a factor that is less than 2^n. We then decompose into the orbits of Ω under $(\mathrm{soc}(G))_{\Delta_1}$. Continuing this process recursively until all sets are singletons results in a total of at most $4^n \leq n^{O(d/\log d)}$ problems. This yields the following result.

Theorem 8.4.6 (Babai, Kantor and Luks [16])**.** *Isomorphism of colored graphs of degree at most d can be tested in $n^{O(d/\log d)}$ time.*

With some additional work, these techniques can also be applied to canonization using the methods of [18] which we describe in the next section.

8.4.5 Canonization of graphs of bounded degree

We now discuss how the algorithms described above can be extended to perform *graph canonization*: given a graph X, compute a unique representative $\mathrm{Can}(X)$ of its isomorphism class. Canonization is at least as hard as graph isomorphism since given two graphs X and Y, $X \cong Y$ if and only if $\mathrm{Can}(X) = \mathrm{Can}(Y)$.

The main idea behind the algorithm for performing canonization of bounded-degree graphs [18] is to replace the color automorphism problem with the *string placement problem.* Consider the strings $x, y \in \Sigma^{\Omega}$. We say these strings are isomorphic if there exists $\pi \in \mathrm{Sym}(\Omega)$ such that $\pi x = y$. If $G \leq \mathrm{Sym}(\Omega)$, then the strings x and y are G-isomorphic (denoted $x \cong_G y$) if there exists $g \in G$ such that $gx = y$. We say that a function $\mathrm{Can}(G) : \Sigma^{\Omega} \to \Sigma^{\Omega}$ is a canonical form with respect to G if for all $x, y \in \Sigma^{\Omega}$, $\mathrm{Can}(x, G) \cong_G x$ and $x \cong_G y$ if and only if $\mathrm{Can}(x, G) = \mathrm{Can}(y, G)$. In the case where $G = \mathrm{Sym}(\Omega)$, we omit G and write $\mathrm{Can}(x)$. Suppose that $\mathrm{Can}(G) : \Sigma^{\Omega} \to \Sigma^{\Omega}$ is a canonical form of x with respect to G. The notion of canonical form can be extended to cosets by defining $\mathrm{Can}(x, \sigma G) = \mathrm{Can}(\sigma x, G)$. The *canonical placement coset* with respect to G is defined to be $\mathrm{CP}(x, \sigma G) = \{g \in G \mid gx = \mathrm{Can}(x, \sigma G)\}$. It is easy to see that the following properties hold

$$\mathrm{CP}(x, \sigma G) = \sigma \mathrm{CP}(\sigma x, G) \tag{8.1}$$

$$\mathrm{CP}(x, \sigma G) = \tau \mathrm{Aut}_G(\tau x) \text{ for } \tau \in \mathrm{CP}(x, \sigma G) \tag{8.2}$$

The notation $\mathrm{Aut}_G(\tau x)$ denotes the group of G-automorphisms of τx. Babai and Luks [18] showed that these properties (together with the assumption that $\mathrm{CP}(x, \sigma G) \subseteq \sigma G$) characterize canonical placement functions. That is, any function that satisfies equations 8.1 and 8.2 defines the canonical placement coset for some canonical form with respect to σG. Then assuming CP is such a function, we obtain a canonical form by computing $\mathrm{CP}(x, \sigma G)$ and defining $\mathrm{Can}(x, \sigma G) = \mathrm{CP}(x, \sigma G)x$. Our goal is therefore to define an algorithm which is efficient and satisfies equations 8.1 and 8.2.

Such an algorithm can be defined using techniques similar to those used in the color automorphism problem. We neglect the optimizations described in Subsection 8.4.4 and adapt the basic $n^{O(d \log d)}$ algorithm to compute canonical placement cosets. We first make another generalization to the problem to allow recursion. If Δ is a G-stable subset of Ω then we define $\mathrm{CP}_\Delta(x, \sigma G)$ to be the canonical placement coset of the string x restricted to Δ (denoted $x\big|_\Delta$). As before, there are three cases. If $|\Delta| = 1$, then since Δ is G-stable, any $g \in G$ is an automorphism of $x\big|_\Delta$ so $\mathrm{CP}_\Delta(x, \sigma G) = \sigma G$. The second case occurs when the action of G on Δ is intransitive. Here, we again partition Δ into two nonempty G-stable subsets Δ_1 and Δ_2. We then set $\mathrm{CP}_\Delta(x, \sigma G) = \mathrm{CP}_{\Delta_1}(x, \mathrm{CP}_{\Delta_2}(x, \sigma G))$. We can think of this as first placing the substring $x\big|_{\Delta_2}$ into canonical form and then placing the substring $x\big|_{\Delta_1}$ into canonical form. The third case is when G is transitive on G. While the first two cases were essentially the same as in the algorithm for the color automorphism problem, performing string placement results in an important difference in the third case.

As before, we compute a minimal block system $\Delta_1, \ldots, \Delta_m$ for the action of G on Δ. However, now we must ensure that this minimal block system is constructed in a way which depends only on G and the natural ordering on Ω (think of Ω as $[n]$). This can be done by considering all pairs $\alpha, \beta \in \Omega$ and calculating for each pair the smallest block $\Delta_{\alpha,\beta}$ in which they are contained. We choose among the pairs that yield a nontrivial block, the pair α, β that comes first under the lexicographic ordering on all pairs. We then obtain a block system from $\Delta_{\alpha,\beta}$ by computing its images under G. Since there is also a lexicographic ordering on subsets of Ω, we can continue the process of selecting pairs recursively on this block system to obtain a minimal block system $\Delta_1, \ldots, \Delta_m$. We reiterate that this minimal block system depends only on G and the natural ordering on Ω.

Once we have computed the minimal block system, we proceed in the same manner as before with one additional trick. We compute the subgroup H that stabilizes each of the blocks and a complete set of representatives of $\tau_1, \ldots, \tau_m$ of G/H. We then calculate $\mathrm{CP}_\Delta(x, \sigma\tau_i H) = \rho_i H_i$ for each i. Next, we reindex so that $\rho_1 x\big|_\Delta = \cdots = \rho_s x\big|_\Delta < \rho_{s+1} x\big|_\Delta \leq \cdots \leq \rho_m x\big|_\Delta$ where $\leq$ is with respect to the lexicographic order. We then define

$\mathrm{CP}_\Delta(x, \sigma G) = \rho_1 \langle H_1, \{\rho_1^{-1}\rho_i\}_{2 \leq i \leq s}\rangle$. One can show [18] that this algorithm satisfies equations 8.1 and 8.2 from which it follows that it computes the canonical placement coset of some canonical form. The complexity analysis is the same as for the color automorphism problem.

In order to compute the canonical form of a graph X of degree at most d, we compute the graphs X_r and Y_r as before and build up the canonical placement coset gradually. At each step, we have the canonical placement coset of the subgraph Y_r and we use the string canonization algorithm to extend it to the canonical placement coset of Y_{r+1}. As before, the groups that arise are contained in Γ_{d-1} so that we obtain the same runtime. At this point the graph still depends on the edge e that we choose. However, this dependency can be eliminated by computing the canonical form with respect to each edge and then selecting the one which comes first lexicographically. This yields the following theorem.

Theorem 8.4.7 (Babai and Luks [18]). *Canonization of colored graphs of degree at most d can be performed in $n^{O(d \log d)}$ time.*

We remark that with more effort, the optimizations of Subsection 8.4.4 can also be applied to computing canonical forms; this gives an $n^{O(d/\log d)}$ algorithm.

8.5 The WL algorithm

Before discussing Zemlyachenko's degree reduction lemma, it is necessary to introduce the WL algorithm. The algorithm is a technique for iteratively recoloring the nodes of a graph in an attempt to discover restrictions that any automorphisms of the graph must obey. The algorithm cannot distinguish all non-isomorphic graphs, but it is known to fail only on an exponentially small fraction [17]. One starts with a graph X and colors each node by its degree. At each iteration of the WL algorithm, the color of each node is replaced by the pair consisting of its own color and the multiset of the colors of its neighbors. In order to keep the amount of space needed to store each color manageable, after each iteration the color of each node is replaced by its index in the sequence of all colors assigned to nodes where the colors are ordered lexicographically. In this way, the number of colors is reduced to at most n. It is easy to see that the partition that corresponds to the coloring is a (possibly improper) refinement of the color partition before the iteration was performed. The WL algorithm terminates once an iteration fails to produce a proper refinement. It is straightforward to see that the WL algorithm runs in $O(n^3)$ time since it can properly refine a partition at most $n-1$ times. Note that when the WL algorithm terminates, the induced subgraph $X(C_i)$ on any color class C_i is regular; moreover, the degree of a node in the induced bipartite subgraph $X(C_i, C_j)$ consisting of the edges between two different color classes C_i and C_j depends only on the color of the node. (Such graphs are called semiregular.) The WL algorithm can also be applied to a graph with an arbitrary initial coloring rather than the one where each node is initially colored by its degree. This will be useful in the algorithm for general graphs.

8.6 Zelmyachenko's degree reduction lemma and general graph isomorphism

The degree-reduction lemma uses the WL algorithm together with *individualization*. Given a graph, we can individualize a given vertex by erasing its current color and replacing it with the first color in $[n]$ that is not used for any other node. We define the degree of vertex x in color k to be the number of neighbors of x that have color k. The co-degree of x in color k is the number of vertexes with color k which are not neighbors of x. The color-degree of a vertex is the maximum over each color k of the minimum of the degree and co-degree in color k.

Lemma 8.6.1 (Zemlyachenko, cf. [7]). *Let X be a graph. Given a sequence of nodes $x_1, \ldots, x_m$ in X, we run the following procedure. At iteration i, we individualize x_i and run WL. Then for any d, there exists a sequence $x_1, \ldots, x_{4n/d}$ of nodes in X such that the graph Y resulting from the procedure described has color-degree at most d.*

All known algorithms [7, 76, 18, 16] for bounded degree graph isomorphism algorithms are based on the color automorphism problem and therefore apply more generally to graphs where the degree of each node is bounded by d in every color. Essentially, the algorithm is the same except that one must treat the neighborhood of a node in each color separately. This results in a more general algorithm with the same runtime. To obtain an even more general algorithm for graphs of color-degree at most d, we first run the WL algorithm. This ensures that each $X(C_i)$ is regular and each $X(C_i, C_j)$ is semiregular. If the degree of any $X(C_i)$ is greater than the degree of its complement, then we replace $X(C_i)$ in X by its complement. Similarly, if the degree of a node in some $X(C_i, C_j)$ is greater than its degree in the complement of $X(C_i, C_j)$, then we replace $X(C_i, C_j)$ by its complement. This process can change the isomorphism class of X. However, when testing if $X \cong Y$ we can keep track of the subgraphs that are complemented in X and Y and ensure that they correspond to the same colors. A similar trick applies to computing canonical forms of graphs of color-degree d.

Combining Zemlyachenko's lemma with the $n^{O(d/\log d)}$ algorithm for testing isomorphism of graphs of degree at most d, we get an $n^{4n/d+O(d/\log d)}$ algorithm for general graphs where d is a parameter that we can choose. By setting $4n/d = \Theta(d/\log d)$, one finds that the optimal choice is $d = \Theta(\sqrt{n \log n})$ which results in the following theorem.

Theorem 8.6.2 (Babai, Kantor and Luks [16]). *Graph isomorphism can be decided in* $2^{O(\sqrt{n \log n})}$ *time.*

We remark that in the case of strongly regular graphs, there is a faster $2^{O(\sqrt[3]{n} \log^2 n)}$ algorithm [119]. It is worth noting that graph canonization can also be performed in the same time bound by choosing the sequence of vertexes which results in the lexicographically least adjacency matrix. Essentially the same algorithms also work for colored and directed graphs.

8.7 Conclusion and open problems

While the results on bounded-degree graph isomorphism [7, 76, 18, 16] are impressive, no improvements for general graphs have been made in the nearly 30 years since those papers were published. A related problem posed in [18] is the *hypergraph isomorphism problem* where one must decide if two hypergraphs with n nodes are isomorphic. For a long time, it was open even to find a singly-exponential algorithm for this problem until such an algorithm was found by Luks [75]. Babai and Codenotti [11] later obtained the stronger result that isomorphism of hypergraphs of rank k (where each hyperedge contains at most k elements) can be decided in $2^{\tilde{O}(k^2\sqrt{n})}$ time. Combinatorially, it is easy to see that there exists a map from the set of all graphs on n nodes into the set of all rank 4 hypergraphs with $O(\sqrt{n})$ nodes such that two graphs are isomorphic if and only if their images under the map are isomorphic. If such a map could be computed efficiently, it would yield an $2^{\tilde{O}(\sqrt[4]{n})}$ algorithm for general graphs. We consider the problem of whether such a map can be computed efficiently to be an interesting open question.

For the case of bounded-degree graphs, there are two bottlenecks that one encounters when attempting to obtain an $n^{o(d/\log d)}$ algorithm. These are the bound on the index of the Sylow p-subgroup P of a primitive group G and the bound on the number of subproblems one obtains when the socle of the primitive group is a direct product of alternating groups (see Subsection 8.4.4). The first of these obstacles has since been overcome by advances in permutation group theory [93] while the second remains intact. Since the algorithm in this case is quite naive (as it splits the blocks in half arbitrarily) and because the socle in this case has a great deal of structure, it seems that there should be a more efficient decomposition. However, it is not immediately clear how one would obtain such an algorithm. We note that this is an important question since an $n^{o(d/\log d)}$ algorithm for graphs of degree at most d would give a superpolynomial speedup over the best algorithm for general graph isomorphism.

Chapter 9
PREVIOUS ALGORITHMS FOR GROUP ISOMORPHISM

Before moving on to our own results in 10 – 12, we review previously known algorithms for group isomorphism in this chapter. While relatively general and unstructured classes of groups such as the p- and solvable groups resisted progress until this work, several results are known about more restricted classes of groups. The simplest and most general of these (which we call the *generator enumeration algorithm*) is capable of deciding isomorphism of general groups in $n^{\log_p + O(1)}$ time [44, 74, 84] where p is the smallest prime dividing the order of the group. We give a complete description of the generator-enumeration algorithm in Section 9.1. Another simple result is that isomorphism of Abelian groups can be tested in polynomial time [74, 113, 125, 63] as we will show in Section 9.2. This result can be proved in various ways, but all of them depend on the structure theorem for finitely generated Abelian groups. There are also several more recent results for various structured classes of non-Abelian groups. We briefly survey these in Section 9.3.

9.1 The generator-enumeration algorithm

One of the most basic algorithms are group isomorphism is the *generator-enumeration algorithm* [44, 74, 84]. The algorithm works by enumerating all possible images a generating set S for group G could have under an isomorphism to a second group H. The idea is to simply test if any of these partial mappings extends to a full isomorphism between the groups. As we shall see, an isomorphism can be defined by its restriction to a generating set; one of these partial mappings yields an isomorphism if and only if the groups are isomorphic. Since there are at most $n^{|S|}$ such partial mappings, this results in an $n^{|S|+O(1)}$ time algorithm for testing isomorphism of general groups. We will also show that every group has a generating set of size at most $\log_p n$ where p is the smallest prime dividing the order of the group so this is $n^{\log_p n+O(1)}$ time in the worst case. Throughout this thesis, we will use n to denote the order of the group G.

The first step in showing that this idea actually works is to prove that any isomorphism can be defined by its restriction to a generating set.

Proposition 9.1.1. *Let $\phi : G \to H$ be a group isomorphism and let $G = \langle S \rangle$. Then for any isomorphism $\phi' : G \to H$ such that $\phi(x) = \phi'(x)$ for all $x \in S$, $\phi = \phi'$.*

Proof. Suppose that $\phi(x) = \phi'(x)$ for all $x \in S$. We need to show that $\phi(x) = \phi'(x)$ for *all* $x \in G$. For any $x \in G$, we know that $x = x_1^{\epsilon_1} \cdots x_k^{\epsilon_k}$ where each $x_i \in S$ and $\epsilon_i \in \{-1, 1\}$. From this we see that

$$\begin{aligned}\phi(x) &= \phi(x_1)^{\epsilon_1} \cdots \phi(x_k)^{\epsilon_k} \\ &= \phi'(x_1)^{\epsilon_1} \cdots \phi'(x_k)^{\epsilon_k} \\ &= \phi'(x_1^{\epsilon_1} \cdots x_k^{\epsilon_k}) \\ &= \phi'(x)\end{aligned}$$

as claimed. □

Thus, it suffices to iterate over all mappings from S into H. Next, we need a way to efficiently test if such a map extends to an isomorphism.

Lemma 9.1.2. *Let $G = \langle S \rangle$ and H be groups of order n and let $f : S \to H$ be a function. Then we can test if f extends to an isomorphism $\phi : G \to H$ in $O(n^3)$ time.*

Proof. We use a trick from [6]. Consider the subgroup $K = \langle (x, f(x)) \mid x \in S \rangle$ of $G \times H$. It is straightforward to show that f extends to an isomorphism from G to H if and only if

(a) $|K| = |G| = |H| = n$

(b) $\rho_1[K] = G$ and $\rho_2[K] = H$ where ρ_i is the projection onto the ith coordinate

We will argue that both of these conditions can be verified in $O(n^3)$ time. We note that it suffices to compute the set of elements in K in polynomial time. To do this, we simply maintain a set A of elements that can be formed as products of the generators $(x, f(x))$ where $x \in S$. Initially, $A = \{(x, f(x)) \mid x \in S\}$. At each step, we update A by adding inverses of all elements in A and all products of elements of A. The resulting set is closed under the group operation, so it is a subgroup of $G \times H$ that contains $\{(x, f(x)) \mid x \in S\}$. If follows that $A = K$ and it is easy to see that this procedure runs in polynomial time. □

It is worth noting that we have taken no effort to optimize the runtime in the above proof. With greater care, it is not hard to show that the time complexity $O(n^3)$ can be reduced to $O(n \log n)$. Since the input size is $O(n^2 \log n)$, this improved algorithm takes sublinear time.

Together, Proposition 9.1.1 and Lemma 9.1.2 show that we can test if G and H are isomorphic in $n^{|S|+O(1)}$ time assuming that we are given a generating set S.

Corollary 9.1.3. *Let G and H be groups and let S be a generating set for G. Then we can test if $G \cong H$ in $n^{|S|+O(1)}$ time.*

Next, we'll prove that every group G of order n has a generating set of size at most $\log_p n$ where p is the smallest prime dividing the order of the group. Moreover, we will show that such a generating set can be found in polynomial time.

Lemma 9.1.4. *Let G be a group of order $n > 1$ where p is the smallest prime dividing the order of G. The we can compute a generating set for G of size at most $\log_p n$.*

Proof. The idea is to build a generating set one element at a time and show that the order of the subgroup generated grows by a factor of at least p at every step. We start by arbitrarily choosing an element $x_1 \in G$ such that $x_1 \neq 1$ and let $G_1 = \langle x_1 \rangle$. At the i^{th} step where $i > 1$, we select an element $x_i \notin G_{i-1}$ and define $G_i = \langle x_1, \ldots, x_i \rangle$. Let k be the smallest integer such that $G_k = G$.

We claim that $k \leq \log_p n$. To see this, note that each G_{i-1} is a proper subgroup of G_i so $[G_i : G_{i-1}] \geq p$ for $1 \leq i \leq k$. Therefore, $p^k \leq n$ so $k \leq \log_p n$. Moreover, it is clear that the above computation can be performed in polynomial time. □

Combining Corollary 9.1.3 with Lemma 9.1.4, shows that we can test if two groups are isomorphic in $n^{\log_p n+O(1)}$ time.

Corollary 9.1.5. *Let G and H be groups of order n. Then we can test if $G \cong H$ in $n^{\log_p n+O(1)}$ time where p is the smallest prime dividing the order of the group.*

9.2 Testing isomorphism of Abelian groups

In the previous section, we showed an extremely general algorithm that took quasi-polynomial time. In this section, we insist on polynomial time but only require the algorithm to work for Abelian groups. Various authors [74, 113, 125, 63] have shown algorithms that prove that Abelian group isomorphism is in polynomial time. We give our own proof which is simpler but results in a larger but still polynomial runtime. We make no effort to optimize the exponents in runtime and it is easy to improve them by taking greater care. Our algorithm relies on Theorem 3.4.2 which we reproduce here for convenience.

Theorem 3.4.2 (The structure of finitely generated Abelian groups (invariant factor version))**.** *Let G be a finitely generated Abelian group. Then there exist positive integers $d_1, \ldots, d_k$ and m such that $d_i \mid d_{i+1}$ for each i and*

$$G \cong \mathbb{Z}_{d_1} \times \cdots \times \mathbb{Z}_{d_k} \times \mathbb{Z}^m$$

Moreover, this decomposition is unique up to reordering the factors.

Note that for the finite groups that we consider, $m = 0$. Our strategy is as follows: we find an element x of order d_k and argue that there exists a subgroup[1] K of G such that $G = K \times \langle x \rangle$. This allows us to recover the structure constant d_k. Since $K \cong \mathbb{Z}_{d_1} \times \cdots \times \mathbb{Z}_{d_{k-1}}$, we obtain the rest of the d_i's recursively.

It is easy to see that we can find an element of order d_k in $O(n^2)$ time since this is just an element of maximal order in G. Therefore, we proceed by proving that $G = K \times \langle x \rangle$ for some $K < G$.

[1]Note that every subgroup of an Abelian group is normal so the direct product is well-defined.

Lemma 9.2.1. *Let G be an Abelian group and let x be an element of G of maximal order. Then there is a subgroup $K < G$ such that $G = K \times \langle x \rangle$. Moreover, we can compute such an x and K in $O(n^2 \log^2 n)$ time.*

Proof. To compute an element $x \in G$ of maximal order in polynomial time, we simply calculate the order of every element of G. To compute a subgroup K such that $G = K \times \langle x \rangle$, we start by finding an element $y_1 \in G \setminus \langle x \rangle$. If no such y_1 exists, then we can take $K = 1$. Otherwise, $\langle x \rangle$ and $K_1 = \langle y_1 \rangle$ are disjoint subgroups of G. At the i^{th} step for $i > 1$, we proceed by choosing $y_i \in G \setminus \langle x, y_1, \ldots, y_{i-1} \rangle$ and let $K_i = \langle y_1, \ldots, y_i \rangle$. Then K_i and $\langle x \rangle$ are disjoint. We proceed until $K_j \times \langle x \rangle = \langle x, y_1, \ldots, y_j \rangle$ coincides with G. We then stop and set $K = K_j$. □

An algorithm for testing isomorphism of Abelian groups can be obtained by recursive application of Lemma 9.2.1.

Theorem 9.2.2. *Let G and H be Abelian groups. Then we can test if $G \cong H$ in $O(n^2 \log^3 n)$ time.*

Proof. Let G be as in Theorem 3.4.2. By Theorem 3.4.2, it suffices to show how to recover the structure constants d_i (including their multiplicities). By applying Lemma 9.2.1, we obtain a decomposition $G = K \times \langle x \rangle$ where $|x| = d_k$ in polynomial time. Since $K \cong \mathbb{Z}_{d_1} \times \cdots \times \mathbb{Z}_{d_{k-1}}$ by Theorem 3.4.2, we can obtain the remaining structure constants $d_1, \ldots, d_{k-1}$ by continuing recursively with the subgroup K. □

It is worth noting that the above bound can be improved to $O(n \log n)$ time [63].

9.3 Other algorithms for group isomorphism

Faster algorithms have been obtained for various special cases beyond the Abelian groups. Le Gall [70] gave a polynomial-time algorithm for groups consisting of a semidirect product of an Abelian group with a cyclic group of coprime order. Le Gall's proof was based on a partial structure theorem for this class of groups. His result was extended to a class of groups with a normal Hall subgroup by Qiao, Sarma and Tang [94] by noting that it was related to a certain group action on a subgroup called the *socle* (which we discussed in Subsection 8.4.4). An *Abelian Sylow tower* for a group is a normal series whose quotients are isomorphic to a maximal p-subgroup of G for some prime p that divides the order of the group. Babai and Qiao [19] showed that testing isomorphism of groups with Abelian Sylow towers is in polynomial time. Babai, Codenotti, Grochow and Qiao [12] showed an $n^{O(\log \log n)}$ time algorithm for the class of groups with no normal Abelian subgroups; the runtime was later improved to polynomial by Babai, Codenotti and Qiao [34, 13]. The *solvable radical* of a group is its unique maximal normal solvable subgroup. The *central radical groups* are those where the solvable radical is contained in the center of the group. Grochow and Qiao [51] generalized the result of Babai, Codenotti, Grochow and Qiao [12] for groups with no Abelian normal subgroups by showing an $n^{O(\log \log n)}$ time algorithm for central radical groups. In another line of work, Lewis and Wilson [71] showed that isomorphism of quotients of the Heisenberg group can be decided in polynomial time.

Chapter 10

P-GROUP ISOMORPHISM

10.1 Introduction

The main result of this chapter is an algorithm that is faster than the generator-enumeration algorithm for p-groups, which are believed to be the hard case of group isomorphism [12, 34, 19]. Before this work, obtaining an $n^{(1-\epsilon)\log_p n+O(1)}$ algorithm where $\epsilon > 0$ was discussed as a longstanding open problem [72][1].

Theorem 10.1.1. *p-group isomorphism is decidable in $n^{\min\{(1/2)\log_p n+O(p\log p),\, \log_p n\}}$ time. In particular, $n^{(1/2)\log_p n+O(\log n/\log\log n)}$ and $n^{(1/2)\log n+O(1)}$ are upper bounds on its time complexity.*

Theorem 2.2.1 from Chapter 2 then follows as a corollary, as promised.

The first step in our algorithm reduces group isomorphism to many instances of composition-series isomorphism. (Two composition series are isomorphic if there exists an isomorphism that maps each subgroup in the first series to the corresponding subgroup in the second series.)

Theorem 10.1.2. *Testing isomorphism of two groups G and H is $n^{(1/2)\log_p n+O(1)}$ time Turing reducible to testing isomorphism of composition series for G and H where p is the smallest prime dividing the order of the group.*

This bound can be proved by counting the number of composition series using a simple argument. We are grateful to Laci Babai for pointing this out as it simplifies our algorithm.

[1]Subsequent to the initial version of [111], James Wilson (personal communication) showed an upper bound of $n^{c\log_p n+O(1)}$ where $c < 1/4$ for the p-group isomorphism algorithm from [90]. However, the analysis has not been published.

Our second step is to reduce p-group composition-series isomorphism to testing isomorphism of graphs of degree $p+O(1)$. We accomplish this by constructing a tree with a node of each coset for each of the intermediate subgroups in the composition series. By adding certain gadgets that encode the multiplication table of the group, we show that composition series for two p-groups are isomorphic if and only if the graphs resulting from this construction are isomorphic. By applying the $n^{O(d \log d)}$ time algorithm stated in Theorem 8.4.5 [76, 18] for testing isomorphism of graphs of degree at most d, we obtain an $n^{O(p)}$ time algorithm for p-group composition-series isomorphism. Combining this result with our reduction from group isomorphism to composition-series isomorphism yields an $n^{(1/2)\log_p n+O(p)}$ time algorithm for testing isomorphism of p-groups. Combining this algorithm with the generator-enumeration algorithm completes the proof of Theorem 10.1.1.

Recall that the canonical form of a class of objects is a function that maps each object to a unique representative of its isomorphism class. Since canonical forms of graphs of degree at most d can be computed in $n^{O(d \log d)}$ time by Theorem 8.4.7 [76, 18], Theorem 10.1.1 can be modified to perform *p-group canonization* in the same complexity bound. (It is worth noting that Luks showed that there is a faster $n^{O(d/\log d)}$ algorithm for testing isomorphism of graphs of degree at most d by Theorem 8.4.6 [16], but it does not improve our results.) If $p \leq \alpha$ is small, we compute the canonical form of the graph that arises from each choice of composition series and choose the one that comes first lexicographically. A canonical multiplication table for the p-group is then recovered from this canonical form. When $p > \alpha$, we use a variant of the generator-enumeration algorithm that performs group canonization.

For the necessary background on group theory, see Chapter 3. In Section 10.2, we start by reducing group isomorphism to composition-series isomorphism. In Section 10.3, we present the reduction from p-group composition-series isomorphism to low-degree graph isomorphism. In Section 10.4, we derive our algorithms for p-group isomorphism.

10.2 Reducing group isomorphism to composition-series isomorphism

In this section, we prove an upper bound on the number of composition series for a group and provide a simple method for enumerating all such composition series. Originally [108], we used a more complex construction to enumerate all composition series within a particular class and an upper bound was proved on the size of this class of composition series. However, Laci Babai pointed out that the upper bound actually holds for the class of all composition series. This allows us to employ a much simpler argument.

Lemma 10.2.1. *Let G be a group. Then the number of composition series for G is at most $n^{(1/2)\log_p n+O(1)}$ where p is the smallest prime dividing the order of G. Moreover, one can enumerate all composition series for G in $n^{(1/2)\log_p n+O(1)}$ time.*

Proof. We show that one can enumerate a class of chains that contains all maximal chains of subgroups in $n^{\log_p n+O(1)}$ time. Since every maximal chain of subgroups contains at most one composition series as a subchain, this suffices to prove the result.

We start by choosing the first nontrivial subgroup in the series. Each of these is generated by a single element so there are at most n choices. If we have a chain $G_0 = 1 < \cdots < G_k$ of subgroups of G, then the next subgroup in the chain can be chosen in at most $|G/G_k|$ ways since different representatives of the same coset generate the same subgroup. Since each $|G_{i+1}| \geq p\,|G_i|$, we see that the number of choices $|G/G_k|$ for G_{k+1} is at most n/p^k. Therefore, the total number of choices required to construct a chain of subgroups in this manner is at most

$$\begin{aligned}\prod_{k=0}^{\lfloor \log_p n\rfloor -1} (n/p^k) &\leq p^{\sum_{k=0}^{\lceil \log_p n\rceil} k} \\ &= p^{(1/2)\log_p^2 n+O(\log_p n)} \\ &\leq n^{(1/2)\log_p n+O(1)}\end{aligned}$$

Since the set of subgroup chains enumerated by this process includes all maximal chains of subgroups, the result follows. □

We say that two composition series $G_0 = 1 \lhd \cdots \lhd G_m = G$ and $H_0 = 1 \lhd \cdots \lhd H_{m'} = H$ are isomorphic if there exists an isomorphism $\phi : G \to H$ such that each $\phi[G_i] = H_i$. (Note that if these composition series are isomorphic, then $m = m'$.) It is now very easy to obtain the Turing reduction from group isomorphism to composition series isomorphism.

Theorem 10.1.2. *Testing isomorphism of two groups G and H is $n^{(1/2)\log_p n + O(1)}$ time Turing reducible to testing isomorphism of composition series for G and H where p is the smallest prime dividing the order of the group.*

Proof. Let G and H be groups. Fix a composition series S for G. If $G \cong H$, then some composition series S' for H will be isomorphic to S. Thus, testing isomorphism of G and H reduces to testing if S is isomorphic to some composition series for S'. The result is then immediate from Lemma 10.2.1. □

The reduction also applies to reducing group canonization to composition series canonization. For the convenience of the reader, we explicitly define canonical forms for groups and composition series.

Definition 10.2.2. *A map* $\text{Can}_{\mathbf{Grp}}$ *is a* canonical form for groups *if for each group G,* $\text{Can}_{\mathbf{Grp}}(G)$ *is an $n \times n$ multiplication table with elements in $[n]$ that is isomorphic to G, such that, if G and H are groups, $G \cong H$ if and only if* $\text{Can}_{\mathbf{Grp}}(G) = \text{Can}_{\mathbf{Grp}}(H)$.

Definition 10.2.3. *A map* $\text{Can}_{\mathbf{Comp}}$ *is a* canonical form for composition series *if for each composition series S for a group G with subgroup chain $G_0 = 1 < \cdots < G_m = G$,* $\text{Can}_{\mathbf{Comp}}(S) = (M, \psi[G_0], \ldots, \psi[G_m])$ *such that the following hold.*

(a) M is an $n \times n$ matrix with entries in $[n]$.

(b) M is the multiplication table for a group that is isomorphic to G under $\psi : G \to [n]$.

(c) If S and S' are composition series then $S \cong S'$ if and only if $\text{Can}_{\mathbf{Comp}}(S) = \text{Can}_{\mathbf{Comp}}(S')$.

Theorem 10.2.4. *Computing the canonical form of a group is $n^{(1/2)\log_p n+O(1)}$ time Turing reducible to computing canonical forms of composition series for the group where p is the smallest prime dividing the order of the group.*

Proof. Let G be a group. We use Lemma 10.2.1 to enumerate all of the at most $n^{(1/2)\log_p n+O(1)}$ composition series S for G and compute the canonical form of each one. From each such canonical form, we extract the multiplication table and define $\mathrm{Can}_{\mathbf{Grp}}(G)$ to be the lexicographically least matrix among all such multiplication tables. Since two groups are isomorphic if and only if the sets of isomorphism classes of their composition series coincide, it follows that $\mathrm{Can}_{\mathbf{Grp}}$ is a canonical form. $\square$

10.3 Composition-series isomorphism and canonization

In this section, we reduce composition-series isomorphism to low-degree graph isomorphism. We also extend the reduction to perform composition-series canonization instead of isomorphism testing. We shall make use of the following result of Babai and Luks [76, 18] that we discussed in Chapter 8.

Theorem 8.4.7 (Babai and Luks [18])**.** *Canonization of colored graphs of degree at most d can be performed in $n^{O(d\log d)}$ time.*

10.3.1 Isomorphism testing

To test if two composition series are isomorphic, we construct a tree by starting with the whole group G and decomposing it into its cosets G/G_{m-1}; we then further decompose each coset in G/G_{m-1} into the cosets G/G_{m-2} that it contains. This process is repeated until we reach the trivial group $G_0 = 1$. We make this precise with the following definition.

Definition 10.3.1. *Let G be a group and consider the composition series S given by the subgroups $G_0 = 1 \lhd \cdots \lhd G_m = G$. Then $T(S)$ is defined to be the rooted tree whose nodes are $\bigcup_i G/G_i$. The root node is G. The leaf nodes are $\{x\} \in G/1$ which we identify with $x \in G$. For each node $xG_{i+1} \in G/G_{i+1}$, there is an edge to each yG_i such that $yG_i \subseteq xG_{i+1}$.*

We now use this tree to define a graph that encodes the multiplication table of G. The idea is to attach a multiplication gadget to the nodes $x, y, z \in G$ for each entry $xy = z$ in the multiplication table. If we did this naively, each node $x \in G$ would have degree $\Omega(n)$. We address this problem by defining a variant of the rooted product [46] which we call a leaf product. Let T_1 and T_2 be rooted trees. The *leaf product* of T_1 and T_2 (denoted $T_1 \odot T_2$) is the tree obtained by creating a copy of T_2 for each leaf node of T_1 and identifying the root of each copy with one of the leaf nodes. We denote by $L(T)$ the set of leaves of the tree T.

Definition 10.3.2. *Let T_1 and T_2 be trees rooted at r_1 and r_2. Then the* leaf product $T_1 \odot T_2$ *is the tree rooted at r_1 with vertex set*

$$V(T_1) \cup \{(x, y) \mid x \in L(T_1) \text{ and } y \in V(T_2) \setminus \{r_2\}\}$$

The set of edges is

$$\begin{aligned} E(T_1) &\cup \{(x, (x, y)) \mid x \in L(T_1) \text{ and } (r_2, y) \in E(T_2)\} \\ &\cup \{((x, y), (x, z)) \mid x \in L(T_1) \text{ and } (y, z) \in E(T_2) \text{ where } y, z \neq r_2\} \end{aligned}$$

Leaf products are non-commutative but are associative if we identify the tuples $(x, (y, z))$, $((x, y), z)$ with (x, y, z) in the vertex set. (This is the same sense in which cross products are associative.) We shall make this identification from now on as it simplifies our notation.

Since we will need to consider isomorphisms of leaf products of trees, it is also useful to define leaf products of tree isomorphisms.

Definition 10.3.3. *For each $1 \leq i \leq k$, let T_i and T_i' be trees rooted at r_i and r_i' and let $\phi_i : T_i \to T_i'$ be an isomorphism. Then the* leaf product $\bigodot_{i=1}^k \phi_i : \bigodot_{i=1}^k T_i \to \bigodot_{i=1}^k T_i'$ *sends each $(x_1, \ldots, x_j)$ to $(\phi_1(x_1), \ldots, \phi_j(x_j))$ where each $x_i \in L(T_i)$ for $i < j$, $x_j \in V(T_j) \setminus \{r_j\}$ and $j \leq k$.*

For a bijection ϕ between the leaves of two trees, we shall use the notation $\hat{\phi}$ to denote the unique isomorphism between the trees to which ϕ extends (when such an isomorphism exists). The following extension of leaf products is convenient. For each $1 \leq i \leq k$, let ϕ_i be a bijection from the leaves of T_i to the leaves of T_i' that extends uniquely to an isomorphism $\hat{\phi}_i : T_i \to T_i'$. Then we define $\bigodot_{i=1}^{k} \phi_i = \bigodot_{i=1}^{k} \hat{\phi}_i$.

It is easy to see that $\bigodot_{i=1}^{k} \phi_i$ is an isomorphism from $\bigodot_{i=1}^{k} T_i$ to $\bigodot_{i=1}^{k} T_i'$.

Proposition 10.3.4. *For each $1 \leq i \leq k$, let T_i and T_i' be rooted trees and let ϕ_i be a bijection between the leaves of T_i and T_i' such that ϕ_i extends uniquely to an isomorphism from T_i to T_i'. Then $\bigodot_{i=1}^{k} \phi_i : \bigodot_{i=1}^{k} T_i \to \bigodot_{i=1}^{k} T_i'$ is a well-defined isomorphism.*

As we mentioned earlier, simply attaching multiplication gadgets to the leaves of the tree $T(S)$ would result in a tree of large degree. We resolve this problem by considering the tree $T(S) \odot T(S)$ instead. We show how to construct multiplication gadgets so that each of the n^2 leaf nodes is involved in only a constant number of edges. This causes the resulting graph to have degree $p + O(1)$ when G is a p-group. The details of this construction are described in the following definition.

Definition 10.3.5. *Let G be a group and let S be a composition series. Let M be the tree with a root connected to three nodes $\leftarrow$, $\rightarrow$ and $=$ with colors "left", "right" and "equals" respectively. To construct $X(S)$, we start with the tree $T(S) \odot T(S) \odot M$ and connect multiplication gadgets to the leaf nodes. For each $x, y \in G$, we create the path $((x, y, \leftarrow), (y, x, \rightarrow), (xy, y, =))$. The nodes other than the leaf nodes in $X(S)$ are colored "internal."*

The graph $X(S)$ is a *cone graph*; that is, a rooted tree with additional edges between nodes at the same level. We call the edges that form the tree in a cone graph *tree edges* and the edges between nodes at the same level *cross edges*.

Our next goal is to show that two composition series S and S' are isomorphic if and only if $X(S)$ and $X(S')$ are isomorphic. Let **Comp** be the class of composition series for finite groups and let **CompTree** be the class of graphs that are isomorphic to a graph $X(S)$ for some composition series S. For each pair of composition series S and S' and each isomorphism $\phi : S \to S'$, we overload the symbol X from Definition 10.3.5 by defining $X(\phi) : X(S) \to X(S')$ to be $\phi \odot \phi \odot \mathrm{id}_M$.

We seek to show that for two composition series S and S', the map $X_{S,S'} : \mathrm{Iso}(S, S') \to \mathrm{Iso}(X(S), X(S'))$ given by $\phi \mapsto X(\phi)$ is surjective and can be evaluated in polynomial time. We note that in particular, this result shows that X can be used to reduce composition series isomorphism to testing isomorphism of the resulting graphs. We start by showing that any isomorphism between S and S' maps to an isomorphism between $X(S)$ and $X(S')$.

Lemma 10.3.6. *For each pair of composition series S and S', $X_{S,S'} : \mathrm{Iso}(S, S') \to \mathrm{Iso}(X(S), X(S'))$ is well-defined.*

Proof. Let $G_0 = 1 \lhd \cdots \lhd G_m = G$ and $H_0 = 1 \lhd \cdots \lhd H_m = H$ be the subgroup chains for the composition series S and S' and let $\phi : S \to S'$ be an isomorphism. We can view ϕ as a bijection from the leaves of $T(S)$ to the leaves of $T(S')$. Since each $\phi[G_i] = H_i$, we see that ϕ extends to a unique isomorphism $\hat{\phi} : T(S) \to T(S')$. By Proposition 10.3.4, $X(\phi) : T(S) \odot T(S) \odot M \to T(S') \odot T(S') \odot M$ is a tree isomorphism. Then by Definition 10.3.5, we just need to show that $X(\phi)$ respects the cross edges representing the multiplication gadgets.

Let $x, y \in G$. Then $X(S)$ contains the path $((x, y, \leftarrow), (y, x, \rightarrow), (xy, y, =))$. In H, $X(S')$ contains the path $((\phi(x), \phi(y), \leftarrow), (\phi(y), \phi(x), \rightarrow), (\phi(xy), \phi(y), =))$ since $\phi(x)\phi(y) = \phi(xy)$. By definition, we see that $X(\phi)$ maps the path $((x, y, \leftarrow), (y, x, \rightarrow), (xy, y, =))$ in $X(S)$ to the path $((\phi(x), \phi(y), \leftarrow), (\phi(y), \phi(x), \rightarrow), (\phi(xy), \phi(y), =))$ in $X(S')$. Since $X(S)$ and $X(S')$ have equal numbers of cross edges, it follows that $X(\phi) : X(S) \to X(S')$ is an isomorphism. □

Next, we show that each $X_{S,S'}$ is surjective. This is more difficult and is accomplished by the next two results. We first show that every isomorphism from $X(S)$ to $X(S')$ can be expressed as a leaf product.

Lemma 10.3.7. *Let S and S' be composition series for the groups G and H and let $\theta : X(S) \to X(S')$ be an isomorphism. Define $\phi : G \to H$ to be $\theta\big|_G$. Then*

(a) $\theta = \phi \odot \phi \odot \mathrm{id}_M$ and

(b) $\phi : S \to S'$ is an isomorphism.

Proof. First, we prove part (a). It is clear that ϕ is a bijection between G and H that extends uniquely to an isomorphism from $T(S)$ to $T(S')$. Let $x, y \in G$. We will say a path from x to y is *left-right* if it starts at x, moves to a node colored "left" along tree edges (away from the root), follows a cross edge to a node colored "right" and then moves to y along tree edges (towards the root). Since the only cross edge in $X(S)$ colored ("left", "right") between the subtrees of $T(S) \odot T(S) \odot M$ rooted at x and y is $((x, y, \leftarrow), (y, x, \rightarrow))$, there is exactly one left-right path from x to y. We denote this path by $P(x, y)$.

Since θ maps the root of $X(S)$ to the root of $X(S')$, θ maps left-right paths to left-right paths. Therefore, θ sends $P(x, y)$ to $P(\phi(x), \phi(y))$ so the node $(x, y, \leftarrow)$ in $X(S)$ is mapped to the node $(\phi(x), \phi(y), \leftarrow)$ in $X(S')$.

For part (b), we let $x, y, z \in G$ such that $xy = z$. This multiplication rule is represented in $X(S)$ by the path $((x, y, \leftarrow), (y, x, \rightarrow), (z, y, =))$. By part (a), we know that θ maps this path to $((\phi(x), \phi(y), \leftarrow), (\phi(y), \phi(x), \rightarrow), (\phi(z), \phi(y), =))$. This implies that $\phi(x)\phi(y) = \phi(z)$ in H so that ϕ is an isomorphism from G to H.

Let $G_0 = 1 \lhd \cdots \lhd G_m = G$ and $H_0 = 1 \lhd \cdots \lhd H_m = H$ be the chains of subgroups in the composition series S and S'. It remains to show that each $\phi[G_i] = H_i$. Since ϕ is an isomorphism from G to H, it follows that $\phi(1) = 1$. This implies that θ maps each node G_i in $X(S)$ to the node H_i in $X(S')$. Then because the elements of G_i correspond precisely to those nodes $x \in G$ such that x is a descendant of the node G_i in $T(S) \odot T(S) \odot M$, it follows that $\phi[G_i] = H_i$. Thus, ϕ is an isomorphism from S to S'. □

Theorem 10.3.8. *For each pair of composition series S and S', $X_{S,S'}$ is a bijection. Moreover, both $X(S)$ and $X(\phi)$ where $\phi \in \mathrm{Iso}(S, S')$ can be computed in polynomial time.*

Proof. Combining Lemmas 10.3.6 and 10.3.7 shows that each $X_{S,S'}$ is surjective. To see that it is injective, we note that if $\phi, \psi \in \mathrm{Iso}(S, S')$ and $X(\phi) = X(\psi)$ then $\phi \odot \phi \odot \mathrm{id}_M = \psi \odot \psi \odot \mathrm{id}_M$ so $\phi = \psi$. Since X is defined in terms of leaf products and leaf products can be evaluated in polynomial time, X can also be evaluated in polynomial time. □

The correctness of our reduction follows.

Corollary 10.3.9. *Let S and S' be composition series. Then $S \cong S'$ if and only if $X(S) \cong X(S')$.*

In order to obtain an efficient algorithm for p-group composition-series isomorphism, we must show that the degree of the graph is not too large.

Lemma 10.3.10. *Let G be a group with a composition series S such that α is an upper bound for the order of any factor. Then the graph $X(S)$ has degree at most* $\max\{\alpha+1, 4\}$ *and size $O(n^2)$.*

Proof. The tree $T(S)$ has size $O(n)$ and degree $\alpha + 1$ while the tree M has size 4 and degree 3. Therefore $T(S) \odot T(S) \odot M$ (and hence $X(S)$) has size $O(n^2)$ and degree $\max\{\alpha+1, 4\}$. Adding the edges for the multiplication gadgets in $X(S)$ increases the degrees of the leaves of $T(S) \odot T(S) \odot M$ to at most 3, so $X(S)$ also has degree $\max\{\alpha + 1, 4\}$. □

We are now in a position to obtain an algorithm for composition-series isomorphism.

Theorem 10.3.11. *Let S and S' be composition series such that α is an upper bound for the order of any factor. Then we can test if $S \cong S'$ in $n^{O(\alpha \log \alpha)}$ time.*

Proof. We can compute the graphs $X(S)$ and $X(S')$ in polynomial time. By Corollary 10.3.9, $S \cong S'$ if and only if $X(S) \cong X(S')$. By Lemma 10.3.10, the number of nodes in $X(S)$ is $O(n^2)$ and the degree is at most $\max\{\alpha+1, 4\} = O(\alpha)$. Then we can test if $X(S) \cong X(S')$ in $n^{O(\alpha \log \alpha)}$ time using the bounded-degree graph isomorphism algorithm from Theorem 8.4.7. □

10.3.2 Canonization

We also show how to compute canonical forms of composition series. This result is also useful for further improving the efficiency of the algorithm for p-group isomorphism (see Chapter 12). Our high-level strategy for constructing a canonical form for a composition series S is to compute the canonical form of the graph $X(S)$. We then reconstruct a composition series $Y(\mathrm{Can}_{\mathbf{Graph}}(X(S)))$ isomorphic to S by inspecting the structure of $\mathrm{Can}_{\mathbf{Graph}}(X(S))$.

Definition 10.3.12. *For each composition series S for a group G and a graph $A \cong X(S)$, we fix an arbitrary isomorphism $\pi : X(S) \to A$. We define $Y(A)$ to be the composition series $\pi[1] \lhd \cdots \lhd \pi[G]$ for the group with elements $\pi[G]$, where we define $\pi(x)\pi(y) = \pi(z)$ if there exists a path $(a_{\pi(x)}, a_{\pi(y)}, a_{\pi(z)})$ colored ("left", "right", "equals"), such that $a_{\pi(x)}$, $a_{\pi(y)}$ and $a_{\pi(z)}$ are descendants of x, y and z in the image of the tree $T(S) \odot T(S) \odot M$ under π.*

For each pair of composition series S and S' for groups G and H, graphs $A \cong X(S)$ and $A' \cong X(S')$, let $\pi : X(S) \to A$ and $\pi' : X(S') \to A'$ be the fixed isomorphisms chosen above. Then for each isomorphism $\theta : A \to A'$, we define $Y(\theta) : \pi[G] \to \pi'[H]$ to be $\theta\big|_{\pi[G]}$.

First, we need to show that each $Y(A)$ is well-defined.

Lemma 10.3.13. *Let S be a composition series, let A be a graph and let $\pi : X(S) \to A$ be an isomorphism. Then $Y(A)$ is a well-defined composition series that can be computed in polynomial time and $Y(\pi)$ is an isomorphism from S to $Y(A)$.*

Proof. Let $G_0 = 1 \lhd \cdots \lhd G_m = G$ be the subgroup chain for S. We note that the height of $T(S) \odot T(S) \odot M$ is $2m + 1$ where m is the composition length of S. Now, G is the group consisting of the elements at a distance of m from the root so $\pi[G]$ is independent of which isomorphism $\pi : X(S) \to A$ we consider. Moreover, we can compute $\pi[G]$ in polynomial time. For each $x, y, z \in G$, $xy = z$ if and only if there exists a path $((x, y, \leftarrow), (y, x, \to), (z, y, =))$ in $X(S)$. Equivalently, $xy = z$ if and only if there exists a path (a_x, a_y, a_z) colored ("left", "right", "equals") where a_x, a_y and a_z are descendants of x, y and z in $T(S) \odot T(S) \odot M$.

Consider the set of elements $\pi[G]$. For each $\pi(x), \pi(y), \pi(z) \in \pi[G]$, define $\pi(x)\pi(y) = \pi(z)$ if and only if there exists a path $(a_{\pi(x)}, a_{\pi(y)}, a_{\pi(z)})$ colored ("left", "right", "equals") where $a_{\pi(x)}$, $a_{\pi(y)}$ and $a_{\pi(z)}$ are descendants of $\pi(x)$, $\pi(y)$ and $\pi(z)$ in the image of $T(S) \odot T(S) \odot M$ under π. Then $\pi[G]$ is a group that we can compute in polynomial time and $Y(\pi)$ is a group isomorphism from G to $\pi[G]$.

Now, for each G_i, $\pi[G_i]$ consists of the nodes in $\pi[G]$ that are descendants of the node $\pi(G_i)$. Each node $\pi(G_i)$ is the node on the path from the root of A to $\pi(1)$ at distance $m - i$

from the root. The node $\pi(1)$ is the identity of the group $\pi[G]$ and can therefore be found by inspecting the multiplication rules of $\pi[G]$. Thus, we can compute each set of nodes $\pi[G_i]$ in polynomial time independently of π. This yields a composition series $\pi[1] \triangleleft \cdots \triangleleft \pi[G]$ that does not depend on the choice of π. From Definition 10.3.12, we see that this composition series is in fact $Y(A)$. Moreover, $Y(\pi)$ is an isomorphism from S to $Y(A)$. $\square$

As for X, we define $Y_{A,A'} : \mathrm{Iso}(A, A') \to \mathrm{Iso}(Y(A), Y(A'))$ by $\theta \mapsto Y(\theta)$ for each pair of graphs $A, A' \in \mathbf{CompTree}$. In order to compute canonical forms, we shall need to show that each $Y_{A,A'}$ is surjective and can be evaluated in polynomial time.

Theorem 10.3.14. *For each pair of graphs $A, A' \in \mathbf{CompTree}$, $Y_{A,A'}$ is a bijection. Moreover, both $Y(A)$ and $Y(\theta)$ where $\theta \in \mathrm{Iso}(A, A')$ can be computed in polynomial time.*

Proof. Let S and S' be composition series with chains of subgroups $G_0 = 1 \triangleleft \cdots \triangleleft G_m = G$ and $H_0 = 1 \triangleleft \cdots \triangleleft H_m = H$, let $A \cong X(S)$ and $A' \cong X(S')$ be graphs, and let $\pi : X(S) \to A$, $\pi' : X(S) \to A'$ and $\theta : A \to A'$ be isomorphisms.

First, we observe that Y respects composition. Since $\psi = \theta\pi$ is an isomorphism from $X(S)$ to A', Lemma 10.3.13 implies that $Y(\psi) = Y(\theta)Y(\pi)$ is an isomorphism from S to $Y(A')$ and that $Y(\pi)$ is an isomorphism from S to $Y(A)$. It follows that $Y(\theta) = Y(\psi)(Y(\pi))^{-1}$ is an isomorphism from $Y(A)$ to $Y(A')$. Thus, $Y_{A,A'}$ is a well-defined function.

To show that $Y_{A,A'}$ is bijective, we first note that $YX = I_{\mathbf{Comp}}$, which implies that $Y_{X(S),X(S')}$ is surjective. Lemma 10.3.7 implies that $Y_{X(S),X(S')}$ is also injective. Now, for each $\theta : A \to A'$, we have $\theta = \pi' \rho \pi^{-1}$ for some isomorphism $\rho : X(S) \to X(S')$. Therefore, $Y(\theta) = Y(\pi')Y(\rho)Y(\pi^{-1})$. Since $Y_{X(S),X(S')}$ is a bijection, we see that $Y_{A,A'}$ is also a bijection. The fact that Y can be evaluated in polynomial time follows from Definition 10.3.12 and Lemma 10.3.13. $\square$

To devise an algorithm for composition series canonization, we utilize X and Y together with the canonical form for graphs of bounded degree Theorem 8.4.7 (which we denote by $\mathrm{Can}_{\mathbf{Graph}}$).

Theorem 10.3.15. *The map $Y \circ \mathrm{Can}_{\mathbf{Graph}} \circ X$ is a canonical form for composition series. If S is a composition series such that α is an upper bound for the order of any factor, then we can compute $\mathrm{Can}_{\mathbf{Comp}}(S) = (Y \circ \mathrm{Can}_{\mathbf{Graph}} \circ X)(S)$ in $n^{O(\alpha \log \alpha)}$ time.*

Proof. Let S and S' be composition series. First, $X(S) \cong \mathrm{Can}_{\mathbf{Graph}}(X(S))$ by Theorem 10.3.8 which implies that $S \cong Y(\mathrm{Can}_{\mathbf{Graph}}(X(S)))$ by Theorem 10.3.14.

If $S \cong S'$, then $X(S) \cong X(S')$ by Corollary 10.3.9 and $\mathrm{Can}_{\mathbf{Graph}}(X(S)) = \mathrm{Can}_{\mathbf{Graph}}(X(S'))$, so $Y(\mathrm{Can}_{\mathbf{Graph}}(X(S))) = Y(\mathrm{Can}_{\mathbf{Graph}}(X(S')))$. On the other hand, if $S \ncong S'$, then $X(S) \ncong X(S')$ by Theorem 10.3.8 and $\mathrm{Can}_{\mathbf{Graph}}(X(S)) \ncong \mathrm{Can}_{\mathbf{Graph}}(X(S'))$ so $Y(\mathrm{Can}_{\mathbf{Graph}}(X(S))) \ncong Y(\mathrm{Can}_{\mathbf{Graph}}(X(S')))$ by Theorem 10.3.14. In particular, $Y(\mathrm{Can}_{\mathbf{Graph}}(X(S))) \neq Y(\mathrm{Can}_{\mathbf{Graph}}(X(S')))$. Thus, $Y \circ \mathrm{Can}_{\mathbf{Graph}} \circ X$ is a canonical form for composition series.

By Theorems 10.3.8 and 10.3.14, X and Y can be evaluated in polynomial time since the graph $\mathrm{Can}_{\mathbf{Graph}}(X(S))$ has size $O(n^2)$ and degree $\alpha + O(1)$ by Lemma 10.3.10. Then by Theorem 8.4.7, computing the canonical form of $X(S)$ takes $n^{O(\alpha \log \alpha)}$ time. □

10.4 Algorithms for p-group isomorphism and canonization

The intermediate results of Sections 10.2 and 10.3 put us in a position to prove Theorem 10.1.1.

Theorem 10.1.1. *p-group isomorphism is decidable in $n^{\min\{(1/2)\log_p n + O(p \log p),\, \log_p n\}}$ time. In particular, $n^{(1/2)\log_p n + O(\log n / \log\log n)}$ and $n^{(1/2)\log n + O(1)}$ are upper bounds on its time complexity.*

Proof. Combining Theorems 10.1.2 and 10.3.11 yields an $n^{(1/2)\log_p n + O(p \log p)}$ time algorithm for testing isomorphism of p-groups. On the other hand, every p-group has a generating set of size at most $\log_p n$ so the generator-enumeration algorithm runs in $n^{\log_p n + O(1)}$ time for p-groups. Combining these two algorithms shows that p-group isomorphism is decidable in $n^{\min\{(1/2)\log_p n + O(p \log p),\, \log_p n\}}$ time.

Let $\alpha = \log n/(\log\log n)^2$. By upper bounding $\min\{(1/2)\log_p n + O(p\log p),\ \log_p n\}$ with $(1/2)\log_p n + O(p\log p)$ when $p \leq \alpha$ and with $\log_p n$ when $p > \alpha$, we see that $\min\{(1/2)\log_p n + O(p),\ \log_p n\}$ is upper bounded by $(1/2)\log_p n + O(\log n/\log\log n)$. The upper bound $(1/2)\log n + O(1)$ can be obtained by showing that the maximum of $(1/2)\log_p n + O(p\log p)$ for $p \leq \alpha$ is attained at $p = 2$.

□

We remark that the above algorithm relies on the $n^{O(d\log d)}$ algorithm from Theorem 8.4.7 [76, 18] for computing canonical forms of graphs of degree d rather than the faster $n^{O(d/\log d)}$ algorithm [18, 16] for testing isomorphism of such graphs. This does not change the result as $\mathsf{polylog}(d)$ factors in the exponent of the graph isomorphism testing procedure require us to choose a different cutoff α in the proof of Theorem 10.1.1 but do not affect the final result.

We now adapt our algorithm to perform p-group canonization. The main tool we are missing for this result is the ability to compute the canonical form of a p-group in $n^{\log_p n + O(1)}$ time. Given a total order on an alphabet Σ, define the *standard order* on Σ^* by $x \prec y$ if $|x| < |y|$ or $|x| = |y|$ and x comes before y lexicographically. We adapt the generator-enumeration algorithm to perform canonization using a lemma that orders the elements of a group using a generating set. We start by defining the ordering.

Definition 10.4.1. *Let G be a group with an ordered generating set $\mathbf{g} = (g_1, \ldots, g_k)$. Define a total order $\prec_{\mathbf{g}}$ on G by $x \prec_{\mathbf{g}} y$ if $w_{\mathbf{g}}(x) \prec w_{\mathbf{g}}(y)$ where each $w_{\mathbf{g}}(x) = (x_1, \ldots, x_j)$ is the first word in $\{g_1, \ldots, g_k\}^*$ under the standard ordering such that $x = x_1 \cdots x_j$.*

Lemma 10.4.2. *Let G and H be groups with ordered generating sets $\mathbf{g} = (g_1, \ldots, g_k)$ and $\mathbf{h} = (h_1, \ldots, h_k)$, and let $x, y \in G$. Then*

(a) $\prec_{\mathbf{g}}$ is a total ordering on G.

(b) if $\phi : G \to H$ is an isomorphism such that each $\phi(g_i) = h_i$, then $x \prec_{\mathbf{g}} y$ if and only if $\phi(x) \prec_{\mathbf{h}} \phi(y)$.

(c) we can decide if $x \prec_{\mathbf{g}} y$ in $O(n\,|\mathbf{g}|)$ time.

Proof. Let $S = \{g_1, \ldots, g_k\}$. For part (a), it is clear that $\prec_{\mathbf{g}}$ is a total order since $w_{\mathbf{g}} : G \to S^*$ is clearly injective and the standard ordering on S^* is a total order.

For part (b), consider an isomorphism $\phi : G \to H$ such that each $\phi(g_i) = h_i$. Then if $w_{\mathbf{g}}(x) = (x_1, \ldots, x_j)$, $w_{\mathbf{h}}(\phi(x)) = (\phi(x_1), \ldots, \phi(x_j))$. Thus, $x \prec_{\mathbf{g}} y$ if and only if $w_{\mathbf{g}}(x) \prec w_{\mathbf{g}}(y)$ (by definition of ϕ) if and only if $w_{\mathbf{h}}(\phi(x)) \prec w_{\mathbf{h}}(\phi(y))$ if and only if $x \prec_{\mathbf{h}} y$.

For part (c), it suffices to show how to compute $w_{\mathbf{g}}(x)$ in polynomial time. Consider the Cayley graph $\mathrm{Cay}(G, S)$ for the group G with generating set S. Then the word $w_{\mathbf{g}}(x)$ corresponds to the edges in the minimum length path from 1 to x in $\mathrm{Cay}(G, S)$ that comes first lexicographically. We can find this path in $O(n\,|\mathbf{g}|)$ time by visiting the nodes in breadth-first order starting with 1. At the j^{th} stage, we know $w_{\mathbf{g}}(y)$ for all $y \in G$ at a distance of at most j from the root. We then compute $w_{\mathbf{g}}(x)$ for each x at a distance of $j+1$ from the root by selecting the minimal word $w_{\mathbf{g}}(x) : g_{x,y}$ over all edges (x, y) associated with an element $g_{x,y}$ of S. $\square$

We utilize this order to permute the rows and columns of the multiplication table of the group.

Definition 10.4.3. *Let G be a group and let $\mathbf{g}$ be an ordered generating set for G. We relabel each element of G by its position in the ordering $\prec_{\mathbf{g}}$. We then permute the rows and columns of the resulting multiplication table so that the elements for the rows and columns appear in the order $1, \ldots, n$ and denote the result by $M_{\mathbf{g}}$.*

Clearly, $M_{\mathbf{g}}$ defines a group isomorphic to G. The following lemma provides a means of adapting the generator-enumeration algorithm to group canonization.

Lemma 10.4.4. *Let G and H be groups, let $\mathcal{G}_\ell$ and $\mathcal{H}_\ell$ be the collections of all ordered generating sets of G and H of size at most ℓ, and define $M_\ell(G) = \{M_{\mathbf{g}} \mid \mathbf{g} \in \mathcal{G}_\ell\}$. Then*

(a) If $G \ncong H$, then $M_\ell(G) \cap M_\ell(H) = \emptyset$.

(b) If $G \cong H$, then $M_\ell(G) = M_\ell(H)$.

Proof. For part (a), suppose $G \ncong H$ but $M \in M_\ell(G) \cap M_\ell(H)$. Then G would be isomorphic to the group defined by the multiplication table M which is also isomorphic to H.

For part (b), fix an isomorphism $\phi : G \to H$. We claim that $M_{\mathbf{g}} = M_{\phi(\mathbf{g})}$ for each $\mathbf{g} \in \mathcal{G}_\ell$. We know from Lemma 10.4.2 that for $x, y \in G$, $x \prec_{\mathbf{g}} y$ if and only if $\phi(x) \prec_{\phi(\mathbf{g})} \phi(y)$. Since $\phi(x)\phi(y) = \phi(xy)$, it follows that $M_{\mathbf{g}} = M_{\phi(\mathbf{g})}$. Therefore, $M_\ell(G) = M_\ell(H)$. □

Recall that the rank of a group is the size of a minimal generating set.

Corollary 10.4.5. *Let G be a group. Then we can compute a canonical form for G in $n^{\mathrm{rank}(G)+O(1)}$ time.*

Proof. We first determine the rank of G in $n^{\mathrm{rank}(G)+O(1)}$ time by brute force. Then we compute the set $\mathcal{G}_{\mathrm{rank}(G)}$ and choose $\mathrm{Can}_{\mathbf{Grp}}(G) = M_{\mathbf{g}}$ where $\mathbf{g} \in \mathcal{G}_{\mathrm{rank}(G)}$ to be the element that comes first lexicographically. The fact that the map defined by this computation is a canonical form is immediate from Lemma 10.4.4. □

It is now easy to adapt Theorem 10.1.1 to perform p-group canonization.

Theorem 10.4.6. *p-group canonization is in $n^{\min\{(1/2)\log_p n + O(p \log p),\, \log_p n\}}$ time.*

Proof. Let G be a p-group. Combining Theorems 10.2.4 and 10.3.15 yields an $n^{(1/2)\log_p n + O(p \log p)}$ time algorithm for group canonization while Corollary 10.4.5 gives an $n^{\log_p n + O(1)}$ time algorithm. The result then follows from the same argument used in the proof of Theorem 10.1.1.

□

Chapter 11
SOLVABLE-GROUP ISOMORPHISM

11.1 Introduction

In Chapter 10, we showed a square-root speedup over the generator-enumeration algorithm for the class of p-groups. This chapter extends that result to the class of solvable groups using Hall's theory of Sylow bases [53], which we shall introduce in this chapter.

Since the algorithm for solvable-group isomorphism presented in this chapter has much in common with the algorithm for p-group isomorphism from Chapter 10, we start by briefly reviewing that algorithm. Recall that the algorithm of Chapter 10 has two main steps:

1) an $n^{(1/2)\log_p n+O(1)}$ time Turing reduction from group isomorphism to composition-series isomorphism and
2) an algorithm for testing p-group composition series isomorphism in $n^{O(p\log p)}$ time.

Step (1) follows by bounding the number of composition series. For step (2), we construct rooted trees whose levels represent the factors in the composition series; the multiplication table is then encoded by attaching gadgets to the leaves. Since the orders of the composition factors bound the number of children at the corresponding levels of the tree and each leaf is connected to a constant number of gadgets, the resulting graph has degree at most $p+O(1)$. This yields a polynomial-time many-one reduction from composition-series isomorphism to low-degree graph isomorphism. Combining this with the $n^{O(d\log d)}$ time algorithm of Theorem 8.4.5 [76, 18] for testing isomorphism of graphs of degree at most d yields an $n^{O(p\log p)}$ time algorithm for p-group composition-series isomorphism as claimed in step (2).

As we showed in Chapter 10, combining steps (1) and (2) yields an $n^{(1/2)\log_p n+O(p\log p)}$ algorithm for p-groups (we will refer to this as the graph-isomorphism component of the p-group algorithm). This algorithm is faster than generator-enumeration when p is small and slower when it is large. (We consider a prime small if it is at most $\alpha = \log n/(\log\log n)^2$ and large if it is greater than α.) By choosing between these two algorithms according to the value of p, we obtain an $n^{(1/2)\log_p n + O(\log n/\log\log n)}$ time algorithm; this gives a square root speedup over generator enumeration regardless of the value of p.

Our main result leverages Hall's theory of Sylow bases [53] to extend this algorithm to solvable groups.

Theorem 11.1.1. *Solvable-group isomorphism is decidable in $n^{(1/2)\log_p n + O(\log n/\log\log n)}$ deterministic time.*

The algorithm for solvable groups follows the same framework but is more complicated. The main conceptual challenge is that solvable groups can have composition factors of large order as well as other composition factors of small order. This is problematic since both generator enumeration and the graph-isomorphism based p-group algorithm just described will take roughly $n^{\log_p n}$ time for a group that has many small composition factors and one large composition factor.

In order to overcome this obstacle, we need a way to (in effect) apply the graph-isomorphism component of the p-group algorithm to the part of the group that corresponds to the small prime factors while applying the generator-enumeration algorithm to the part of the group that corresponds to large prime factors. Since these two parts of a solvable group do not form a direct product decomposition, we need a way of actually combining these two algorithms since we cannot separate the group into independent parts and run the algorithms separately.

Wagner [127] gave a method for reducing the degree of the graph by restricting the isomorphism to be fixed on the quotient of G by a subgroup G_i in the composition series. If there is a subgroup G_i in the composition series whose prime divisors are all large, then the number of ways of fixing the isomorphism on the quotient G/G_i is relatively small so we can test isomorphism of the composition series. Thus, we could handle large composition factors if we had a way of moving all the large primes to the top of the composition series.

Since it is not clear that there is always a composition series with all the large primes at the top, we use a different structure. The key idea in our algorithm for solvable-group isomorphism is to use Sylow bases to separate the large and small prime divisors[1] (according to the threshold $\alpha = \log n/\log\log n$) into subgroups P_1 and P_2 of G such that $G = P_1P_2$. We call the pair (P_1, P_2) an *α-decomposition* for G and define it formally later. We also let (Q_1, Q_2) be an α-decomposition for H. The correctness of this step is guaranteed by the following lemma which follows easily from Hall's theorems [53].

Lemma 11.1.2. *For any α, solvable-group isomorphism is deterministic polynomial-time Turing-reducible to testing isomorphism of α-decompositions of the group.*

As a corollary, we obtain Theorem 2.2.2 as claimed in Chapter 2.

We then choose a composition series S_2 for P_2 and a composition series S_2' for Q_2. There is no need to choose composition series for P_1 and Q_1 since we plan to apply Wagner's degree reduction trick to these subgroups. We call the pairs (P_1, S_2) and (Q_1, S_2') *α-composition pairs* for G and H. We say that (P_1, S_2) is isomorphic to (Q_1, S_2') if there is an isomorphism from G to H that restricts to isomorphisms from P_1 to Q_1 and S_2 to S_2'. By enumerating all possible composition series as in the case for p-groups, we can reduce the problem to α-composition pair isomorphism.

Lemma 11.1.3. *Testing isomorphism of the α-decompositions (P_1, P_2) and (Q_1, Q_2) of the groups G and H is $n^{(1/2)\log_p n + O(1)}$ deterministic time Turing reducible to testing isomorphism of α-composition pairs for (P_1, P_2) and (Q_1, Q_2) where p is the smallest prime dividing the order of the group.*

It remains to show how to test if two α-composition pairs are isomorphic. Solving this problem is the main challenge in generalizing the p-group algorithm to solvable groups. As before, we accomplish this by constructing a graph. However, now our graph for G must

[1]We thank Laci Babai for suggesting this simplification. An earlier version of this chapter broke G into many factors, which made it more complicated.

represent both the decomposition $G = P_1P_2$ and the composition series S_2. We start by constructing a tree; the top of the tree corresponds to the subgroup P_1 while the bottom corresponds to S_2. The degree of the top part of the tree is reduced to a constant using Wagner's trick at the cost of a factor of $n^{\alpha+O(1)}$. Extra gadgets are used to require any isomorphism to respect the decomposition $G = P_1P_2$. The multiplication table is represented by attaching gadgets to the leaves in the same way as before. The result is a graph that has degree $\alpha + O(1)$ and represents the isomorphism class of the α-composition pair (P_1, S_2). Combining with the $n^{O(d \log d)}$ time algorithm of Theorem 8.4.5 [76, 18] for testing isomorphism of graphs of degree at most d completes the proof of Theorem 11.1.1.

As in the case of p-groups, we extend our algorithm for solvable-group isomorphism to compute canonical forms of solvable groups within the same amount of time. Later, in Chapter 12, we will show how to combine this canonization algorithm with a general collision detection framework to reduce the 1/2 in the exponent of Theorem 11.1.1 to 1/4.

In Section 11.2, we reduce solvable-group isomorphism to α-decomposition isomorphism and from α-decomposition isomorphism to α-composition pair isomorphism. In Section 11.3, we present the reduction from α-composition pair isomorphism to low-degree graph isomorphism. In Section 11.4, we derive our algorithms for solvable-group isomorphism.

11.2 *Reducing solvable-group isomorphism to α-composition pair isomorphism*

In this section, we define the notions of α-decompositions and α-composition pairs and show Turing reductions from solvable-group isomorphism to α-decomposition isomorphism and from α-decomposition isomorphism to α-composition isomorphism. The first reduction can be done in polynomial time using Hall's theorems [53] while the second follows by counting the number of composition series.

From now on, we assume for convenience that the groups G and H have the same order; if this is not the case, then G and H are not isomorphic. We let α be a parameter that we will later set to $\log n/(\log \log n)^2$. We start with the definition of an α-decomposition.

Definition 11.2.1. *Let G be a group. An* α-decomposition *of G is a pair of subgroups (P_1, P_2) such that*

(a) $G = P_1 P_2$,

(b) every prime dividing $|P_1|$ is greater than α and

(c) every prime dividing $|P_2|$ is at most α

We say that the α-decompositions (P_1, P_2) and (Q_1, Q_2) for the groups G and H are *isomorphic* if there is an isomorphism $\phi : G \to H$ such that $\phi[P_i] = Q_i$ for each i. In order to reduce solvable-group isomorphism to α-decomposition isomorphism, we now recall two of Hall's theorems. First, we need to define a Sylow basis.

Definition 11.2.2 (Hall [53], cf. [102])**.** *Let G be a group whose order has the prime factorization $n = \prod_{i=1}^{\ell} p_i^{e_i}$. A* Sylow basis *for G is a set $\{P'_i \mid 1 \le i \le \ell\}$ where each P'_i is a Sylow p_i-subgroup of G and $P'_i P'_j = P'_j P'_i$ for all i and j.*

In a Sylow basis $\{P'_i \mid 1 \le i \le \ell\}$, we will always assume that each P'_i is a Sylow p_i-subgroup of G. We say that the Sylow bases $\{P_i \mid 1 \le i \le \ell\}$ of G and $\{Q_i \mid 1 \le i \le \ell\}$ of H are *isomorphic* if there exists an isomorphism $\phi : G \to H$ such that $\phi[P_i] = Q_i$ for all i. It is easy to construct an α-decomposition from a Sylow basis by letting P_1 be the product of the Sylow subgroups that correspond to primes that are *greater* than α and letting P_2 be the product of the Sylow subgroups that correspond to primes that are *less* than α.

The following theorem is useful for proving that the reduction from solvable-group isomorphism to α-decomposition isomorphism takes polynomial time.

Theorem 11.2.3 (Hall [53], cf. [102])**.** *A group G is solvable if and only if it has a Sylow basis.*

Two Sylow bases $\{P'_i \mid 1 \le i \le \ell\}$ and $\{Q'_i \mid 1 \le i \le \ell\}$ of G are *conjugate* if there exists $g \in G$ such that for all i, $P_i'^{g} = Q'_i$.

Theorem 11.2.4 (Hall [53], cf. [102])**.** *Any two Sylow bases of a solvable group are conjugate.*

Notice that this implies that the group G has at most n Sylow bases. We also require the ability to compute a Sylow basis of a solvable group. This was shown by Kantor and Taylor [61] in the setting of permutation groups so it also holds in our case where the group is specified by its Cayley table.

Theorem 11.2.5 (Kantor and Taylor [61])**.** *A Sylow basis of a solvable group can be computed deterministically in polynomial time.*

Armed with these results, it is now easy to reduce solvable-group isomorphism to α-decomposition isomorphism. The following lemma from the introduction explains why our results are restricted to the class of solvable groups.

Lemma 11.1.2. *For any α, solvable-group isomorphism is deterministic polynomial-time Turing-reducible to testing isomorphism of α-decompositions of the group.*

Proof. Let G and H be solvable groups of order $n = \prod_{i=1}^{\ell} p_i^{e_i}$. We compute a Sylow basis $\{P_i' \mid 1 \leq i \leq \ell\}$ for G. Define $P_1 = \prod_{i:p_i>\alpha} P_i'$ and $P_2 = \prod_{i:p_i\leq\alpha} P_i'$; this is an α-decomposition for G. We compute a Sylow basis $\{Q_i' \mid 1 \leq i \leq \ell\}$ for H and consider all of its n conjugates $\left\{Q_i'^h \mid 1 \leq i \leq \ell\right\}$ where $h \in H$. For each of these, we define $Q_1 = \prod_{i:p_i<\alpha} Q_i'$ and $Q_2 = \prod_{i:p_i\leq\alpha} Q_i')$ and test if the α-decompositions (P_1, P_2) and (Q_1, Q_2) are isomorphic. We claim that $G \cong H$ if and only if (P_1, P_2) is isomorphic to one of the (Q_1, Q_2) computed above.

Clearly, if G and H are not isomorphic then no α-decomposition of G is isomorphic to an α-decomposition of H. If $\phi : G \to H$ is an isomorphism, then $\{\phi[P_i'] \mid 1 \leq i \leq \ell\}$ is a Sylow basis for H. By Theorem 11.2.4, it is equal to some conjugate of $\{Q_i' \mid 1 \leq i \leq \ell\}$. Then

$$(Q_1, Q_2) = (\prod_{i:p_i<\alpha} \phi[P_i'], \prod_{i:p_i\leq\alpha} \phi[P_i'])$$

is an α-decomposition for H that is isomorphic to (P_1, P_2) and our reduction will test if (P_1, P_2) is isomorphic to (Q_1, Q_2). □

Next, we reduce α-decomposition isomorphism to α-composition pair isomorphism. First, we define the notion of an α-composition pair.

Definition 11.2.6. *An* α-composition pair *for an α-decomposition (P_1, P_2) of a solvable group G is a pair (P_1, S_2) where S_2 is a composition series for P_2.*

For convenience, we will sometimes say that (P_1, S_2) is an α-composition pair for G. Let (P_1, S_2) and (Q_1, S_2') be a α-decompositions for G and H. Then (P_1, S_2) and (Q_1, S_2') are isomorphic if there is an isomorphism ϕ from (P_1, P_2) to (Q_1, Q_2) which restricts to an isomorphism from S_2 to S_2'.

The reduction from α-decomposition isomorphism to α-composition pair isomorphism, requires an upper bound on the number of composition series for a group and a way to enumerate all composition series. This was accomplished by Lemma 10.2.1 from Chapter 10 which we restate here for convenience.

Lemma 10.2.1. *Let G be a group. Then the number of composition series for G is at most $n^{(1/2)\log_p n + O(1)}$ where p is the smallest prime dividing the order of G. Moreover, one can enumerate all composition series for G in $n^{(1/2)\log_p n + O(1)}$ time.*

We are now ready to derive the reduction from α-decomposition isomorphism to testing isomorphism of α-composition pairs.

Lemma 11.1.3. *Testing isomorphism of the α-decompositions (P_1, P_2) and (Q_1, Q_2) of the groups G and H is $n^{(1/2)\log_p n + O(1)}$ deterministic time Turing reducible to testing isomorphism of α-composition pairs for (P_1, P_2) and (Q_1, Q_2) where p is the smallest prime dividing the order of the group.*

Proof. Let S_2 be an arbitrary composition series for P_2. For each composition series S_2' for Q_2, we test if the α-composition pairs (P_1, S_2) and (Q_1, S_2') are isomorphic. If $\phi : (P_1, P_2) \to (Q_1, Q_2)$ is an isomorphism, then $(Q_1, \phi[S_2])$ is an α-composition pair for H that is isomorphic to (P_1, S_2). Thus, the α-decompositions (P_1, P_2) and (Q_1, Q_2) are isomorphic if and only if the α-composition pair (P_1, S_2) is isomorphic to (Q_1, S_2') for some composition series S_2' for Q_2. The order of Q_2 is at most n; the smallest prime dividing the order of Q_2 is equal to the smallest prime dividing the order of H by Definition 11.2.1. The complexity then follows from Lemma 10.2.1. □

Putting together Lemmas 11.1.2 and 11.1.3 immediately results in the following corollary.

Corollary 11.2.7. *For any α, testing isomorphism of the solvable groups G and H is $n^{(1/2)\log_p n+O(1)}$ deterministic polynomial-time Turing-reducible to testing isomorphism of α-composition pairs for G and H where p is the smallest prime dividing the order of the group.*

We can also prove Turing reductions from solvable-group canonization to α-decomposition canonization and from α-decomposition canonization to α-composition canonization. For the convenience of the reader, we explicitly define canonical forms of α-decompositions and α-decomposition pairs. The definition of the canonical form of a group was already given in Definition 10.2.2.

Definition 11.2.8. *A map* $\mathrm{Can}_{\alpha\text{-}\mathbf{Decomp}}$ *is a* canonical form for α-decompositions *if for each α-decomposition (P_1, P_2) of a group G, $\mathrm{Can}_{\alpha\text{-}\mathbf{Decomp}}(P_1, P_2) = (M, \psi[P_1], \psi[M_2])$ such that the following hold.*

(a) M is an $n \times n$ matrix with entries in $[n]$.

(b) M is the multiplication table for a group that is isomorphic to G under the isomorphism $\psi : G \to [n]$.

(c) If (P_1, P_2) and (Q_1, Q_2) are α-decompositions then $(P_1, P_2) \cong (Q_1, Q_2)$ if and only if $\mathrm{Can}_{\alpha\text{-}\mathbf{Decomp}}(P_1, P_2) = \mathrm{Can}_{\alpha\text{-}\mathbf{Decomp}}(Q_1, Q_2)$.

Definition 11.2.9. *A map* $\mathrm{Can}_{\alpha\text{-}\mathbf{Pair}}$ *is a* canonical form for α-composition pairs *if for each α-composition pair $(P_1, S_2 = (P_{2,0} = 1 < \cdots < P_{2,m} = P_2))$ of an α-decomposition (P_1, P_2) of a group G, $\mathrm{Can}_{\alpha\text{-}\mathbf{Pair}}(P_1, S_2) = (M, \psi[P_1], \psi[P_{2,0}], \ldots, \psi[P_{2,m}])$ such that the following hold.*

(a) M is an $n \times n$ matrix with entries in $[n]$.

(b) M is the multiplication table for a group that is isomorphic to G under $\psi : G \to [n]$.

(c) If (P_1, S_2) and (Q_1, S_2') are α-decompositions then $(P_1, S_2) \cong (Q_1, S_2')$ if and only if $\mathrm{Can}_{\alpha\text{-}\mathbf{Pair}}(P_1, S_2) = \mathrm{Can}_{\alpha\text{-}\mathbf{Pair}}(Q_1, S_2')$.

Our canonical form reductions now follow via similar techniques.

Lemma 11.2.10. *Computing the canonical form of a solvable group is polynomial-time Turing reducible to computing canonical forms of α-decompositions for the group where p is the smallest prime dividing the order of the group.*

Proof. Let G be a solvable group of order $n = \prod_{i=1}^{\ell} p_i^{e_i}$. For each Sylow basis $\{P_i' \mid 1 \leq i \leq \ell\}$ of G, we let $P_1 = \prod_{i:p_i>\alpha} P_i'$ and $P_2 = \prod_{i:p_i\leq\alpha} P_i'$ and compute $\mathrm{Can}_{\alpha\text{-}\mathbf{Decomp}}(P_1, P_2)$. We define $\mathrm{Can}_{\mathbf{Grp}}(G)$ to be the multiplication table of the lexicographically least of these canonical forms. Since two groups are isomorphic if and only if the sets of isomorphism classes of their α-decompositions coincide, it follows that $\mathrm{Can}_{\mathbf{Grp}}$ is a canonical form. By Theorem 11.2.4, there are at most n Sylow bases for G which can be enumerated in polynomial time. Thus, the reduction can be performed in polynomial time. □

Lemma 11.2.11. *Computing the canonical form of an α-decomposition of a group is $n^{(1/2)\log_p n+O(1)}$ time Turing reducible to computing canonical forms of α-composition pairs for the group where p is the smallest prime dividing the order of the group.*

Proof. Let (P_1, P_2) be an α-decomposition of a group G. We use Lemma 10.2.1 to enumerate all of the at most $n^{(1/2)\log_p n+O(1)}$ composition series S_2 for P_2. We define $\mathrm{Can}_{\alpha\text{-}\mathbf{Decomp}}(P_1, P_2) = (M, \psi[P_1], \psi[P_{2,m}])$ where $(M, \psi[P_1], \psi[P_{2,0}], \ldots, \psi[P_{2,m}])$ is the lexicographically least canonical form of the α-composition pairs (P_1, S_2) that result from this process. It follows from Definition 11.2.9 that $\mathrm{Can}_{\alpha\text{-}\mathbf{Decomp}}$ is a canonical form. □

Combining Lemmas 11.2.10 and 11.2.11 yields the following corollary.

Corollary 11.2.12. *Computing the canonical form of a solvable group is $n^{(1/2)\log_p n+O(1)}$ time Turing reducible to computing canonical forms of α-composition pairs for the group where p is the smallest prime dividing the order of the group.*

11.3 α-composition-pair isomorphism and canonization

In this section, we show our reduction from α-composition pair isomorphism to low-degree graph isomorphism. Our reduction also extends to reducing α-composition pair canonization to computing canonical forms of low-degree graphs. Our proofs follow an outline similar to the analogous reduction from composition series isomorphism to low-degree graph isomorphism in the case of p-groups, but are more complex due to the more general structure of solvable groups.

11.3.1 Isomorphism testing

At a high level, our algorithm consists of the following steps. First, we augment our α-composition pair (P_1, P_2) by choosing an ordered generating set $\mathbf{g}$ for the subgroup P_1 (which corresponds to the large primes) to obtain the *augmented α-composition pair* $(P_1, S_2, \mathbf{g})$. We say that a mapping $\phi : G \to H$ is an isomorphism between the augmented α-decompositions $(P_1, S_2, \mathbf{g})$ and $(Q_1, S_2', \mathbf{h})$ for G and H if ϕ is an α-composition pair isomorphism for (P_1, S_2) and (Q_1, S_2') and $\phi(\mathbf{g}) = \mathbf{h}$. The reason for choosing an augmented α-composition pair is so that we can reduce the degree of the part of the graph we construct that corresponds to P_1 using the trick due to Wagner [126] mentioned in Section 11.1.

Since one can fix an ordered generating set $\mathbf{g}$ for P_1 and consider all possible ordered generating sets for Q_1, it is easy to see that α-composition pair isomorphism is $n^{\log_\alpha n + O(1)}$ Turing-reducible to augmented α-composition pair isomorphism. (Recall that we will later set $\alpha = \log n / \log\log n$ so this is $n^{O(\log n / \log\log n)}$ time and is less than the complexity we are aiming for.) We state this in the following lemma.

Lemma 11.3.1. *Testing isomorphism of the α-composition pairs (P_1, S_2) and (Q_1, S_2') for the solvable groups G and H is $n^{\log_\alpha n + O(1)}$ deterministic time Turing reducible to testing isomorphism of augmented α-composition pairs for (P_1, S_2) and (Q_1, S_2') where p is the smallest prime dividing the order of the group.*

We then construct a tree whose leaves represent the elements of G; by using the ordered generating set $\mathbf{g}$ chosen above, we are able to ensure that the degree of this tree is at most $\alpha + O(1)$. By augmenting this tree with gadgets that represent the multiplication table of the group, we obtain an object that represents the isomorphism class of the augmented α-composition pair $(P_1, P_2, \mathbf{g})$. The final step of the algorithm is to apply the following result of Babai and Luks [76, 18] mentioned in Chapter 8.

Theorem 8.4.7 (Babai and Luks [18]). *Canonization of colored graphs of degree at most d can be performed in $n^{O(d \log d)}$ time.*

The main challenge compared to p-group isomorphism is dealing with the fact that some of the prime divisors of a solvable group can be small while others may be large. This is the main reason why the correctness proof is significantly more complex than for p-groups. Since a p-group has exactly one prime divisor, it was possible to handle the cases of small and large primes separately using a graph-isomorphism based p-group algorithm (which is fast when the prime is small) and the generator-enumeration algorithm (which is fast when the prime is large). On the other hand, for solvable groups, it is necessary to design a hybrid algorithm that is fast for both cases simultaneously.

As mentioned above, the first step in the graph construction is to define a tree for an augmented α-composition pair $(P_1, P_2, \mathbf{g})$. We do this by constructing trees T_1 and T_2 whose leaves correspond to the elements of P_1 and P_2. In order to define the part of the tree corresponding to P_1, we need a way to canonically order the elements of a group given an ordered generating set. This is accomplished by Definition 10.4.1 and Lemma 10.4.2. We restate them here for convenience.

Definition 10.4.1. *Let G be a group with an ordered generating set $\mathbf{g} = (g_1, \ldots, g_k)$. Define a total order $\prec_{\mathbf{g}}$ on G by $x \prec_{\mathbf{g}} y$ if $w_{\mathbf{g}}(x) \prec w_{\mathbf{g}}(y)$ where each $w_{\mathbf{g}}(x) = (x_1, \ldots, x_j)$ is the first word in $\{g_1, \ldots, g_k\}^*$ under the standard ordering such that $x = x_1 \cdots x_j$.*

Lemma 10.4.2. *Let G and H be groups with ordered generating sets $\mathbf{g} = (g_1, \ldots, g_k)$ and $\mathbf{h} = (h_1, \ldots, h_k)$, and let $x, y \in G$. Then* *(a)* $\prec_{\mathbf{g}}$ *is a total ordering on* G.

(b) if $\phi : G \to H$ *is an isomorphism such that each* $\phi(g_i) = h_i$*, then* $x \prec_{\mathbf{g}} y$ *if and only if* $\phi(x) \prec_{\mathbf{h}} \phi(y)$.

(c) we can decide if $x \prec_{\mathbf{g}} y$ *in* $O(n\,|\mathbf{g}|)$ *time.*

Now we can define the tree that corresponds to P_1. We do this by choosing a balanced binary tree whose leaves are elements of P_1. The choice of this tree is arbitrary so long as it depends only on $\prec_{\mathbf{g}}$. The reason for constructing the trees for P_1 and P_2 separately is that this allows us to ensure that the tree for P_1 has only constant degree. Otherwise, it would have degree $\Omega(n)$ for groups divisible by large primes which would result in a very slow algorithm. Later on, we will combine the trees for P_1 and P_2 to obtain a tree whose leaves correspond to elements of G.

Definition 11.3.2. *Let* P_1 *be a group with ordered generating set* $\mathbf{g} = (g_1, \ldots, g_k)$*. To construct the rooted tree* $T(P_1, \mathbf{g})$*, we create a leaf node for each element of* P_1 *and color each node by the number that corresponds to its position in the ordering* $\prec_{\mathbf{g}}$*; we then arrange the nodes on a line from smallest to largest according to their colors. We attach a parent node to each pair of adjacent leaves starting with the smallest pair; if* $|P_1|$ *is odd, we attach a single parent node to the last leaf. We then arrange the parent nodes just generated on a line according to the ordering on their children and add new parent nodes for them in the same way. We continue in this manner until we obtain a single root node from which all the leaves are descended; this yields the tree* $T(P_1, \mathbf{g})$.

Next, we define the tree $T(S_2)$ for the S_2 using Definition 10.3.1 from Chapter 10. We also need a way to combine the trees for P_1 and S_2. For this, we need the notion of a leaf product from Definition 10.3.2. We are now finally in a position to define the tree for a augmented α-composition pair.

Definition 11.3.3. *Let* $(P_1, S_2, \mathbf{g})$ *be an augmented* α*-composition pair for a solvable group* G*. We define* $T(P_1, S_2, \mathbf{g}) = T(P_1, \mathbf{g}) \odot T(S_2)$.

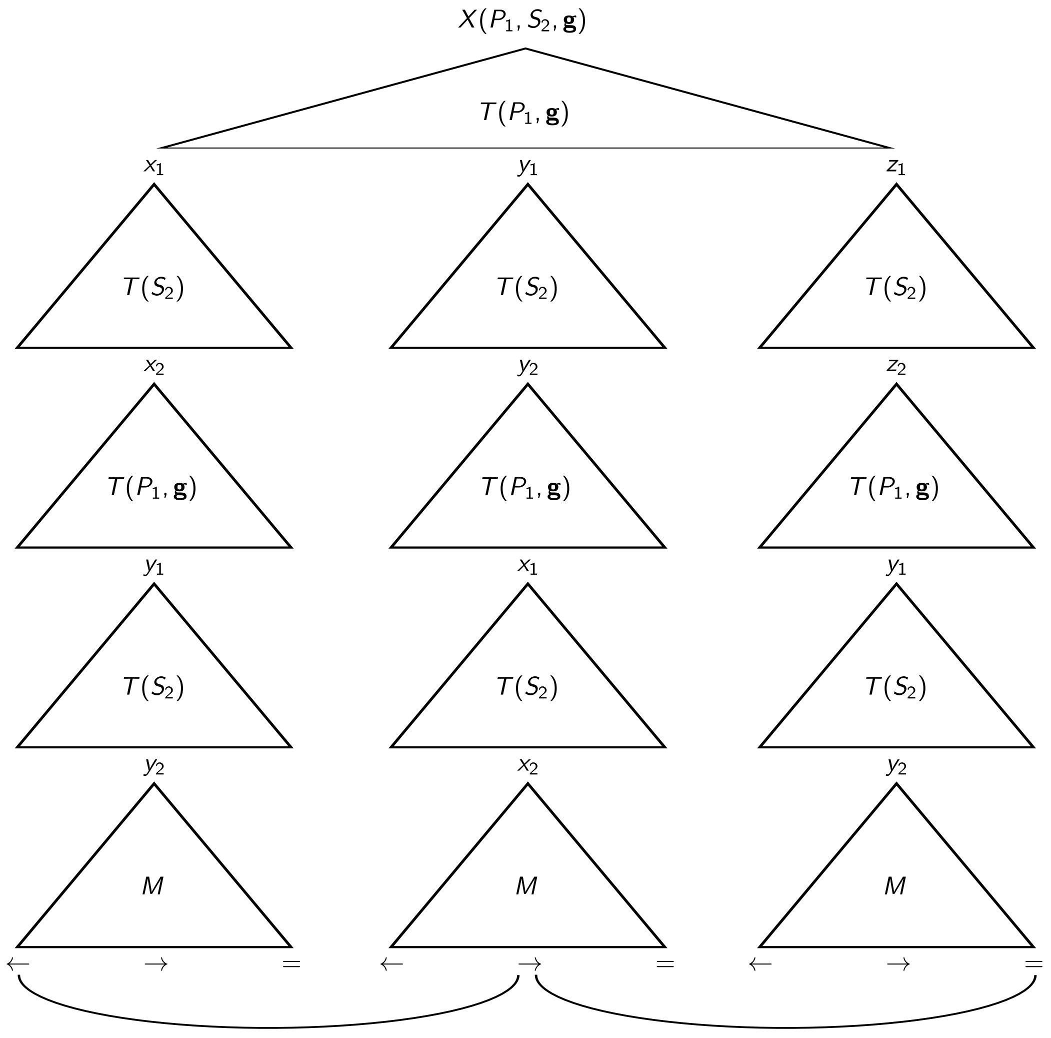


Figure 11.1: The graph $X(P_1, S_2, \mathbf{g})$ with the multiplication gadget for $xy = z$ where $z = xy$, $*^{-1}(x) = (x_1, x_2)$, $*^{-1}(y) = (y_1, y_2)$ and $*^{-1}(z) = (z_1, z_2)$

As in the case of p-groups, we cannot attach the aforementioned multiplication gadgets directly to the tree $T(P_1, S_2, \mathbf{g})$ because each leaf be attached to n gadgets and would thus have degree $\Omega(n)$; this would cause our algorithm to be extremely slow. We resolve this by utilizing the leaf product of $T(P_1, S_2, \mathbf{g})$ with itself so that each multiplication gadget is only attached to a constant number of leaves.

The following notation is convenient as it allows us to easily associate elements of G with nodes in the tree $T(P_1, S_2, \mathbf{g})$. Let $* : \{(x_1, x_2) \mid x_i \in P_i\} \to G$ by $*(x_1, x_2) = x_1 x_2$ and note that this is a bijection. Similarly, we define $\bullet : \{(x_1, x_2) \mid x_i \in Q_i\} \to H$ by $\bullet(x_1, x_2) = x_1 x_2$. We can then represent each $x \in G$ by the node $*^{-1}(x)$ in $T(P_1, S_2, \mathbf{g})$ and attach the gadget for each multiplication rule $xy = z$ to the nodes $*^{-1}(x)$, $*^{-1}(y)$ and $*^{-1}(z)$. We formalize this in the following definition.

Definition 11.3.4. *Let $(P_1, S_2, \mathbf{g})$ be an augmented α-composition pair for a solvable group G and define M to be the tree with a root connected to three nodes $\leftarrow$, $\rightarrow$ and $=$ with colors "left", "right" and "equals" respectively. We construct $X(P_1, S_2, \mathbf{g})$ by starting with the tree $T(P_1, S_2, \mathbf{g}) \odot T(P_1, S_2, \mathbf{g}) \odot M$ and connecting multiplication gadgets to the leaf nodes. For each $x, y \in G$, we create the path $((*^{-1}(x), *^{-1}(y), \leftarrow), (*^{-1}(y), *^{-1}(x), \rightarrow), (*^{-1}(xy), *^{-1}(y), =))$. We color each node $(x_1, 1)$ where $x_1 \in P_1$ "second identity." Finally, we color the remaining nodes "internal."*

The graph $X(P_1, S_2, \mathbf{g})$ can be thought of a rooted tree with edges added between some nodes at the same levels. The edges from the original tree are called *tree edges* and the edges between nodes at the same level are called *cross edges*. We show $X(P_1, S_2, \mathbf{g})$ in Figure 11.1.

The correctness of our reduction is based on the fact that two augmented composition pairs $(P_1, S_2, \mathbf{g})$ and $(Q_1, S_2', \mathbf{h})$ are isomorphic if and only if $X(P_1, S_2, \mathbf{g})$ and $X(Q_1, S_2', \mathbf{h})$ are isomorphic. We prove this in the remainder of this subsection.

Some additional terminology is required for the proof. We define **ACP** to be the class of augmented composition pairs for finite solvable groups and let **ACPTree** be the class of graphs that are isomorphic to the graph $X(P_1, S_2, \mathbf{g})$ for some augmented composition pair $(P_1, S_2, \mathbf{g})$. We overload the symbol X from Definition 11.3.4 by defining $X(\phi) : X(P_1, S_2, \mathbf{g}) \to X(Q_1, S_2', \mathbf{h})$ to be $\phi\big|_{P_1} \odot \phi\big|_{P_2} \odot \phi\big|_{P_1} \odot \phi\big|_{P_2} \odot \mathrm{id}_M$ for each α-composition pair isomorphism $\phi : (P_1, S_2, \mathbf{g}) \to (Q_1, S_2', \mathbf{h})$.

In order to prove the correctness of our reduction, we need to show that the augmented α-composition pairs $(P_1, S_2, \mathbf{g})$ and $(Q_1, S_2', \mathbf{h})$ are isomorphic if and only if the graphs $X(P_1, S_2, \mathbf{g})$ and $X(Q_1, S_2', \mathbf{h})$ are isomorphic. The forward direction of the implication is equivalent to the assertion that $X_{(P_1,S_2,\mathbf{g}),(Q_1,S_2',\mathbf{h})} : \mathrm{Iso}((P_1,S_2,\mathbf{g}),(Q_1,S_2',\mathbf{h})) \to \mathrm{Iso}(X(P_1,S_2,\mathbf{g}),X(Q_1,S_2',\mathbf{h}))$ is well-defined. Proving the converse is more difficult and is one of the main lemmas of this subsection.

Lemma 11.3.5. *Let $(P_1,S_2,\mathbf{g})$ and $(Q_1,S_2',\mathbf{h})$ be augmented α-composition pairs for the solvable groups G and H. Then the map*

$$X_{(P_1,S_2,\mathbf{g}),(Q_1,S_2',\mathbf{h})} : \mathrm{Iso}((P_1,S_2,\mathbf{g}),(Q_1,S_2',\mathbf{h})) \to \mathrm{Iso}(X(P_1,S_2,\mathbf{g}),X(Q_1,S_2',\mathbf{h}))$$

is well-defined.

Before proceeding with the proof, it is convenient to introduce additional notation. Let $x, y \in G$. Consider the sequence of nodes that starts at $*^{-1}(x)$, follows tree edges (away from the root) to a node colored "left", follows a cross edge to a node colored "right", then follows tree edges (towards the root) to $*^{-1}(y)$, follows tree edges (away from the root) back to the same node colored "right" and finally follows a cross edge to a node colored "equal"; we call this a *W-sequence* from x to y to xy since its shape resembles a W (see Figure 11.1). Since W-sequences correspond to multiplication gadgets, there is exactly one W-sequence from $*^{-1}(x)$ to $*^{-1}(y)$: namely, the one that results from the multiplication gadget

$$((*^{-1}(x), *^{-1}(y), \leftarrow), (*^{-1}(y), *^{-1}(x), \rightarrow), (*^{-1}(xy), *^{-1}(y), =)).$$

Therefore, we denote *the* W-sequence from x to y to xy by $W(x,y)$. We now proceed with our proof.

Proof. Consider the augmented α-composition pairs $(P_1,S_2,\mathbf{g})$ and $(Q_1,S_2',\mathbf{h})$ for the solvable groups G and H. Let $\phi : (P_1,S_2,\mathbf{g}) \to (Q_1,S_2',\mathbf{h})$ be an isomorphism and let $P_{2,0} = 1 \lhd \cdots \lhd P_{2,m} = P_2$ and $Q_{2,0} = 1 \lhd \cdots \lhd Q_{2,m} = Q_2$ be the subgroup chains for S_2 and S_2'. Because $\phi(\mathbf{g}) = \mathbf{h}$, it follows from Lemma 10.4.2 that $\phi\big|_{P_1}$ extends to a unique isomorphism between the rooted colored trees $T(P_1,\mathbf{g})$ and $T(Q_1,\mathbf{h})$. Moreover, since each $\phi[P_{2,i}] = Q_{2,i}$, we see that $\phi\big|_{P_2}$ extends to a unique isomorphism from $T(S_2)$ to $T(S_2')$.

Thus, $\phi\big|_{P_1} \odot \phi\big|_{P_2}$ is an isomorphism from $T(P_1, \mathbf{g}) \odot T(S_2)$ to $T(Q_1, \mathbf{h}) \odot T(S_2')$; therefore, $X(\phi) = \phi\big|_{P_1} \odot \phi\big|_{P_2} \odot \phi\big|_{P_1} \odot \phi\big|_{P_2} \odot \mathrm{id}_M$ is a tree isomorphism.

Let $x, y \in G$ and let $*^{-1}(x) = (x_1, x_2)$. Then $X(\phi)$ maps $*^{-1}(x)$ to $(\phi(x_1), \phi(x_2)) = \bullet^{-1}(\phi(x))$ as $\phi(x) = \phi(x_1)\phi(x_2)$. Similarly, recalling that we identified expressions of the forms $((x_1, x_2), (y_1, y_2))$ and (x_1, x_2, y_1, y_2), we see that $X(\phi)$ maps $(*^{-1}(x), *^{-1}(y))$ to $(\bullet^{-1}(\phi(x)), \bullet^{-1}(\phi(y)))$

Consider the path

$$((*^{-1}(x), *^{-1}(y), \leftarrow), (*^{-1}(y), *^{-1}(x), \rightarrow), (*^{-1}(xy), *^{-1}(y), =))$$

in $X(P_1, S_2, \mathbf{g})$. The image of this path under $X(\phi)$ is

$$((\bullet^{-1}(\phi(x)), \bullet^{-1}(\phi(y)), \leftarrow), (\bullet^{-1}(\phi(y)), \bullet^{-1}(\phi(x)), \rightarrow), (\bullet^{-1}(\phi(xy)), \bullet^{-1}(\phi(y)), =)).$$

By Definition 11.3.4, this path is one of the multiplication gadgets in $X(Q_1, S_2', \mathbf{h})$. Thus, $X(\phi)$ maps each W-sequence in $X(P_1, S_2, \mathbf{g})$ to a W-sequence in $X(Q_1, S_2', \mathbf{h})$. Moreover, $X(\phi)$ maps each node $(x_1, 1)$ to $(\phi(x_1), 1)$, so it respects the "second identity" color. This implies that $X(P_1, S_2, \mathbf{g}) \cong X(Q_1, S_2', \mathbf{h})$ since both graphs have the same number of multiplication gadgets (and hence the same number of W-sequences). $\square$

In order to show if that if the graphs $X(P_1, S_2, \mathbf{g})$ and $X(Q_1, S_2', \mathbf{h})$ are isomorphic then so are the augmented α-composition pairs $(P_1, S_2, \mathbf{g})$ and $(Q_1, S_2', \mathbf{h})$, it suffices to show that the map $X_{(P_1,S_2,\mathbf{g}),(Q_1,S_2',\mathbf{h})} : \mathrm{Iso}((P_1, S_2, \mathbf{g}), (Q_1, S_2', \mathbf{h})) \to \mathrm{Iso}(X(P_1, S_2, \mathbf{g}), X(Q_1, S_2', \mathbf{h}))$ is surjective. This is the key to our correctness proof and implies that augmented α-composition pair isomorphism reduces to testing isomorphism of the resulting graphs. To do this, we need to show that every isomorphism from $X(P_1, S_2, \mathbf{g})$ to $X(Q_1, S_2', \mathbf{h})$ can be written as a leaf product of group isomorphisms. We accomplish this by restricting the isomorphism between the graphs to certain subsets of nodes and showing that the isomorphism is the leaf product of these restrictions (which turn out to be group isomorphisms). An isomorphism $\theta : X(P_1, S_2, \mathbf{g}) \to X(Q_1, S_2', \mathbf{h})$ induces the bijection $\phi = \bullet \circ \theta \circ *^{-1} : G \to H$. We call this ϕ the *induced bijection* for θ.

Lemma 11.3.6. *Let $X(P_1, S_2, \mathbf{g})$ and $X(Q_1, S_2', \mathbf{h})$ be augmented α-composition pairs for the solvable groups G and H, let $\theta : X(P_1, S_2, \mathbf{g}) \to X(Q_1, S_2', \mathbf{h})$ be an isomorphism and let ϕ be its induced bijection. Then*

(a) $\phi : G \to H$ is a group isomorphism,

(b) $\phi_1 = \phi\big|_{P_1} : P_1 \to Q_1$ and $\phi_2 = \phi\big|_{P_2} : P_2 \to Q_2$ are group isomorphisms,

(c) $\theta = \phi_1 \odot \phi_2 \odot \phi_1 \odot \phi_2 \odot \mathrm{id}_M$ and

(d) $\phi : (P_1, S_2, \mathbf{g}) \to (Q_1, S_2', \mathbf{h})$ is an augmented α-composition pair isomorphism.

Proof. Let us start with part (a). It follows from the assumption that θ is an isomorphism (and hence bijective) that ϕ is a bijection.

Let $x, y \in G$. Now, θ maps the nodes $*^{-1}(x)$ and $*^{-1}(y)$ in $X(P_1, S_2, \mathbf{g})$ to $\bullet^{-1}(\phi(x))$ and $\bullet^{-1}(\phi(y))$ by definition of ϕ. It follows that θ maps the W-sequence $W(x, y)$ from x to y to xy in $X(P_1, S_2, \mathbf{g})$ to the W-sequence $W(\phi(x), \phi(y))$ in $X(Q_1, S_2', \mathbf{h})$. Now, since θ maps $*^{-1}(xy)$ to $\bullet^{-1}(\phi(xy))$, it follows that the W-sequence $W(\phi(x), \phi(y))$ in $X(Q_1, S_2', \mathbf{h})$ is from $\phi(x)$ to $\phi(y)$ to $\phi(xy)$. Therefore, by Definition 11.3.4, $\phi(xy) = \phi(x)\phi(y)$ so ϕ is a group isomorphism.

Now we prove (b). Let $x_1 \in P_1$. Because θ respects the "second identity" color, it follows that it maps $(x_1, 1)$ to $(x_1', 1)$ for some $x_1' \in Q_1$. Then $x_1' = \phi(x_1)$ which implies that $\phi[P_1] = Q_1$.

Now let $x_2 \in P_2$. Because ϕ is an isomorphism, $\phi(1) = 1$; thus, θ sends the node $(1, 1)$ to $(1, 1)$ which implies that it maps 1 to 1. Thus, for some $x_2' \in Q_2$,

$$\theta(1, x_2) = (1, x_2')$$
$$\theta(*^{-1}(x_2)) = \bullet^{-1}(x_2')$$
$$\phi(x_2) = x_2'.$$

Thus, $\theta(1, x_2) = (1, \phi(x_2))$ so $\phi[P_2] = Q_2$ and ϕ_2 is a group isomorphism.

For part (c), let $x, y \in G$ and $*^{-1}(x) = (x_1, x_2)$. By part (b), θ sends the node x_1 to

$\phi_1(x_1)$. Therefore, for some $x_2' \in Q_2$,

$$\begin{aligned}\theta(x_1, x_2) &= (\phi(x_1), x_2') \\ \bullet(\theta(x_1, x_2)) &= \phi(x_1)x_2' \\ \phi(x) &= \phi(x_1)x_2'.\end{aligned}$$

Since $\phi(x) = \phi(x_1)\phi(x_2)$, this implies that $x_2' = \phi(x_2)$ so θ maps $*^{-1}(x) = (x_1, x_2)$ to $\bullet^{-1}(\phi(x)) = (\phi(x_1), \phi(x_2))$.

Now consider a node $(*^{-1}(x), *^{-1}(y), \ell)$ where $x, y \in G$ and $\ell \in \{\leftarrow, \rightarrow, =\}$. As $(*^{-1}(x), *^{-1}(y))$ is in the subtree rooted at $*^{-1}(x)$, θ sends it to a node of the form $(\bullet^{-1}(\phi(x)), \bullet^{-1}(b))$ for some $b \in H$. Similarly, θ maps the node $(*^{-1}(y), *^{-1}(x))$ to a node of the form $(\bullet^{-1}(\phi(y)), \bullet^{-1}(a))$ for some $a \in H$. Now, because $(*^{-1}(x), *^{-1}(y))$ and $(*^{-1}(y), *^{-1}(x))$ are in the W-sequence from x to y to xy, $(\bullet^{-1}(\phi(x)), \bullet^{-1}(b))$ and $(\bullet^{-1}(\phi(y)), \bullet^{-1}(a))$ are in the W-sequence from $\phi(x)$ to $\phi(y)$ to $\phi(xy)$. Then by Definition 11.3.4, $a = \phi(x)$ and $b = \phi(y)$. Therefore, θ maps $(*^{-1}(x), *^{-1}(y))$ to $(*^{-1}(\phi(x)), *^{-1}(\phi(y)))$. Because of the coloring of the leaves in Definition 11.3.4, it follows that $\theta = \phi_1 \odot \phi_2 \odot \phi_1 \odot \phi_2 \odot \mathrm{id}_M$.

Finally, let us prove part (d). We already know that ϕ is a group isomorphism by part (a). By part (b), we know that each $\phi[P_i] = Q_i$.

Let $P_{2,0} = 1 \lhd \cdots \lhd P_{2,m} = P_2$ and $Q_{2,0} = 1 \lhd \cdots \lhd Q_{2,m} = Q_2$ be the subgroup chains for S_2 and S_2'. We need to show that each $\phi[P_{2,i}] = Q_{2,i}$. By part (c), θ maps $(1, 1)$ in $X(P_1, S_2, \mathbf{g})$ to $(1, 1)$ in $X(Q_1, S_2', \mathbf{h})$. Now the path from the root of $X(P_1, S_2, \mathbf{g})$ to $(1, 1)$ contains the nodes $(1, P_{2,m}), \ldots, (1, P_{2,0})$ (in that order). Moreover, the descendants of the node $(1, P_{2,i})$ that are in $P_1 \times P_2$ are $\{(1, x_2) \mid x_2 \in P_{2,i}\}$. Similarly, the path from the root of $X(Q_1, S_2', \mathbf{h})$ to $(1, 1)$ contains the nodes $(1, Q_{2,m}), \ldots, (1, Q_{2,0})$ (in that order) and the descendants of the node $(1, Q_{2,i})$ that are also in $Q_1 \times Q_2$ are $\{(1, x_2') \mid x_2' \in Q_{2,i}\}$. Therefore, θ maps each set $\{(1, x_2) \mid x_2 \in P_{2,i}\}$ to $\{(1, x_2') \mid x_2' \in Q_{2,i}\}$. Then, by definition of ϕ, $\phi[P_{2,i}] = Q_{2,i}$ and part (d) is proved. $\square$

We now prove that $X_{(P_1,S_2,\mathbf{g}),(Q_1,S_2',\mathbf{h})}$ is bijective. For isomorphism testing, we only need to show that it is surjective. However, we will need it to be injective later when we discuss canonical forms.

Theorem 11.3.7. *Let $(P_1, S_2, \mathbf{g})$ and $(Q_1, S_2', \mathbf{h})$ be augmented α-composition pairs for the solvable groups G and H. Then $X_{(P_1,S_2,\mathbf{g}),(Q_1,S_2',\mathbf{h})}$ is a bijection. Moreover, both $X(P_1, S_2, \mathbf{g})$ and $X(\phi)$ where $\phi \in \mathrm{Iso}((P_1, S_2, \mathbf{g}), (Q_1, S_2', \mathbf{h}))$ can be computed in polynomial time.*

Proof. The graph $X_{(P_1,S_2,\mathbf{g}),(Q_1,S_2',\mathbf{h})}$ is well-defined by Lemma 11.3.5. Let $\theta : X(P_1, S_2, \mathbf{g}) \to X(Q_1, S_2', \mathbf{h})$ be an isomorphism. By Lemma 11.3.6, the induced bijection $\phi : (P_1, S_2, \mathbf{g}) \to (Q_1, S_2', \mathbf{h})$ is an isomorphism and $\theta = \phi_1 \odot \phi_2 \odot \phi_1 \odot \phi_2 \odot \mathrm{id}_M$ where each $\phi_i = \phi\big|_{P_i}$. Then $X(\phi) = \theta$ so $X_{(P_1,S_2,\mathbf{g}),(Q_1,S_2',\mathbf{h})}$ is surjective.

Let $\phi, \psi : (P_1, S_2, \mathbf{g}) \to (Q_1, S_2', \mathbf{h})$ be isomorphisms and suppose that $X(\phi) = X(\psi)$. Then $\phi_1 \odot \phi_2 \odot \phi_1 \odot \phi_2 \odot \mathrm{id}_M = \psi_1 \odot \psi_2 \odot \psi_1 \odot \psi_2 \odot \mathrm{id}_M$ where each $\phi_i = \phi\big|_{P_i}$ and each $\psi_i = \psi\big|_{P_i}$. Therefore, each $\phi_i = \psi_i$ so $X_{(P_1,S_2,\mathbf{g}),(Q_1,S_2',\mathbf{h})}$ is injective. □

Correctness of our reduction now follows.

Corollary 11.3.8. *Let $(P_1, S_2, \mathbf{g})$ and $(Q_1, S_2', \mathbf{h})$ be augmented α-composition pairs for the solvable groups G and H. Then $(P_1, S_2, \mathbf{g}) \cong (Q_1, S_2', \mathbf{h})$ if and only if $X(P_1, S_2, \mathbf{g}) \cong X(Q_1, S_2', \mathbf{h})$.*

Because X is defined in terms of leaf products of structures that can be computed in polynomial time, it is immediate that X can also be evaluated in polynomial time.

Lemma 11.3.9. *Let $(P_1, S_2, \mathbf{g})$ and $(Q_1, S_2', \mathbf{h})$ be augmented α-composition pairs for the solvable groups G and H and let $\phi : (P_1, S_2, \mathbf{g}) \to (Q_1, S_2', \mathbf{h})$ be an isomorphism. Then both $X(P_1, S_2, \mathbf{g})$ and $X(\phi)$ can be computed in polynomial time.*

The last ingredient that we require for our algorithm for augmented α-composition pair isomorphism is a bound on the degree of the graph.

Lemma 11.3.10. *Let $(P_1, S_2, \mathbf{g})$ be an augmented α-composition pair for the solvable group G. Then the graph $X(P_1, S_2, \mathbf{g})$ has degree at most $\max\{\alpha + 1, 4\}$ and size $O(n^2)$.*

Proof. The trees $T(P_1, \mathbf{g})$, $T(S_2)$ and M have degrees 3, at most $\alpha + 1$ and 3 respectively. Since $|P_1|\,|P_2| = n$, the size of $T(P_1, \mathbf{g}) \odot T(S_2)$ is $O(n)$. Thus, $T(P_1, \mathbf{g}) \odot T(S_2) \odot T(P_1, \mathbf{g}) \odot T(S_2) \odot M$ has size $O(n^2)$ and degree at most $\max\{\alpha + 1, 4\}$. □

Finally, we obtain our result for augmented α-composition pair isomorphism.

Theorem 11.3.11. *Let $(P_1, S_2, \mathbf{g})$ and $(Q_1, S_2', \mathbf{h})$ be augmented α-composition pairs for the solvable groups G and H. Then we can test if $(P_1, S_2, \mathbf{g}) \cong (Q_1, S_2', \mathbf{h})$ in $n^{O(\alpha \log \alpha)}$ time.*

Proof. By Lemma 11.3.9, we can compute the graphs $X(P_1, S_2, \mathbf{g})$ and $X(Q_1, S_2', \mathbf{h})$ in polynomial time. By Lemma 11.3.10 and Theorem 8.4.7, we can decide if $X(P_1, S_2, \mathbf{g}) \cong X(Q_1, S_2', \mathbf{h})$ in $n^{O(\alpha \log \alpha)}$ time. Finally, Corollary 11.3.8 tells us that $(P_1, S_2, \mathbf{g}) \cong (Q_1, S_2', \mathbf{h})$ if and only if $X(P_1, S_2, \mathbf{g}) \cong X(Q_1, S_2', \mathbf{h})$. □

Using Lemma 11.3.1, we obtain the following corollary.

Corollary 11.3.12. *Let (P_1, S_2) and (Q_1, S_2') be α-composition pairs for the solvable groups G and H. Then we can test if $(P_1, S_2) \cong (Q_1, S_2')$ in $n^{\log_\alpha n + O(\alpha \log \alpha)}$ time.*

11.3.2 Canonization

In this subsection, we extend our results for testing isomorphism of α-composition pairs to canonization. As we shall we in Chapter 12, this result can be leveraged to obtain faster algorithms for solvable-group isomorphism via collision arguments. Our canonization algorithm requires another map Y that reverses the action of X by sending back to the augmented α-composition pairs from which they arise. We start with the definition for Y. As with X, we overload notation so that Y can also be applied to isomorphisms between graphs.

Definition 11.3.13. *For each augmented α-composition pair $(P_1, S_2, \mathbf{g})$ for a solvable group G and each graph $A \cong X(P_1, S_2, \mathbf{g})$, we fix an arbitrary isomorphism $\pi : X(P_1, S_2, \mathbf{g}) \to A$. Let $P_{2,0} = 1 \lhd \cdots \lhd P_{2,m} = P_2$ be the subgroup chain for S_2. Then we define $Y(A) = (\pi[P_1 \times \{1\}], \pi[\{1\} \times P_{2,0}] \lhd \cdots \lhd \pi[\{1\} \times P_{2,m}], \pi(\mathbf{g}))$.*

Here, $\pi[\{(x_1, x_2) \mid x_i \in P_i\}]$ is interpreted as a group containing each $\pi[\{1\} \times P_{2,i}]$ as a subgroup. For each $x_i, y_i, z_i \in P_i$, we define $\pi(x_1, x_2)\pi(y_1, y_2) = \pi(z_1, z_2)$ if and only if there exists a path $(a_{\pi(x)}a_{\pi(y)}, a_{\pi(z)})$ colored ("left", "right", "equals"), such that $a_{\pi(x)}$, $a_{\pi(y)}$ and $a_{\pi(z)}$ are descendants of the nodes $\pi(x_1, x_2)$, $\pi(y_1, y_2)$ and $\pi(z_1, z_2)$ in the image of the tree $T(P_1, \mathbf{g}) \odot T(S_2) \odot T(P_1, \mathbf{g}) \odot T(S_2) \odot M$ under π.

Let $(P_1, S_2, \mathbf{g})$ and $(Q_1, S_2', \mathbf{h})$ be augmented α-composition pairs for the groups G and H and consider the graphs $A \cong X(P_1, S_2, \mathbf{g})$ and $A' \cong X(Q_1, S_2', \mathbf{h})$. Let $\pi : X(P_1, S_2, \mathbf{g}) \to A$ and $\pi' : X(Q_1, S_2', \mathbf{h}) \to A'$ be the fixed isomorphisms chosen above. Then for each isomorphism $\theta : A \to A'$, we define $Y(\theta) : \pi[\{(x_1, x_2) \mid x_i \in P_i\}] \to \pi'[\{(x_1, x_2) \mid x_i \in Q_i\}]$ to be $\theta\big|_{\pi[\{(x_1,x_2) \mid x_i \in P_i\}]}$.

As for X, we define $Y_{A,A'} : \mathrm{Iso}(A, A') \to \mathrm{Iso}(Y(A), Y(A'))$ by $\theta \mapsto Y(\theta)$ for each pair of graphs $A, A' \in \mathbf{ACPTree}$.

Our first step is to show that Y is well-defined. Once this is proved, we can leverage Theorem 11.3.7 to show that each $Y_{A,A'}$ is bijective. This allows us to define a canonical form for augmented α-composition pairs in terms of $\mathrm{Can}_{\mathbf{Graph}}$, X and Y.

Lemma 11.3.14. *Let $(P_1, S_2, \mathbf{g})$ be an augmented α-composition pair for the solvable group G, let A be a graph and let $\pi : X(P_1, S_2, \mathbf{g}) \to A$ be an isomorphism. Then $Y(A)$ is a well-defined augmented composition pair and can be computed in polynomial time. Moreover, $Y(\pi) : (P_1, S_2, \mathbf{g}) \to Y(A)$ is an isomorphism.*

Proof. We claim that $\pi[\{(x_1, x_2) \mid x_i \in P_i\}]$ is indeed a group if interpreted according to Definition 11.3.13. Let $x_i, y_i, z_i \in P_i$. Then $\pi(x_1, x_2)\pi(y_1, y_2) = \pi(z_1, z_2)$ if and only if there exists a path $(a_{\pi(x)}a_{\pi(y)}, a_{\pi(z)})$ colored ("left", "right", "equals"), such that $a_{\pi(x)}$, $a_{\pi(y)}$ and $a_{\pi(z)}$ are descendants of the nodes $\pi(x_1, x_2)$, $\pi(y_1, y_2)$ and $\pi(z_1, z_2)$ in A. Since π is an isomorphism, this is equivalent to the existence of a path $(a_x a_y, a_z)$ colored ("left", "right", "equals"), such that a_x, a_y and a_z are descendants of the nodes (x_1, x_2), (y_1, y_2) and (z_1, z_2) in $X(P_1, S_2, \mathbf{g})$.

This is in turn equivalent to the existence of a W-sequence from x to y to z where $x = x_1x_2$, $y = y_1y_2$ and $z = z_1z_2$. By definition, this W-sequence exists if and only if $xy = z$.

Therefore, $\pi[\{(x_1, x_2) \mid x_i \in P_i\}]$ is a group and $Y(\pi)$ is a group isomorphism from G to $\pi[\{(x_1, x_2) \mid x_i \in P_i\}]$. It is immediate that $Y(A)$ is an augmented α-composition pair and $Y(\pi)$ is an augmented α-composition pair isomorphism.

Now we show how to compute $Y(A)$ in polynomial time. Let $\ell = \lceil \log |P_1| \rceil$ and let the subgroup chain for S_2 be $P_{2,0} = 1 \lhd \cdots \lhd P_{2,m}$. Then ℓ is the height of $T(P_1, \mathbf{g})$ and m is the height of $T(S_2)$. Thus, by Definition 11.3.4, $\pi[P_1 \times \{1\}]$ consists of the nodes in A colored "second identity" at a depth of $\ell + m$ from the root.

To compute each $\pi[\{1\} \times P_{2,k}]$, we first find the node $\pi(1, 1)$; this is the identity element of the group $\pi[\{(x_1, x_2) \mid x_i \in P_i\}]$. The node $\pi(1, P_{2,k})$ is the node on the path from the root to $\pi(1, 1)$ in A that is at a distance of $\ell + k$ from the root. Then, by Definition 11.3.4, each $\pi[\{1\} \times P_{2,k}]$ consists of the nodes in A descended from $\pi(1, P_{2,k})$ that are at a distance of $m - k$ from $\pi(1, P_2)$. □

Now we can show that each $Y_{A,A'}$ is surjective.

Theorem 11.3.15. *Consider the graphs $A, A' \in$ **ACPTree**. Then $Y_{A,A'}$ is a bijection and both $Y(A)$ and $Y(\theta)$ where $\theta \in \mathrm{Iso}(Y(A), Y(A'))$ can be computed in polynomial time.*

Proof. Let $(P_1, S_2, \mathbf{g})$ and $(Q_1, S_2', \mathbf{h})$ be augmented α-composition pairs for the solvable groups G and H such that $\pi : X(P_1, S_2, \mathbf{g}) \to A$, $\pi' : X(Q_1, S_2', \mathbf{h}) \to A'$ and $\theta : A \to A'$ are isomorphisms.

First, we observe that Y respects composition and let $\psi = \theta\pi : X(P_1, S_2, \mathbf{g}) \to A'$. Since θ and π are isomorphisms so is ψ; Lemma 11.3.14 then implies that $Y(\psi) = Y(\theta)Y(\pi)$ is also an isomorphism. Therefore, $Y(\theta) = Y(\psi)(Y(\pi))^{-1}$ is an isomorphism and so $Y_{A,A'}$ is a well-defined function.

Now we prove that $Y_{A,A'}$ is a bijection. It follows from Definitions 11.3.4 and 11.3.13 that $YX = I_{\mathbf{ACP}}$. By Theorem 11.3.7, $X_{(P_1,S_2,\mathbf{g}),(Q_1,S_2',\mathbf{h})}$ is bijective; this implies that $Y_{X(P_1,S_2,\mathbf{g}),X(Q_1,S_2',\mathbf{h})}$ is also bijective since the identity is bijective. Now we just need to show that $Y_{A,A'}$ is bijective. For each isomorphism $\theta : A \to A'$, there exists an isomorphism ρ :

$X(P_1, S_2, \mathbf{g}) \to X(Q_1, S_2', \mathbf{h})$ such that $\theta = \pi' \rho \pi^{-1}$. It follows that $Y(\theta) = Y(\pi')Y(\rho)Y(\pi^{-1})$ from which we see that $Y_{A,A'}$ is indeed bijective.

We already showed that $Y(A)$ can be computed in polynomial time in Lemma 11.3.14 and it follows easily from Definition 11.3.13 that $Y(\theta)$ can be computed in polynomial time. □

While Theorem 11.3.15 is enough to obtain our canonization results, we point out that X and Y form a category equivalence when viewed as functors. Moreover, the results of this section can be derived from this more general fact.

To construct our canonical form for augmented α-composition pairs, we convert our augmented α-composition pairs to graphs of degree at most $\alpha + O(1)$ by applying X. Then we compute the canonical form of the resulting graph using Theorem 8.4.7 and convert it back into an augmented α-composition pair by applying Y. We use $\mathrm{Can}_{\mathbf{Graph}}$ to denote the map from graphs to their canonical forms from Theorem 8.4.7.

Theorem 11.3.16. *$Y \circ \mathrm{Can}_{\mathbf{Graph}} \circ X$ is a canonical form for augmented α-composition pairs. Moreover, for any α-composition pair $(P_1, S_2, \mathbf{g})$, we can compute $(Y \circ \mathrm{Can}_{\mathbf{Graph}} \circ X)(P_1, S_2, \mathbf{g})$ in $n^{O(\alpha \log \alpha)}$ time.*

Proof. Consider two α-composition pairs $(P_1, S_2, \mathbf{g})$ and $(Q_1, S_2', \mathbf{h})$ for the solvable groups G and H. By Corollary 11.3.8, $(P_1, S_2, \mathbf{g}) \cong (Q_1, S_2', \mathbf{h})$ if and only if

$$X(P_1, S_2, \mathbf{g}) \cong X(Q_1, S_2', \mathbf{h}).$$

Thus, $(P_1, S_2, \mathbf{g}) \cong (Q_1, S_2', \mathbf{h})$ if and only if

$$\mathrm{Can}_{\mathbf{Graph}}(X(P_1, S_2, \mathbf{g})) = \mathrm{Can}_{\mathbf{Graph}}(X(Q_1, S_2', \mathbf{h}))$$

Now, clearly, if $(P_1, S_2, \mathbf{g}) \cong (Q_1, S_2', \mathbf{h})$,

$$Y(\mathrm{Can}_{\mathbf{Graph}}(X(P_1, S_2, \mathbf{g}))) = Y(\mathrm{Can}_{\mathbf{Graph}}(X(Q_1, S_2', \mathbf{h})))$$

On the other hand, if $(P_1, S_2, \mathbf{g}) \ncong (Q_1, S_2', \mathbf{h})$, then

$$\begin{aligned}\mathrm{Can}_{\mathbf{Graph}}(X(P_1, S_2, \mathbf{g})) &\ncong \mathrm{Can}_{\mathbf{Graph}}(X(Q_1, S_2', \mathbf{h}))\\ Y(\mathrm{Can}_{\mathbf{Graph}}(X(P_1, S_2, \mathbf{g}))) &\ncong Y(\mathrm{Can}_{\mathbf{Graph}}(X(Q_1, S_2', \mathbf{h})))\\ Y(\mathrm{Can}_{\mathbf{Graph}}(X(P_1, S_2, \mathbf{g}))) &\neq Y(\mathrm{Can}_{\mathbf{Graph}}(X(Q_1, S_2', \mathbf{h}))).\end{aligned}$$

Thus, $Y \circ \mathrm{Can}_{\mathbf{Graph}} \circ X$ is a complete invariant. Also, $X(P_1, S_2, \mathbf{g}) \cong \mathrm{Can}_{\mathbf{Graph}}(X(P_1, S_2, \mathbf{g}))$ so since $YX = I_{\mathbf{ACP}}$, we have $(P_1, S_2, \mathbf{g}) \cong Y(\mathrm{Can}_{\mathbf{Graph}}(X(P_1, S_2, \mathbf{g})))$ by Theorem 11.3.15. Thus, $Y \circ \mathrm{Can}_{\mathbf{Graph}} \circ X$ is a canonical form.

Lastly, we show that $Y(\mathrm{Can}_{\mathbf{Graph}}(X(P_1, S_2, \mathbf{g})))$ can be computed in $n^{O(\alpha \log \alpha)}$ time. By Theorem 11.3.7, we can compute $X(P_1, S_2, \mathbf{g})$ in polynomial time. By Lemma 11.3.10 and Theorem 8.4.7, it takes $n^{O(\alpha \log \alpha)}$ time to compute $\mathrm{Can}_{\mathbf{Graph}}(X(P_1, S_2, \mathbf{g}))$. Finally, by Theorem 11.3.15, we can compute $Y(\mathrm{Can}_{\mathbf{Graph}}(X(P_1, S_2, \mathbf{g})))$ in polynomial time from $\mathrm{Can}_{\mathbf{Graph}}(X(P_1, S_2, \mathbf{g}))$. □

The following corollary is now easily proved. We include the bound on the space required since it will be relevant in Chapter 12.

Corollary 11.3.17. *Canonization of α-composition pairs can be done deterministically in $n^{\log_\alpha n + O(\alpha \log \alpha)}$ time using $n^{\log_\alpha n + O(1)}$ space.*

Proof. Let (P_1, S_2) be an α-composition pair. Note that the algorithm of Theorem 8.4.7 can be performed in polynomial space. The result then follows by enumerating the $n^{\log_\alpha n}$ ways of choosing the fixed generators $\mathbf{g}$. For each such choice, we apply Theorem 11.3.16 to compute the canonical form of the augmented α-decomposition pair $(P_1, S_2, \mathbf{g})$ in $n^{O(\alpha \log \alpha)}$ time. We then chose the lexicographically least of these as the canonical form of (P_1, S_2). □

11.4 Algorithms for solvable-group isomorphism and canonization

Armed with the results of Sections 11.2 and 11.3, it is easy to prove Theorem 11.1.1 as claimed at the beginning of this chapter.

Theorem 11.1.1. *Solvable-group isomorphism is decidable in* $n^{(1/2)\log_p n + O(\log n / \log\log n)}$ *deterministic time.*

Proof. Let α be a parameter to be chosen later. By combining Lemma 11.2.7 and Corollary 11.3.12, we obtain an $n^{(1/2)\log_p n + \log_\alpha n + O(\alpha \log \alpha)}$ time algorithm for solvable-group isomorphism. The optimal choice for α is $\log n/(\log\log n)^2$. The complexity is then $n^{(1/2)\log_p n + O(\log n/\log\log n)}$ as claimed.

□

Our algorithm for solvable-group canonization follows by a similar argument.

Theorem 11.4.1. *Solvable-group canonization is in* $n^{(1/2)\log_p n + O(\log n/\log\log n)}$ *deterministic time.*

Proof. Let α be a parameter to be chosen later. By combining Corollaries 11.2.12 and 11.3.17, we obtain an $n^{(1/2)\log_p n + \log_\alpha n + O(\alpha \log \alpha))}$ time algorithm for solvable-group canonization. The optimal choice for α is $\alpha = \log n/(\log\log n)^2$. The complexity is again $n^{(1/2)\log_p n + O(\log n/\log\log n)}$ as claimed.

□

Chapter 12
BIDIRECTIONAL COLLISION DETECTION

12.1 Introduction

In the last few chapters, we focused on the group isomorphism problem. In this chapter, we take a more general view and study generic *isomorphism problems*. In such a problem, we are given two algebraic or combinatorial objects and must decide if they have the same structure. This chapter introduces a general technique for obtaining deterministic speedups for many isomorphism problems. We apply the resulting lemmas to improve the best algorithms known for a number of isomorphism-testing problems including several classes of groups.

In bidirectional collision detection, we consider structures that restrict the isomorphisms between objects in some class. For example, given any ordered generating sets for two groups, there is at most one isomorphism that maps the first ordered generating set to the second. The idea behind bidirectional collision detection applies to objects with isomorphism-restricting structures that can be split in half. To test isomorphism between two objects A and B, we then choose the first half of the structure for A in all possible ways and choose the second half arbitrarily; the structure for B is constructed by choosing the first half arbitrarily and the second half in all possible ways. If A and B are isomorphic, the arbitrary choice made for the first half of the structure for B will correspond to some choice for the first half of the structure for A and the arbitrary choice for the second half of the structure for A will correspond to some choice for the second half of the structure for B. Since we only have to enumerate roughly the square root of the number of isomorphism-restricting structures, bidirectional collision detection yields a square-root speedup over the naive brute-force algorithm for many isomorphism problems.

We apply bidirectional collision detection to several theoretical algorithms for isomorphism testing. First, we utilize bidirectional collision detection to obtain a faster algorithm for the group isomorphism problem.

The purpose of this chapter is to introduce *bidirectional collision detection* — a new technique for obtaining deterministic speedups by relating isomorphism testing in many classes of objects to collision detection. Since bidirectional collision detection in particular applies to the class of collision problems that arise in group isomorphism, we obtain a deterministic square-root speedup over the best previous algorithm for general groups.

Theorem 12.1.1. *General group isomorphism is decidable in $n^{(1/2)\log_p n+O(1)}$ deterministic time where p is the smallest prime dividing the order of the group.*

In Chapter 10, we showed a deterministic square-root speedup over the generator-enumeration algorithm for the class of p-groups. We generalized this result to the hard special case of solvable groups in Chapter 11. This chapter uses bidirectional collision detection to obtain the improved bounds of Chapters 10 and 11 for general groups. Since the techniques used in Chapters 10 and 11 are independent of bidirectional collision detection, we can combine these ideas to obtain a deterministic fourth-root speedup over the generator-enumeration algorithm for the class of solvable groups.

Theorem 12.1.2. *Solvable-group isomorphism is decidable in $n^{(1/4)\log_p n+O(\log n/\log\log n)}$ deterministic time where p is the smallest prime dividing the order of the group.*

While randomized analogues of our algorithms also exist [108], they do not improve on the time and space requirements of our deterministic algorithms.

Bidirectional collision detection can also be applied to the ring isomorphism problem to obtain a square-root speedup over the natural analogue of the generator-enumeration algorithm for rings.

In the case of graph isomorphism, bidirectional collision detection can be used to reduce the constant in the exponent of the best general algorithm previously known [16] by a factor of $1/\sqrt{2}$. This is achieved by using bidirectional collision detection to obtain an improved version of Zemlyachenko's lemma which is then combined with the $n^{O(d/\log d)}$ algorithm [16, 77] for computing canonical forms of graphs of degree at most d.

While most algorithms for isomorphism problems can be implemented in polynomial space, our algorithms require space roughly equal to their runtimes. By breaking the underlying bidirectional collision problem up into blocks, we show that the generator-enumeration algorithm and our algorithm are extreme points of our time-space tradeoff $TS = n^{\log_p n + O(1)}$ for general group isomorphism. For solvable groups, we get a time-space tradeoff of $TS = n^{(1/2)\log_p n + O(\log n/\log\log n)}$. Using a modification of the quantum algorithm for collision detection [26], we obtain quantum time-space tradeoffs of $T\sqrt{S} = n^{(1/2)\log_p n + O(1)}$ for general groups and $T\sqrt{S} = n^{(1/4)\log_p n + O(\log n/\log\log n)}$ for solvable groups. Analogous time-space tradeoffs exist in general for the classes of objects that we consider.

Laci Babai and Eugene Luks (personal communication) have since combined bidirectional collision detection with other ideas to obtain an $n^{(1/4)\log_p n + O(\log\log n)}$ algorithm for general groups. This extends our result for solvable groups to the general case.

We start by introducing the framework for bidirectional collision-detection Section 12.2. The rest of the chapter applies these lemmas to various isomorphism problems to obtain speedups. In Section 12.3, we combine bidirectional collision detection with the generator-enumeration algorithm to obtain our deterministic square-root speedup for general group isomorphism. We show a deterministic fourth-root speedup over generator-enumeration for solvable-group isomorphism in Section 12.4. In Section 12.5, we discuss a deterministic square-root speedup for the ring isomorphism problem. In Section 12.6, we show a speedup for worst-case graph isomorphism. We conclude with the current state of the art and open problems in Section 12.7.

12.2 Bidirectional collision detection lemmas

In this section, we prove a general lemma that yields a deterministic speedup for isomorphism testing in any class with objects whose structure "splits" in a certain way. Later in this chapter, we shall see that our lemma is sufficiently powerful to yield speedups for several well-known isomorphism problems. First, we introduce the idea behind bidirectional collision detection by applying it to a toy problem involving binary functions.

12.2.1 Bidirectional collision detection and the NPN classification of binary functions

Consider the class of all binary functions on n variables. Under the negation-permutation-negation (NPN) classification of binary functions (cf. [118, 49]), two functions are equivalent if they can be made equal by negating some subset of the input variables, permuting the input variables and possibly negating the output variable. A natural problem is then to test if two binary functions given as truth tables are NPN-equivalent. By considering all possible combinations of negations and permutations, we obtain a deterministic $O(4^n n!)$ time algorithm for testing NPN-equivalence. Luks [75] reduced this to $2^{O(n)}$ time using his algorithm for hypergraph isomorphism.

In order to illustrate bidirectional collision detection, we shall consider a simpler variant of the NPN-equivalence problem. Let us say that two binary functions are negation-equivalent if they can be made equal by negating some subset of their inputs. Consider the problem of testing if two binary functions given as truth tables are negation-equivalent. An obvious way to test if two binary functions f and g of n variables are negation equivalent is to negate the inputs of f according to the 2^n possible subsets. Negating each subset yields a new function f' and we can test if $f' = g$ in $O(2^n)$ time. The functions f and g are negation-equivalent if and only if $f' = g$ where f' is the function that arises from negating some subset of the variables. Therefore, we can test if f and g are negation equivalent deterministically in $O(4^n)$ time using a naive algorithm.

We can do better using bidirectional collision detection. Consider two binary functions f and g of n variables. Let A be the set of all binary functions that can be obtained from f by negating a subset of the first $n/2$ variables and let B be the set of binary functions that can be obtained from g by negating a subset of the last $n/2$ variables. Then f and g are negation-equivalent if and only if A and B contain a common element. We can test if this is the case by sorting the sets A and B lexicographically and merging the results. Since $|A| =$

$|B| = 2^{n/2}$, this can be done deterministically in $O(n2^{(3/2)n})$ time while the naive algorithm requires $O(4^n)$ operations. Thus, bidirectional collision detection uses quadratically fewer comparisons. Since the sizes of the objects we consider are typically small, square-root speedups typically apply to the time complexity as well as to the number of high-level operations required on the objects involved.

12.2.2 General bidirectional collision detection lemmas

Let $\mathcal{C}$ be the class in which we wish to test isomorphism. We associate a tree $\mathcal{T}(A)$ to each object $A \in \mathcal{C}$. Each path in $\mathcal{T}(A)$ from the root to a leaf represents a series of choices that capture the structure of A. For example, in the class of groups, each node on such a path will correspond to a choice of a generator so that paths from the root correspond to generating sets. We then define a "partial canonical form" function $\mathrm{Can}_{\mathcal{C}}$ that maps each pair consisting of an object $A \in \mathcal{C}$ and a leaf of $\mathcal{T}(A)$ to an object in $\mathcal{C}$ that is isomorphic to A. For each isomorphism $\phi : A \to B$ where $B \in \mathcal{C}$, we require that there exists an isomorphism $\mathcal{T}(\phi) : \mathcal{T}(A) \to \mathcal{T}(B)$ such that the corresponding leaves in $\mathcal{T}(A)$ and $\mathcal{T}(B)$ are mapped to the same object by $\mathrm{Can}_{\mathcal{C}}$. Thus, we can think of $\mathrm{Can}_{\mathcal{C}}$ as computing a canonical form of an object $A \in \mathcal{C}$ with respect to the choices that correspond to a leaf of $\mathcal{T}(A)$. Let **Tree** be the class of finite rooted trees and let L be a function that maps each tree to its set of leafs. We formalize these ideas with following definition.

Definition 12.2.1. *The triple* $(\mathcal{C}, \mathcal{T}, \mathrm{Can}_{\mathcal{C}})$ *is a* collision system *if* $\mathcal{C}$ *is a class of objects,* $\mathcal{T}$ *and* $\mathrm{Can}_{\mathcal{C}}$ *are functions such that*

(a) for each $A \in \mathcal{C}$, $\mathcal{T}(A)$ *is a rooted tree,*

(b) for each isomorphism $\phi : A \to B$ *with* $A, B \in \mathcal{C}$, $\mathcal{T}(\phi) : \mathcal{T}(A) \to \mathcal{T}(B)$ *is a rooted tree isomorphism,*

(c) for each $A \in \mathcal{C}$ *and each leaf* x *in* $\mathcal{T}(A)$, $\mathrm{Can}_{\mathcal{C}}(A, x)$ *is an object in* $\mathcal{C}$ *isomorphic to* A *and*

(d) for all $A, B \in \mathcal{C}$ *each leaf* x *in* $\mathcal{T}(A)$, *and every isomorphism* $\phi : A \to B$, $\mathrm{Can}_{\mathcal{C}}(A, x) =$

$\mathrm{Can}_{\mathcal{C}}(B, \mathcal{T}(\phi)(x))$

The idea behind bidirectional collision detection in its most general form is to compute subtrees $\mathcal{T}_1(A)$ of $\mathcal{T}(A)$ and $\mathcal{T}_2(B)$ such that $A \cong B$ if and only if there exist leaves x in $\mathcal{T}_1(A)$ and y in $\mathcal{T}_2(B)$ such that $\mathrm{Can}_{\mathcal{C}}(A, x) = \mathrm{Can}_{\mathcal{C}}(B, y)$. We formalize this with the following definition.

Definition 12.2.2. *Let $(\mathcal{C}, \mathcal{T}, \mathrm{Can}_{\mathcal{C}})$ be a collision system and let $\mathcal{T}_1 : \mathcal{C} \to \mathbf{Tree}$ and $\mathcal{T}_2 : \mathcal{C} \to \mathbf{Tree}$ be functions such that for each $A \in \mathcal{C}$, the leaves of $\mathcal{T}_1(A)$ and $\mathcal{T}_2(A)$ are subsets of the leaves of $\mathcal{T}(A)$. Then the pair $(\mathcal{T}_1, \mathcal{T}_2)$ is a* strategy *for $(\mathcal{C}, \mathcal{T}, \mathrm{Can}_{\mathcal{C}})$ if for each $A, B \in \mathcal{C}$ such that $\phi : A \to B$ is an isomorphism, there exists a leaf x of $\mathcal{T}_1(A)$ such that $\phi(x)$ is a leaf of $\mathcal{T}_2(B)$.*

This yields the most general form of our bidirectional collision detection lemma. The proof follows very easily from the definition. We use $L(U)$ to denote the leaves of a tree U. We denote by $|A|$ is the size of the description of A.

Lemma 12.2.3. *Let $(\mathcal{T}_1, \mathcal{T}_2)$ be a strategy for a collision system $(\mathcal{C}, \mathcal{T}, \mathrm{Can}_{\mathcal{C}})$ such that for each $A \in \mathcal{C}$, $t(m)$ upper bounds the time required to compute $\mathcal{T}_1(A)$ and $\mathcal{T}_2(A)$ and $\ell(m)$ upper bounds the size of the description of each node in $\mathcal{T}(A)$ where $m = |A| = |B|$. Define $k = |L(\mathcal{T}_1(A))| + |L(\mathcal{T}_2(B))|$. Then for $A, B \in \mathcal{C}$, we can Turing-reduce testing if $A \cong B$ to evaluating k calls to $\mathrm{Can}_{\mathcal{C}}$ of the forms $\mathrm{Can}_{\mathcal{C}}(A, \cdot)$ and $\mathrm{Can}(B, \cdot)$ deterministically in $O(t(m) + k\log(k) + \ell(m))$ time.*

Proof. Compute the trees $\mathcal{T}_1(A)$ and $\mathcal{T}_2(B)$ and collect the objects $\mathrm{Can}_{\mathcal{C}}(A, x)$ and $\mathrm{Can}_{\mathcal{C}}(B, y)$ for all leaves x of $\mathcal{T}_1(A)$ and y of $\mathcal{T}_2(B)$ into two lists $\mathbf{A}$ and $\mathbf{B}$. Then determine if the lists have a common entry by sorting and merging them. $\square$

Usually, this lemma is too general to be especially useful. However, for most of the classes of objects we consider, the tree $\mathcal{T}(A)$ satisfies bounds on the degrees of its nodes that allow us to prove a more specialized and useful lemma. First, we need another definition. For a tree U, we let $h(U)$ denote its height.

Definition 12.2.4 (Deterministic computational assumptions). *A collision system* $(\mathcal{C}, \mathcal{T}, \mathrm{Can}_{\mathcal{C}})$ *is* bounded by b *if for all* $A, B \in \mathcal{C}$

(a) each $b(A) = (b_0(A), \ldots, b_{h(\mathcal{T}(A))-1}(A))$ *where each* $b_i : \mathcal{C} \to \mathbb{N}$*,*

(b) each $b_i(A) \geq 2$*,*

(c) the number of children of any node at each distance i *from the root is at most* $b_i(A)$*,*

(d) each $b_i(A)$ *can be computed in* $\mathsf{poly}(m)$ *time where* $m = |A| = |B|$ *and*

(e) if $A \cong B$*, then each* $b_i(A) = b_i(B)$*.*

We will also need the ability to compute the tree $\mathcal{T}(A)$ incrementally.

Definition 12.2.5. *A collision system* $(\mathcal{C}, \mathcal{T}, \mathrm{Can}_{\mathcal{C}})$ *is* oracular *if*

(a) given $A \in \mathcal{C}$*, we can compute the label of the root node of* $\mathcal{T}(A)$ *in* $\mathsf{poly}(m)$ *time and*

(b) given the label of a node in $\mathcal{T}(A)$ *for some* $A \in \mathcal{C}$*, we can compute the set of labels of its children in* $\mathsf{poly}(m)$ *time*

We define $b_{\max}(A) = \max_{i=0}^{h(\mathcal{T}(A))-1} b_i(A)$. Define each $N_{j,k}(A) = \prod_{i=j}^{k} b_i(A)$ and define $N(A) = N_{0,h(\mathcal{T}(A))-1}$. Our additional assumptions allow us to prove the following time-space tradeoff.

Lemma 12.2.6. *Let* $(\mathcal{C}, \mathcal{T}, \mathrm{Can}_{\mathcal{C}})$ *be an oracular collision system bounded by* b *and let* $A, B \in \mathcal{C}$*. Then using space* $\mathsf{poly}(m) \leq S \leq \sqrt{N(A)/b_{\max}(A)} \cdot \mathsf{poly}(m)$*, we can Turing-reduce testing if* $A \cong B$ *to evaluating* $O(\sqrt{N(A)/b_{\max}(A)})$ *calls to* $\mathrm{Can}_{\mathcal{C}}$ *of the forms* $\mathrm{Can}_{\mathcal{C}}(A, \cdot)$ *and* $\mathrm{Can}(B, \cdot)$ *deterministically in* $T = \frac{N(A)\log(S)\cdot\mathsf{poly}(m)}{S}$ *time where* $m = |A| = |B|$*. In particular, if we set* $S = \sqrt{N(A)/b_{\max}(A)} \cdot \mathsf{poly}(m)$*, the reduction takes time* $T = \sqrt{N(A)}\log(N(A)) \cdot \mathsf{poly}(m)$*.*

Proof. Note that each $N_{0,j}(A)$ is an upper bound on the number of nodes at distance j from the root and since each $b_i(A) \geq 2$ by Definition 12.2.4, $O(N_{0,j}(A))$ is an upper bound on the number of nodes within distance j of the root. We start by computing a $k(A)$ such that $N_{0,k(A)}(A)$ and $N_{k(A)+1,h(\mathcal{T}(A))-1}(A)$ are both within a factor of $\sqrt{b_{\max}(A)}$ of $\sqrt{N(A)}$. To do this, we let j be the largest natural number such that $N_{0,j}(A) \leq \sqrt{N(A)/b_{\max}(A)}$ and set $k(A) = j + 1$.

To test if $A \cong B$, we first check if $h(\mathcal{T}(A)) = h(\mathcal{T}(B))$ and each $b_i(A) = b_i(B)$. If not, then $A \ncong B$. Otherwise, each $N_{i,j}(A) = N_{i,j}(B)$ so $k(A) = k(B)$ and we define $\mathcal{T}_1(A)$ to be the subtree of $\mathcal{T}(A)$ that consists of all nodes within a distance of $k(A)$ of the root plus arbitrary paths from each node at distance $k(A)$ to leaves of $\mathcal{T}(A)$. We let $\mathcal{T}_2(B)$ be a subtree of $\mathcal{T}(B)$ that consists of an arbitrary path of length $k(B)$ from the root of $\mathcal{T}(B)$ to a node v and the subtree of $\mathcal{T}(B)$ rooted at v.

We claim that there are leaves x in $\mathcal{T}_1(A)$ and y in $\mathcal{T}_2(B)$ such that $\mathrm{Can}_{\mathcal{C}}(A, x) = \mathrm{Can}_{\mathcal{C}}(B, y)$ if and only if $A \cong B$. If $A \ncong B$, then for any leaves x of $\mathcal{T}_1(A)$ and y of $\mathcal{T}_2(B)$, we have $\mathrm{Can}_{\mathcal{C}}(A, x) \cong A$ and $\mathrm{Can}_{\mathcal{C}}(B, y) \cong B$ by Definition 12.2.1 so $\mathrm{Can}_{\mathcal{C}}(A, x) \neq \mathrm{Can}_{\mathcal{C}}(B, y)$. Otherwise, if $\phi : A \to B$ is an isomorphism, then $\mathcal{T}(\phi) : \mathcal{T}(A) \to \mathcal{T}(B)$ is also an isomorphism by Definition 12.2.1. Therefore $u = (\mathcal{T}(\phi))^{-1}(v)$ is at a distance of $k(A)$ from the root of $\mathcal{T}(A)$ so u is in $\mathcal{T}_1(A)$. Now, there exists a leaf x in $\mathcal{T}_1(A)$ that is in the subtree of $\mathcal{T}(A)$ rooted at u. Since $\mathcal{T}_2(B)$ contains the subtree of $\mathcal{T}(B)$ rooted at v, it follows that $y = \mathcal{T}(\phi)(x)$ is a leaf of $\mathcal{T}_2(B)$. Thus, $\mathrm{Can}_{\mathcal{C}}(A, x) = \mathrm{Can}_{\mathcal{C}}(B, y)$ by Definition 12.2.1.

Thus, we can decide isomorphism by determining if there exist leaves x in $\mathcal{T}_1(A)$ and y in $\mathcal{T}_2(B)$ such that $\mathrm{Can}_{\mathcal{C}}(A, x) = \mathrm{Can}_{\mathcal{C}}(B, y)$. We note that there are surjections $\iota_1(A) : [b_0] \times \cdots \times [b_{k(A)}] \to L(\mathcal{T}_1(A))$ and $\iota_2(B) : [b_{N(B)+1}] \times \cdots \times [b_{h(\mathcal{T}(B))-1}] \to L(\mathcal{T}_2(B))$ that can be evaluated in $\mathsf{poly}(m)$ time. In order to test if $A \cong B$ using space $O(S)$, we break up the sets $[b_0] \times \cdots \times [b_{k(A)}]$ and $[b_{k(B)+1}] \times \cdots \times [b_{h(\mathcal{T}_2)-1}]$ into chunks of size $\Delta_1 = S/s$ and $\Delta_2 = S/s$ where $s = \mathsf{poly}(m)$ upper bounds the space required for nodes in $\mathcal{T}(A)$ and $\mathcal{T}(B)$. For each pair of chunks U of $[b_0] \times \cdots \times [b_{k(A)}]$ and W of $[b_{k(B)+1}] \times \cdots \times [b_{h(\mathcal{T}_2)-1}]$, we test if there is an $u \in U$ and a $w \in W$ such that $\mathrm{Can}_{\mathcal{C}}(A, \iota_1(A)(u) = \mathrm{Can}_{\mathcal{C}}(B, \iota_2(B)(w))$. This can be accomplished in $O(S \log(S) \cdot \mathsf{poly}(m)$ by computing $\mathrm{Can}_{\mathcal{C}}(A, \iota_1(A)(u)$ and $\mathrm{Can}_{\mathcal{C}}(B, \iota_2(B)(w))$ for all $u \in U$ and $w \in W$ and sorting and merging the resulting lists. Since the number of pairs of chunks is at most

$$\frac{N_{0,k(A)}N_{k(B)+1,h(\mathcal{T}_2(B))-1}}{\Delta_1\Delta_2} \leq \frac{N(A)\cdot\mathsf{poly}(m)}{S^2}$$

the overall time complexity is $T = \frac{N(A)\log(S)\cdot\mathsf{poly}(m)}{S}$. $\square$

We remark that, in the above proof, we have constructed the tree $\mathcal{T}_1(A)$ by adding all children of each node in $\mathcal{T}(A)$ at a distance of less than $k(A)$ from the root and then following an arbitrary path from each node at a distance of $k(A)$ to a leaf of $\mathcal{T}(A)$. Similarly, $\mathcal{T}_2(B)$ was constructed by choosing an arbitrary child of each node at a distance of less than $k(B)$ from the root and selecting all children of the nodes at distances at least $k(B)$ from the root. This is a special case of a more general strategy which could be more efficient for some problems. Let W_1 and W_2 partition $\{0,\ldots,h(\mathcal{T}(A))-1\}$. To construct $\mathcal{T}_1(A)$, we select all children when the distance from the root of $\mathcal{T}(A)$ is contained in W_1 and select an arbitrary child when it is in W_2. The tree $\mathcal{T}_2(B)$, is constructed by selecting all children when the distance from the root of $\mathcal{T}(B)$ is contained in W_2 and selecting an arbitrary child when it is in W_1. How efficient this strategy is depends on the problem under consideration. For the problems discussed in this chapter, there is no advantage but it is possible that it could be useful in other settings.

We can also prove a quantum time-space tradeoff. However, this requires different computational assumptions. Randomized algorithms also exist; however, they are no better than the deterministic algorithms that result from Lemma 12.2.6.

Definition 12.2.7 (Quantum computational assumptions)**.** *A pair (M,ι) is an* index *for a collision system $(\mathcal{C},\mathcal{T},\mathrm{Can}_{\mathcal{C}})$ if $M:\mathcal{C}\to\mathbb{N}$ is a function and for each $A\in\mathcal{C}$, $\iota(A):[M(A)]\to U(A)$ is a bijection such that*

(a) $L(\mathcal{T}(A))\subseteq U(A)$,

(b) $M(A)$ can be computed in $\mathsf{poly}(m)$ time and

(c) $\iota(A)$ can be evaluated in $\mathsf{poly}(m)$ time.

We now show that a quantum time-space tradeoff exists for every indexable collision system. Our proof uses a simple modification of the algorithm for quantum collision detection [26].

Lemma 12.2.8. *Let (M, ι) be an index for an oracular collision system $(\mathcal{C}, \mathcal{T}, \mathrm{Can})$ and let $A, B \in \mathcal{C}$. Then using space $\mathsf{poly}(m) \leq S \leq \sqrt[3]{|L(\mathcal{T}(A))|} \cdot \mathsf{poly}(m)$ where $m = |A| = |B|$, we can Turing-reduce testing if $A \cong B$ to evaluating calls of $\mathrm{Can}_{\mathcal{C}}$ of the forms $\mathrm{Can}_{\mathcal{C}}(A, \cdot)$ and $\mathrm{Can}_{\mathcal{C}}(B, \cdot)$ quantumly in $T = \sqrt{|M(A)|/S} \cdot \mathsf{poly}(m)$ time.*

Proof. Let $\mathbf{A}$ be a list obtained by computing $\mathrm{Can}_{\mathcal{C}}(A, x)$ for $S/\mathsf{poly}(m)$ leafs in $\mathcal{T}(A)$. This can be done in $O(Sh(\mathcal{T}(A)) \cdot \mathsf{poly}(m))$ time by traversing the tree using depth-first search until the required number of leaves are found. Given $k \in [M(B)]$, we can test if $\mathrm{Can}_{\mathcal{C}}(A, x) = \mathrm{Can}_{\mathcal{C}}(B, \iota(B)(k))$ for some $x \in \mathbf{A}$ in $O(\log(S) \cdot \mathsf{poly}(m)$ time. If $A \cong B$, then there are at least $S/\mathsf{poly}(m)$ numbers m such that $\mathrm{Can}_{\mathcal{C}}(A, x) = \mathrm{Can}_{\mathcal{C}}(B, \iota(B)(m))$ for some $x \in \mathbf{A}$. It follows that we can decide if such a collision exists and therefore if $A \cong B$ in $T = (\sqrt{M(B)/S} + Sh(\mathcal{T}(A))) \cdot \mathsf{poly}(m)$ time using Grover's algorithm [52, 25]. □

In the problems we apply this lemma to, $M(A) = |L(\mathcal{T}(A))| \cdot \mathsf{poly}(m)$, $h(\mathcal{T}(A)) = O(\log M(A))$, and $\mathrm{Can}_{\mathcal{C}}$ can be evaluated in $\mathsf{poly}(m)$ time so setting $S = \sqrt[3]{|L(\mathcal{T}(A))|}$, yields a quantum algorithm for testing isomorphism that runs in time $T = \sqrt[3]{|L(\mathcal{T}(A))|} \cdot \mathsf{poly}(m)$.

12.3 General group isomorphism

We now prove a generalization of Theorem 12.1.1 by giving a deterministic time-space tradeoff for group isomorphism. This is accomplished by applying our bidirectional collision detection lemmas to a tree of ordered generating sets of subgroups of G. We call an ordered generating set $\mathbf{g} = (g_1, \ldots, g_k)$ for a subgroup of G *non-redundant* if $g_j \notin \langle g_i \mid 1 \leq i < j \rangle$ for each $j \leq k$. First, we define the tree $\mathcal{T}(G)$ for each group G and the tree isomorphism $\mathcal{T}(\phi)$ on each group isomorphism ϕ.

Definition 12.3.1. *For each group G, the nodes of the tree $\mathcal{T}(G)$ are the non-redundant ordered generating sets of subgroups of G. The root is the empty ordered generating set. Each* *ordered generating set $(g_1, \ldots, g_k)$ of a proper subgroup of G has an edge to $(g_1, \ldots, g_k, g_{k+1})$ for each $g_{k+1} \in G \setminus \{g_j \mid 1 \leq j \leq k\}$.*

If G and H are groups and $\phi : G \to H$ is an isomorphism, we define $\mathcal{T}(\phi) : \mathcal{T}(G) \to \mathcal{T}(H)$ by $\mathcal{T}(\phi)(g_1, \ldots, g_k) = (\phi(g_1), \ldots, \phi(g_k))$ for each $(g_1, \ldots, g_k) \in V(\mathcal{T}(G))$.

It is clear that $\mathcal{T} : \mathbf{Grp} \to \mathbf{Tree}$ satisfies properties (a) and (b) of Definition 12.2.1. The next step is to define $\mathrm{Can}_{\mathbf{Grp}}$. For this, we require the following lemma. It is an immediate consequence of Lemma 10.4.2.

Lemma 12.3.2. *There is a function $\mathrm{Can}_{\mathbf{Grp}}$ such that*

(a) for a group G and an ordered generating set $\mathbf{g}$ for G, $\mathrm{Can}_{\mathbf{Grp}}(G, \mathbf{g})$ is a multiplication table for a group isomorphic to G and

(b) if $\mathbf{g}$ and $\mathbf{h}$ are ordered generating sets for the groups G and H and $\phi : G \to H$ is an isomorphism such that $\phi(\mathbf{g}) = \mathbf{h}$, then $\mathrm{Can}_{\mathbf{Grp}}(G, \mathbf{g}) = \mathrm{Can}_{\mathbf{Grp}}(H, \mathbf{h})$.

While this lemma may seem powerful, its proof is actually fairly simple. Given an ordered generating set $\mathbf{g}$ for a group G, we define an isomorphism-invariant total order $\prec_{\mathbf{g}}$ on G as follows. For each $x \in G$, we let $w_{\mathbf{g}}(x)$ be the lexicographically first word over $\mathbf{g}$ whose product is equal to $\mathbf{g}$. To compare two elements $x, y \in G$, we can then compare the words $w_{\mathbf{g}}(x)$ and $w_{\mathbf{h}}(y)$ lexicographically. We then define $\mathrm{Can}_{\mathbf{Grp}}(G, \mathbf{g})$ to be the multiplication table of G with the elements permuted according to their positions in the ordering $\prec_{\mathbf{g}}$.

Theorem 12.3.3. *Let G and H be groups let p be the smallest prime divisor of the order of the group.*

(a) Using space $\mathsf{poly}(n) \leq S \leq n^{(1/2)\log_p n + O(1)}$, we can decide if $G \cong H$ deterministically in $T = n^{\log_p n + O(1)}/S$ time. In particular, setting $S = n^{(1/2)\log_p n + O(1)}$ yields a deterministic $n^{(1/2)\log_p n + O(1)}$ time algorithm.

(b) Using space $\mathsf{poly}(n) \leq S \leq |L(\mathcal{T}(G))| \leq n^{(1/3)\log_p n + O(1)}$, we can decide if $G \cong H$ quantumly in $T = n^{(1/2)\log_p n + O(1)}/\sqrt{S}$ time. In particular, setting $S = n^{(1/3)\log_p n + O(1)}$ yields an $n^{(1/3)\log_p n + O(1)}$ time quantum algorithm.

Proof. By Lemma 12.3.2, $(\mathbf{Grp}, \mathcal{T}, \mathrm{Can}_{\mathbf{Grp}})$ is an oracular collision system. To prove part (a), we define each $b_i(K) = |K|$ for each group K and observe that b is a bound for $(\mathbf{Grp}, \mathcal{T}, \mathrm{Can}_{\mathbf{Grp}})$. Applying Lemma 12.2.6, we can decide if $G \cong H$ determinstically using space $\mathsf{poly}(n) \leq S \leq n^{(1/2)\log_p n + O(1)}$ in $T = n^{\log_p n + O(1)}/S$ time.

Now we prove part (b). For each group K, let $p(K)$ be the smallest prime divisor of $|K|$ and define $U(K) = [|K|]^{\log_{p(K)}|K|}$; let $M(K) = |K|^{\log_{p(K)}|K|}$ and let $\iota(K) : M(K) \to U(K)$ be an arbitrary bijection that can be evaluated in $\mathsf{poly}(|K|)$. Then (M, ι) is an index for $(\mathbf{Grp}, \mathcal{T}, \mathrm{Can}_{\mathbf{Grp}})$ so by Lemma 12.2.8, we can decide if $G \cong H$ quantumly using $\mathsf{poly}(n) \leq S \leq \sqrt[3]{|L(\mathcal{T}(G))|} \cdot \mathsf{poly}(n)$ space in $T = n^{(1/2)\log_p n + O(1)}/\sqrt{S}$ time. □

12.4 Solvable-group isomorphism

In this section, we show a deterministic fourth-root speedup over the generator-enumeration algorithm for the class of solvable groups. This speedup is obtained by combining bidirectional collision detection with the algorithm from Chapter 11, which gave a square-root speedup over generator enumeration. We start by recalling the high-level structure of this algorithm.

Recall the definition of α-decompositions and α-composition pairs from Definitions 11.2.1 and 11.2.6. The algorithm can then be formulated as follows. First, we reduce solvable-group isomorphism to α-decomposition isomorphism in deterministic polynomial time using Lemma 11.1.2. Next, we use Lemma 11.1.3 to reduce from α-decomposition isomorphism to α-composition pair isomorphism in $n^{(1/2)\log_p n + O(1)}$ deterministic time. Finally, we apply Corollary 11.3.12 to solve α-composition pair isomorphism in $n^{\log_\alpha n + O(\alpha \log \alpha)}$ deterministic time.

In order to apply bidirectional collision detection to obtain an additional square-root speedup, we need to improve Lemma 11.1.3 by formulating the choice of the α-composition pair as a collision system. In order to do this, we need to represent the choices made when choosing a composition series as a tree. For this, we require some additional terminology.

A subgroup $H \leq G$ is *subnormal* (denoted $H \lhd\!\lhd G$) if there is a chain of subgroups

$H \lhd H_1 \lhd H_k \lhd G$. We call a chain of subgroups of the form $1 \lhd G_1 \lhd \cdots \lhd G_k \lhd\!\lhd G$ a *partial composition series* since it can be extended to a composition series. We construct a tree that corresponds to starting with the partial composition series $1 \lhd\!\lhd G$ and growing it by adding one subgroup at a time until we reach the composition series at the leaves.

Definition 12.4.1. *For each group G, the nodes of the tree $\mathcal{T}(G)$ are the partial composition series of G. The root is the partial composition series $1 \lhd\!\lhd G$. The children of each partial composition series $1 \lhd G_1 \lhd \cdots \lhd G_k \lhd\!\lhd G$ of G are the partial composition series $1 \lhd G_1 \lhd \cdots \lhd G_{k+1} \lhd\!\lhd G$.*

Now we are in a position to define the tree of choices for constructing a α-composition pair of an α-decomposition.

Definition 12.4.2. *For each α-decomposition (P_1, P_2) for a group G, we define $\mathcal{T}(P_1, P_2)$ to be the tree $\mathcal{T}(P_2)$. If (P_1, P_2) and (Q_1, Q_2) are α-decompositions and $\phi : (P_1, P_2) \to (Q_1, Q_2)$ is an isomorphism, we define $\mathcal{T}(\phi) : \mathcal{T}(P_1, P_2) \to \mathcal{T}(Q_1, Q_2)$ by $\mathcal{T}(\phi)[S_2]) = \phi[S_2])$ for each partial composition series S_2 for P_2.*

It is now easy to see that $(\alpha\text{-}\mathbf{Decomp}, \mathcal{T}, \mathrm{Can}_{\alpha\text{-}\mathbf{Decomp}})$ is a collision system where α-**Decomp** is the class of all α-decompositions and isomorphisms and $\mathrm{Can}_{\alpha\text{-}\mathbf{Decomp}}$ is defined by the algorithm of Corollary 11.3.17. In order to show that it is oracular, we need to show that the children of any node in the tree $\mathcal{T}(G)$ can be computed in polynomial time.

Lemma 12.4.3. *Let $1 \lhd G_1 \lhd \cdots \lhd G_k \lhd\!\lhd G$ be a partial composition series of G. Then we can compute all partial composition series of the form $1 \lhd G_1 \lhd \cdots \lhd G_{k+1} \lhd\!\lhd G$ determinstically in $\mathsf{poly}(n)$ time.*

Proof. A subgroup G_{k+1} contains G_k as a normal subgroup if and only if $G_{k+1} \leq \mathrm{N}_G(G_k)$. The partial composition series of the form $1 \lhd G_1 \lhd \cdots \lhd G_{k+1} \lhd\!\lhd G$ therefore correspond to the subgroups $G_{k+1} = \langle G_k, g\rangle$ for some $g \in \mathrm{N}_G(G_k)$ where G_{k+1}/G_k is simple and $G_{k+1} \lhd\!\lhd G$. Simplicity can be tested in polynomial time by computing normal closures of the cyclic subgroups; subnormality can be tested in polynomial time by checking if some $K_i = G_{k+1}$ where $K_1 = \langle G_{k+1}^G \rangle$ and each $K_{i+1} = \langle G_{k+1}^{K_i} \rangle$ (cf. [103]). □

We can now obtain an improved version of Lemma 11.1.3.

Lemma 12.4.4. *Testing isomorphism of the α-decompositions (P_1, P_2) and (Q_1, Q_2) of the groups G and H is Turing reducible to α-composition pair canonization*

(a) determinsitically using space $\mathsf{poly}(n) \leq S \leq n^{(1/4)\log_p n + O(1)}$ and time $T = n^{(1/2)\log_p n + O(1)}/S$

(b) quantumly using space $\mathsf{poly}(n) \leq S \leq n^{(1/6)\log_p n + O(1)}$ and time $T = n^{(1/4)\log_p n + O(1)}/\sqrt{S}$

where p is the smallest prime divisor of the order of the group.

Proof. By Lemma 12.4.3, $(\alpha\text{-}\mathbf{Decomp}, \mathcal{T}, \mathrm{Can}_{\alpha\text{-}\mathbf{Decomp}})$ is an oracular collision system. For each α-decomposition (R_1, R_2) for a group K, define $b_i(R_1, R_2) = |R_1| / (p(R_1))^i$ for $0 \leq i < \ell(R_1)$ and $b_i(R_1, R_2) = |R_2| / p(R_2)$ for $\ell(R_1) \leq i < \ell(R_1) + \ell(R_2)$ where $p(K)$ is the smallest prime dividing the order of K and $\ell(K)$ is the composition length of K for each group K. Then b is a bound for $(\alpha\text{-}\mathbf{Decomp}, \mathcal{T}, \mathrm{Can}_{\alpha\text{-}\mathbf{Decomp}})$. Since $\prod_{i=0}^{\ell(K)-1} (|K| / (p(K))^i \leq |K|) \leq |K|^{(1/2)\log_{p(K)}|K| + O(1)}$ for each group K (see Lemma 10.2.1),

$$
\begin{aligned}
N(P_1, P_2) &= \prod_{i=0}^{\ell(P_1)+\ell(P_2)-1} b_i(A) \\
&\leq |P_1|^{(1/2)\log_{p(P_1)}|P_1| + O(1)} \cdot |P_2|^{(1/2)\log_{p(P_2)}|P_2| + O(1)} \\
&\leq (|P_1|\,|P_2|)^{(1/2)(\log_{p(P_1)}|P_1| + \log_{p(P_2)}|P_2|) + O(1)} \\
&\leq n^{(1/2)\log_p n + O(1)}
\end{aligned}
$$

and a similar formula holds for (Q_1, Q_2). Part (a) is then immediate from Lemma 12.2.6.

For part (b), we observe that there is a natural bijection $\iota(R_1, R_2) : [b_0(R_1, R_2)] \times \cdots [b_{\ell(R_1)+\ell(R_2)-1}(R_1, R_2)] \to U(R_1, R_2)$ where $U(R_1, R_2)$ is a set that contains $L(\mathcal{T}(R_1, R_2))$ for each α-decomposition (R_1, R_2). Then (N, ι) is an index for $(\alpha\text{-}\mathbf{Decomp}, \mathcal{T}, \mathrm{Can}_{\alpha\text{-}\mathbf{Decomp}})$ and part (b) follows from Lemma 12.2.8. □

Our improved and generalized algorithms for solvable-group isomorphism now follow.

Theorem 12.4.5. *Solvable-group isomorphism can be solved*

(a) determinsitically using space $n^{O(\log n/\log\log n)} \leq S \leq n^{(1/4)\log_p n}$ and time $T = n^{(1/2)\log_p n + O(\log n/\log\log n)}/S$

(b) quantumly using space $n^{O(\log n/\log\log n)} \leq S \leq n^{(1/6)\log_p n + O(1)}$ and time $T = n^{(1/4)\log_p n + O(\log n/\log\log n)}/\sqrt{S}$

where p is the smallest prime divisor of the order of the group.

Proof. Combining the reductions of Lemma 11.1.2, Lemma 12.4.4 and Corollary 11.3.17 yields algorithms for solvable-group isomorphism. Choosing $\alpha = \log n/(\log\log n)^2$ completes the proof. □

12.5 Ring isomorphism

Similar results to those given for groups in the previous subsection can also be obtained for rings. Since the argument is very similar, we provide only a sketch. Let R be a ring. We define $\mathcal{T}(R)$ to be the tree consisting of non-redundant ordered generating sets of the additive group of R in the same way as for groups. The main issue is that $\text{Can}_{\mathbf{Grp}}$ only deals with a single operations since it is for groups whereas for rings we have two operations. Let $\mathbf{r}$ be an ordered generating set of the additive group of R. We address this issue by defining $\text{Can}_{\mathbf{Ring}}(R, \mathbf{r})$ to be the addition and multiplication tables of R with their elements relabeled according to their positions in the ordering $\prec_{\mathbf{r}}$ defined according to the additive group of R. Now, if Q is a ring and $\phi : R \to Q$ is a ring isomorphism, then $\text{Can}_{\mathbf{Ring}}(R, \mathbf{r}) = \text{Can}_{\mathbf{Ring}}(Q, \phi(\mathbf{r}))$. The same arguments used in the case of groups then imply the following result.

Theorem 12.5.1. *Let R and Q be rings of size n and let p be the smallest prime divisor of n. Then*

(a) using space $\mathsf{poly}(n) \leq S \leq n^{(1/2)\log_p n + O(1)}$, we can decide if $R \cong Q$ determinstically in $T = n^{\log_p n + O(1)}/S$ time. In particular, setting $S = n^{(1/2)\log_p n + O(1)}$ yields a deterministic $n^{(1/2)\log_p n + O(1)}$ time algorithm.

(b) using space $\mathsf{poly}(n) \le S \le |L(\mathcal{T}(R))| \le n^{(1/3)\log_p n+O(1)}$*, we can decide if* $R \cong Q$ *quantumly in* $T = n^{(1/2)\log_p n+O(1)}/\sqrt{S}$ *time. In particular, setting* $S = n^{(1/3)\log_p n+O(1)}$ *yields an* $n^{(1/3)\log_p n+O(1)}$ *time quantum algorithm.*

12.6 Worst-case graph isomorphism

We now show how bidirectional collision detection can be applied to obtain a speedup over the best algorithm previously known for graph isomorphism [16]. We start by reviewing the high-level structure of that algorithm. Since Luks showed an $n^{cd/\log d}$ time algorithm [16] for testing isomorphism of graphs of color-degree at most d (see Chapter 8), one strategy for obtaining an algorithm for testing isomorphism of general graphs is to Turing-reduce testing isomorphism of arbitrary graphs to many instances of testing isomorphism of graphs of smaller color-degree. (The color-degree of a node is at most d if for every color, either there are no more than d neighbors with that color or there are no more than d non-neighbors with that color.) Given a graph, the color-degree may be reduced as follows. Suppose that we wish to ensure that the color-degree is at most d. We choose a vertex with color-degree more than d and individualize it by replacing its color with a new color that is distinct from all other colors in the graph. The partition induced by the new coloring is then refined using a process (cf. [7]) that takes into account the colors of the nodes and the structure of the graph. Repeating this process, one can show that after a sequence of $4n/d$ nodes have been chosen, the graph has color-degree at most d. Two algorithms A_1 and A are defined below based on the procedure we just sketched. Algorithm A_1 takes a sequence of nodes as input and outputs the coloring that results from applying the above process. Algorithm A_2 takes a sequence of nodes and outputs the next node in the sequence chosen according to the above process. This results in the following lemma, which is a slightly strengthened version of Lemma 8.6.1.

Lemma 12.6.1 (Zemlyachenko, cf. [7])**.** *Let* X *and* Y *be graphs of size* n*. There is a deterministic polynomial-time algorithm* A_1 *that takes a graph and a sequence of vertexes as* *its input such that*

(a) if $\phi : X \to Y$ is an isomorphism, then for any sequence of nodes $x_1, \ldots, x_m$ in X, ϕ is also an isomorphism from $A_1(X, (x_1, \ldots, x_m))$ to $A_1(Y, (\phi(x_1), \ldots, \phi(x_m)))$ and

(b) if $X \ncong Y$, then for all sequences of nodes $x_1, \ldots, x_m$ and $y_1, \ldots, y_m$ in X and Y, $A_1(X, (x_1, \ldots, x_m)) \ncong A_1(Y, (y_1, \ldots, y_m))$.

Moreover, we also have a deterministic polynomial-time algorithm A_2 that takes a graph and a sequence of vertexes as its input such that

(c) algorithm A returns a set of nodes and

(d) if we start with the empty sequence () and choose $4n/d$ nodes $x_1, \ldots, x_{4n/d}$ in X by successive calls to A_2 such that each x_i is in the set of nodes returned by A_2 at the i^{th} call, then $A_1(X, (x_1, \ldots, x_{4n/d}))$ has color-degree at most d.

To obtain an algorithm for graph isomorphism, we compute a sequence of $4n/d$ nodes $x_1, \ldots, x_{4n/d}$ in X using A such that $A_1(X, (x_1, \ldots, x_{4n/d}))$ has color-degree at most d; we then consider all $n^{4n/d}$ possible sequences $y_1, \ldots, y_{4n/d}$ of $4n/d$ nodes in Y and check if $A_1(X, (x_1, \ldots, x_{4n/d})) \cong A_1(Y, (y_1, \ldots, y_{4n/d}))$ for one of these sequences. This occurs if and only if $X \cong Y$. By combining with an $n^{cd/\log d}$ algorithm [16] for computing canonical forms of graphs of color-degree at most d, we obtain an $n^{4n/d + cd/\log d + O(1)}$ algorithm for graph isomorphism where d is a parameter that we choose. Minimizing the runtime over d, we get $d = c'\sqrt{n \log n}$ where c' is a constant we choose. This yields the best algorithm known for graph isomorphism that we mentioned in Chapter 8.

Theorem 8.6.2 (Babai, Kantor and Luks [16])**.** *Graph isomorphism can be decided in $2^{O(\sqrt{n \log n})}$ time.*

Optimizing the constant in the exponent by choosing $c' = \sqrt{2/c}$, we obtain a runtime of $2^{(4\sqrt{2c})\sqrt{n \log n} + O(\log n)}$.

Our contribution to this problem is to note that bidirectional collision detection can be applied to Lemma 12.6.1 in order to reduce the total number of sequences of nodes that must be considered to at most $2n^{2n/d}$. First, we define the tree $\mathcal{T}^d(X)$ for each graph X.

Definition 12.6.2. *For each graph X, $\mathcal{T}^d(X)$ is a tree whose nodes are sequences of nodes in X rooted at the empty sequence $()$. To construct $\mathcal{T}^d(X)$, we start at the root and define its children to be $\{(x_1) \mid x_1 \in A_2(X, ())\}$; for a node $(x_1, \ldots, x_k)$ with $k < 4|X|/d$, we define its children to be $\{(x_1, \ldots, x_k, x_{k+1}) \mid x_{k+1} \in A_2(X, (x_1, \ldots, x_k))\}$.*

If X and Y are graphs and $\phi : X \to Y$ is an isomorphism, we define $\mathcal{T}^d(\phi) : \mathcal{T}^d(X) \to \mathcal{T}^d(Y)$ by $\mathcal{T}^d(\phi)(x_1, \ldots, x_k) = (\phi(x_1), \ldots, \phi(x_k))$ for each $(x_1, \ldots, x_k) \in V(\mathcal{T}(X))$.

Thus, all of the nodes of $\mathcal{T}(X)$ are sequences $(x_1, \ldots, x_k)$ of at most $4n/d$ nodes that can be extended to a sequence $(x_1, \ldots, x_{4n/d})$ of $4n/d$ nodes such that $A_1(X, (x_1, \ldots, x_{4n/d}))$ has color-degree at most d.

For a sequence of nodes $(x_1, \ldots, x_k)$ in a graph X, let $\mathrm{Can}_{\mathbf{Graph}}(X, (x_1, \ldots, x_k))$ be the graph obtained by computing a canonical form of $A_1(X, (x_1, \ldots, x_k))$.

Lemma 12.6.3. *For each d, we can Turing-reduce testing isomorphism of the graphs X and Y to calls to calls to $\mathrm{Can}_{\mathbf{Graph}}$ on graphs of color-degree at most d deterministically in $n^{2n/d+O(1)}$ time.*

Proof. It follows from Definition 12.6.2 and Lemma 12.6.1 that $(\mathbf{Graph}, \mathcal{T}^d, A_1)$ is an oracular collision system. Letting each $b_i(Z) = |Z|$ for each graph Z, we see that b is a bound for $(\mathbf{Graph}, \mathcal{T}, A_1)$. Then result then follows from Lemma 12.2.6. □

We remark that a time-space tradeoff (which we omit) also exists for this problem.

Combining this result with Luks' algorithm [16, 77] for computing canonical forms of graphs of color-degree at most d in $n^{cd/\log d}$ time, we obtain a speedup over the best algorithm previously known graph isomorphism.

Theorem 12.6.4. *Graph isomorphism can be decided in $2^{(4\sqrt{c})\sqrt{n \log n}+O(\log n)}$ deterministic time.*

Proof. Let X and Y be graphs of size n and let $d \leq n$ be a parameter that we shall chose later. By Lemma 12.6.3 and Theorem 8.6.2, we can test if $X \cong Y$ deterministically in $n^{2n/d+cd/\log d+O(1)}$ time. Optimizing over d, we find that $d = c'\sqrt{n\log n}$ where c' is a constant we choose. The optimal choice is $c' = 1/\sqrt{c}$ which yields an overall complexity of $2^{(4\sqrt{c})\sqrt{n\log n}} + O(\log n)$. □

This reduces the constant in the exponent of the previous best runtime of $2^{(4\sqrt{2c})\sqrt{n\log n}+O(\log n)}$ by a factor of $1/\sqrt{2}$. We can also prove a quantum analogue of Theorem 12.6.4.

Theorem 12.6.5. *Graph isomorphism can be decided in* $2^{(4\sqrt{2c/3})\sqrt{n\log n}+O(\log n)}$ *quantum time.*

Proof. Let X and Y be graphs of size n. By Lemma 12.6.3, $(\mathbf{Graph}, \mathcal{T}^d, \mathrm{Can}_{\mathbf{Graph}})$ is an oracular collision system. For each graph Z, let $U(Z) = [|Z|]^{4|Z|/d}$, define $M(Z) = |Z|^{4|Z|/d}$ and let $\iota(Z) : [M(Z)] \to U(Z)$ be an arbitrary bijection that can be evaluated in $\mathsf{poly}(|Z|)$ time. Applying Lemma 12.2.8 and setting $S = n^{4n/3d+O(1)}$ yields an $n^{4n/3d+cd/\log d+O(1)}$ time quantum algorithm. Optimizing over d as in the deterministic case, $d = c'\sqrt{n\log n}$ where c' is a constant we choose. The optimal choice is now $c' = \sqrt{2/3c}$ which yields an overall complexity of $2^{(4\sqrt{2c/3})\sqrt{n\log n}+O(\log n)}$. □

Time space tradeoff analogues of Theorems 12.6.4 and 12.6.5 also hold and are easy to prove using the same techniques.

12.7 Conclusion

In this chapter, we introduced the bidirectional collision-detection technique and used it to obtain speedups over the previous best algorithms for the group, ring and graph isomorphism problems. We summarize the state of the art for the isomorphism problems considered in this chapter in Table 12.1. We use the notation T^δ to indicate that the original runtime T has been reduced to T^δ.

It is interesting to note that there is currently no advantage for randomized algorithms over deterministic algorithms in this regime. We consider the question of whether such

Class of objects	**Runtime**	**Paradigm**	**Speedup**
General groups	$n^{(1/2)\log n + O(1)}$	Deterministic	$T^{1/2}$
General groups	$n^{(1/3)\log n + O(1)}$	Quantum	$T^{2/3}$
Solvable groups	$n^{(1/4)\log n + O(\log n / \log\log n)}$	Deterministic	$T^{1/2}$
Solvable groups	$n^{(1/6)\log n + O(\log n / \log\log n)}$	Quantum	$T^{1/2}$
Rings	$n^{(1/2)\log n + O(1)}$	Deterministic	$T^{1/2}$
Rings	$n^{(1/3)\log n + O(1)}$	Quantum	$T^{2/3}$
Graphs	$2^{4\sqrt{c}\sqrt{n\log n} + O(\log n)}$	Deterministic	$T^{1/\sqrt{2}}$
Graphs	$2^{4\sqrt{2c/3}\sqrt{n\log n} + O(1)}$	Quantum	T^{1}

Table 12.1: Algorithms for isomorphism problems

algorithms exist to be an interesting open problem; the techniques used in the author's previous work [108] for constructing faster randomized algorithms no longer suffice so new ideas appear to be required.

Chapter 13

CONCLUSION

In this thesis, we studied various problems in quantum computing and isomorphism testing. In Chapter 5, we addressed the problem of mapping quantum algorithms into practical quantum architectures. This is important since abstract quantum algorithms can perform interactions between arbitrary pairs of qubits, while in a physical implementation of a quantum computer, only qubits that neighbor each other in space can interact. The main result of Chapter 5 showed that any abstract quantum algorithm can be mapped into a natural two-dimensional architecture with only a constant factor increase in the depth. Since this architecture models many quantum computing technologies, our result justifies the assumptions made in many quantum algorithms.

Next, in Chapter 6 we studied an extension of the standard oracle model that results when the oracle is allowed to behave differently based on the outcomes of private coin flips. While this model might seem odd at first glance, it is quite natural from a quantum mechanics perspective, since such oracles correspond to random physical processes. We introduced the notion of an infinity-vs-one separation, which arises when a quantum algorithm can solve an oracle problem using a single query but classical algorithms cannot solve it no matter how many queries are used. We also studied when some number of classical or quantum queries can learn anything about the solution to an oracle problem, and showed (roughly speaking) that k quantum queries can extract information if and only if $2k$ classical queries can extract information.

In Chapter 7, we moved on to the tree isomorphism problem. While there are efficient classical algorithms [4] for tree isomorphism, we considered the problem of computing a quantum state that represents the isomorphism classes of trees. This is known as the state preparation approach to graph isomorphism [3], and is a promising approach to developing efficient quantum algorithms for the graph isomorphism problem. It is therefore important to know that it at least works for trees, since otherwise there would be no hope of using the state preparation approach to test isomorphism of classes of graphs that seem difficult classically. Along the way, we also developed state symmetrization primitives for rearranging permutations of quantum states from certain types collections of states. These primitives may be of interest in other contexts as well.

While Chapter 7 fits into both the quantum computation and isomorphism testing themes of this thesis, in Chapter 10, we move firmly into the domain of isomorphism testing by studying the group isomorphism problem. For several decades, the best worst-case algorithm known for sufficiently general classes of groups was the generator-enumeration algorithm, which is essentially brute force. Our main result in Chapter 10 is a square-root speedup over the generator-enumeration algorithm. Thus, our result answers in the affirmative a longstanding open problem [72, 73]. By introducing additional group theoretic machinery, we are able to generalize this speedup to the larger class of solvable groups in Chapter 11.

In Chapter 12, we consider not only the group isomorphism problem, but also isomorphism problems for many other objects as well. In fact, our main result gives a general lemma for obtaining square-root speedups over algorithms for any isomorphism problem that satisfies certain mild assumptions. In particular, this lemma allows us to obtain a square-root speedup over the generator-enumeration algorithm for the problem of testing isomorphism of arbitrary groups. By combining this idea with the methods of Chapters 10 and 11, we also further improve our results for p- and solvable group isomorphism by obtaining fourth-root speedups.

13.1 Open problems

We leave several problems open. In a work that builds on Chapter 5, we will show that the quadratic increase in the number of qubits needed when simulating abstract quantum circuits is sometimes unavoidable. However, we will also show that in some cases, we can retain only a constant factor increase in the depth while using significantly fewer qubits.

One interesting potential application of the results in Chapter 6 would be to devise a protocol which could be used to prove that a black box is in fact a quantum computer. While it is easy to see how to use the results of Chapter 6 to prove that a device has some quantum characteristics, it is not obvious how prove that it has the full power of a universal quantum computer.

The main problem left open by Chapter 7 is the question of whether complete invariant states can be efficiently prepared for classes of graphs that appear to be difficult classically. A less ambitious problem is to improve the $O(n^5)$ time algorithm of Theorem 7.3.1. It seems like the correct runtime should be $O(n \log n)$ time, but it is not immediately clear how we can do better than $O(n^5)$ time.

The main open question in group isomorphism is whether there is a polynomial time algorithm. A less ambitious open problem — that would still be a huge breakthrough — is to show that group isomorphism can be solved in $n^{o(\log n)}$ time. Achieving these results for p- or solvable groups would be almost as impressive of a breakthrough.

Another interesting question is whether the bidirectional collision detection techniques of Chapter 12 can be improved beyond providing square-root speedups using randomization. While matching lower bounds exist for general collision problems, it is not clear if these lower bounds extend to isomorphism problems. Obtaining either a better upper bound or a matching lower bound would be intriguing.

BIBLIOGRAPHY

[1] D. Aharonov and M. Ben-Or. Fault-tolerant quantum computation with constant error. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 176–188, 1997.

[2] D. Aharonov, M. Ben-Or, R. Impagliazzo, and N. Nisan. Limitations of noisy reversible computation. 1996, `quant-ph/9611028`.

[3] D. Aharonov and A. Ta-Shma. Adiabatic quantum state generation and statistical zero knowledge. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 20–29, 2003.

[4] A. A. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[5] M. Artin. *Algebra*. Pearson Prentice Hall, 2010.

[6] L. Babai. Bounded round interactive proofs in finite groups. *SIAM Journal on Discrete Mathematics*, 5(1):88–111, 1992.

[7] L. Babai. Moderately exponential bound for graph isomorphism. In *Proceedings of the 1981 International FCT-Conference on Fundamentals of Computation Theory*, pages 34–50, 1981.

[8] L. Babai. Monte-Carlo algorithms in graph isomorphism testing. Technical report, Université de Montréal, 2010.

[9] L. Babai. Trading group theory for randomness. In *Proceedings of the Seventeenth Annual ACM Symposium on the Theory of Computing*, pages 421–429, 1985.

[10] L. Babai, P. Cameron, and P. Pálfy. On the orders of primitive groups with restricted nonabelian composition factors. *Journal of Algebra*, 79(1):161–168, 1982.

[11] L. Babai and P. Codenotti. Isomorphism of hypergraphs of low rank in moderately exponential time. In *IEEE 49th Annual IEEE Symposium on the Foundations of Computer Science*, pages 667–676, 2008.

[12] L. Babai, P. Codenotti, J. A. Grochow, and Y. Qiao. Code equivalence and group isomorphism. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1395–1408, 2011.

[13] L. Babai, P. Codenotti, and Y. Qiao. Polynomial-time isomorphism test for groups with no abelian normal subgroups (extended abstract). In *39th International Colloquium on Automata, Languages and Programming*, pages 51–62, 2012.

[14] L. Babai, G. Cooperman, L. Finkelstein, E. Luks, and Á. Seress. Fast monte carlo algorithms for permutation groups. *Journal of Computer and System Sciences*, 50(2):296–308, 1995.

[15] L. Babai, G. Cooperman, L. Finkelstein, and Á. Seress. Nearly linear time algorithms for permutation groups with a small base. In *Proceedings of the 1991 international symposium on Symbolic and algebraic computation*, pages 200–209, 1991.

[16] L. Babai, W. M. Kantor, and E. M. Luks. Computational complexity and the classification of finite simple groups. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 162–171, 1983.

[17] L. Babai and L. Kucera. Canonical labelling of graphs in linear average time. In *20th Annual Symposium on the Foundations of Computer Science*, pages 39–46, 1979.

[18] L. Babai and E. M. Luks. Canonical labeling of graphs. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 171–183, 1983.

[19] L. Babai and Y. Qiao. Polynomial-time isomorphism test for groups with abelian Sylow towers. In *29th International Symposium on Theoretical Aspects of Computer Science*, pages 453–464, 2012.

[20] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, 1995.

[21] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters*, 70(13):1895, 1993.

[22] H. U. Besche, B. Eick, and E. A. O'Brien. A millennium project: Constructing small groups. *International Journal of Algebra and Computation*, 12(5):623–644, 2002.

[23] K. Booth and C. Colbourn. *Problems polynomially equivalent to graph isomorphism.* Computer Science Department, University of Waterloo, 1979.

[24] R. B. Boppana, J. Håstad, and S. Zachos. Does coNP have short interactive proofs? *Information Processing Letters*, 25(2):127–132, 1987.

[25] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. 1996, `quant-ph/9605034`.

[26] G. Brassard, P. Hoyer, and A. Tapp. Quantum algorithm for the collision problem. 1997, `quant-ph/9705002`.

[27] D. E. Browne, E. Kashefi, and S. Perdrix. Computational depth complexity of measurement-based quantum computation. In *In Proceedings of the Fifth Conference on the Theory of Quantum Computation, Communication and Cryptography*, 2010.

[28] H. Buhrman, R. Cleve, J. Watrous, and R. de Wolf. Quantum fingerprinting. *Physical Review Letters*, 87(16):167902, 2001.

[29] A. Chattopadhyay, J. Torán, and F. Wagner. Graph isomorphism is not AC^0 reducible to group isomorphism. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 317–326, 2010.

[30] D. Cheung, D. Maslov, and S. Severini. Translation techniques between quantum circuit architectures. In *Workshop on Quantum Information Processing*, 2007.

[31] A. M. Childs and W. Van Dam. Quantum algorithms for algebraic problems. *Reviews of Modern Physics*, 82(1):1, 2010, `0812.0380`.

[32] B.-S. Choi and R. Van Meter. An $\Theta(\sqrt{n})$-depth quantum adder on a 2D NTC quantum computer architecture. *ACM Journal on Emerging Technologies in Computing Systems*, 8(3):24, 2012, `1008.5093`.

[33] B.-S. Choi and R. Van Meter. On the effect of quantum interaction distance on quantum addition circuits. *ACM Journal on Emerging Technologies in Computing Systems*, 7(3):11:1–11:17, 2011.

[34] P. Codenotti. *Testing Isomorphism of Combinatorial and Algebraic Structures.* PhD thesis, University of Chicago, 2011.

[35] J. Cong, Y. Fan, G. Han, and Z. Zhang. Application-specific instruction generation for configurable processor architectures. In *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, pages 183–189, 2004.

[36] D. Copsey, M. Oskin, F. Impens, T. Metodiev, A. Cross, F. T. Chong, I. L. Chuang, and J. Kubiatowicz. Toward a scalable, silicon-based quantum computing architecture. *IEEE Journal of Selected Topics in Quantum Electronics*, 9(6):1552–1569, 2003.

[37] P. Darga, K. Sakallah, and I. Markov. Faster symmetry discovery using sparsity of symmetries. In *Proceedings of the 45th annual Design Automation Conference*, pages 149–154, 2008.

[38] J. N. De Beaudrap, R. Cleve, and J. Watrous. Sharp quantum versus classical query complexity separations. *Algorithmica*, 34(4):449–461, 2002, `quant-ph/0011065`.

[39] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. In *Royal Society of London*, 1992.

[40] J. Dixon and B. Mortimer. *Permutation Groups.* Graduate Texts in Mathematics Series. Springer-Verlag, 1996.

[41] M. Ettinger and P. Hoyer. A quantum observable for the graph isomorphism problem. 1999, `quant-ph/9901029`.

[42] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. Limit on the speed of quantum computation in determining parity. *Physical Review Letters*, 81(24):5442–5444, 1998.

[43] W. Feit and J. Thompson. Solvability of groups of odd order. *Pacific journal of mathematics*, 13(3):775–1029, 1963.

[44] V. Felsch and J. Neubüser. On a programme for the determination of the automorphism group of a finite group. In *Computational Problems in Abstract Algebra*, pages 59–60, 1970.

[45] A. G. Fowler, S. J. Devitt, and L. C. L. Hollenberg. Implementation of Shor's algorithm on a linear nearest neighbour qubit array. 2004, `quant-ph/0402196`.

[46] C. D. Godsil and B. D. McKay. A new graph product and its spectrum. *Bulletin of the Australian Mathematical Society*, 18(1):21–28, 1978.

[47] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690–728, 1991.

[48] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[49] S. Golomb. On the classification of boolean functions. *IRE Transactions on Circuit Theory*, 6(5):176–186, 1959.

[50] D. Gottesman and I. Chuang. Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *Nature*, 402(6760):390–393, 1999.

[51] J. A. Grochow and Y. Qiao. Algorithms for group isomorphism via group extensions and cohomology. 2013, `1309.1776`.

[52] 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, 1996, `quant-ph/9605043`.

[53] P. Hall. On the Sylow systems of a soluble group. *Proceedings of the London Mathematical Society*, s2-43(1):316–323, 1938.

[54] S. Hallgren, C. Moore, M. Rötteler, A. Russell, and P. Sen. Limitations of quantum coset states for graph isomorphism. In *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, pages 604–617, 2006.

[55] A. W. Harrow and D. J. Rosenbaum. Uselessness for an oracle model with internal randomness. *Quantum Information and Computation*, 14(7&8), May 2014, `1111.1462`.

[56] C. M. Hoffmann. *Group Theoretic Algorithms and Graph Isomrophism*. Springer, 1982.

[57] P. Høyer and R. Špalek. Quantum fan-out is powerful. *Theory of Computing*, 1(5):81–103, 2005.

[58] T. Hungerford. *Algebra*. Graduate Texts in Mathematics. Springer, 1974.

[59] T. Junttila and P. Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics*, pages 135–149, 2007.

[60] W. Kantor. Polynomial-time algorithms for finding elements of prime order and Sylow subgroups. *Journal of Algorithms*, 6(4):478–514, 1985.

[61] W. Kantor and D. Taylor. Polynomial-time versions of Sylow's theorem. *Journal of Algorithms*, 9(1):1–17, 1988.

[62] H. Katebi, K. A. Sakallah, and I. L. Markov. Graph symmetry detection and canonical labeling: Differences and synergies. 2012, `1208.6271`.

[63] T. Kavitha. Linear time algorithms for Abelian group isomorphism and related problems. *Journal of Computer and System Sciences*, 73(6):986–996, 2007.

[64] A. Y. Kitaev. Quantum measurements and the abelian stabilizer problem. 1995, `quant-ph/9511026`.

[65] M. Klin, C. Rücker, G. Rücker, and G. Tinhofer. Algebraic combinatorics in mathematical chemistry. methods and algorithms. i. permutation groups and coherent (cellular) algebras. *Match*, 40:7–138, 1999.

[66] G. Kuperberg. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. 2011.

[67] G. Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM Journal on Computing*, 35(1):170–188, 2005, `quant-ph/0302112`.

[68] S. A. Kutin. Shor's algorithm on a nearest-neighbor machine. 2006, `quant-ph/0609001`.

[69] S. Lang. *Algebra*. Springer, 2002.

[70] F. Le Gall. Efficient isomorphism testing for a class of group extensions. 2008, `0812.2298`.

[71] M. Lewis and J. Wilson. Isomorphism in expanding families of indistinguishable groups. *Groups-Complexity-Cryptology*, 4(1):73–110, 2012.

[72] R. Lipton. An annoying open problem. Gödel's Lost Letter and $\mathsf{P} = \mathsf{NP}$, 2011.

[73] R. Lipton. The group isomorphism problem: A possible polymath problem? Gödel's Lost Letter and $\mathsf{P} = \mathsf{NP}$, 2011.

[74] R. Lipton, L. Snyder, and Y. Zalcstein. *The Complexity of Word and Isomorphism Problems for Finite Groups*. Defense Technical Information Center, 1977.

[75] E. Luks. Hypergraph isomorphism and structural equivalence of boolean functions. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of computing*, pages 652–658, 1999.

[76] E. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982.

[77] E. M. Luks. Permutation groups and polynomial-time computation. In *Groups and Computation 1991*, volume 11, page 139, 1993.

[78] D. Maslov. Linear depth stabilizer and quantum fourier transformation circuits with no auxiliary qubits in finite-neighbor quantum architectures. *Physical Review A*, 76(5):052310, 2007, `quant-ph/0703211`.

[79] R. Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–136, 1979.

[80] B. McKay. Practical graph isomorphism, 1981.

[81] B. D. McKay and A. Piperno. Practical graph isomorphism, II. 2013, `1301.1493`.

[82] D. A. Meyer and J. Pommersheim. On the uselessness of quantum queries. *Theoretical Computer Science*, 412(51):7068–7074, 2011, `1004.1434`.

[83] D. A. Meyer and J. Pommersheim. Single query learning from Abelian and non-Abelian Hamming distance oracles. 2009, `0912.0583`.

[84] G. L. Miller. On the $n^{\log n}$ isomorphism technique (a preliminary report). In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, pages 51–58, 1978.

[85] A. Montanaro, H. Nishimura, and R. Raymond. Unbounded-error quantum query complexity. In *Algorithms and Computation*, pages 919–930. Springer, 2008, `0712.1446`.

[86] C. Moore. Quantum circuits: Fanout, parity, and counting. 1999, `quant-ph/9903046`.

[87] C. Moore, A. Russell, and L. Schulman. The symmetric group defies strong fourier sampling. *SIAM Journal on Computing*, 37(6):1842–1864, 2008.

[88] C. Moore, A. Russell, and P. Sniady. On the impossibility of a quantum sieve algorithm for graph isomorphism. *SIAM Journal on Computing*, 39(6):2377–2396, 2010.

[89] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.

[90] E. A. O'Brien. Isomorphism testing for p-groups. *Journal of Symbolic Computation*, 17(2):133–147, 1994.

[91] P. Papakonstantinou. The depth irreducibility hypothesis. *Electronic Colloquium on Computational Complexity*, 2014.

[92] P. Pham and K. M. Svore. A 2d nearest-neighbor quantum architecture for factoring in polylogarithmic depth. *Quantum Information & Computation*, 13(11–12):937–962, 2013, `1207.6655`.

[93] L. Pyber. Asymptotic results for permutation groups. In *Workshop on Groups and Computation*, 1991.

[94] Y. Qiao, J. Sarma, and B. Tang. On isomorphism testing of groups with normal Hall subgroups. In *28th International Symposium on Theoretical Aspects of Computer Science*, pages 567–578, 2011.

[95] R. Raussendorf and H. J. Briegel. A one-way quantum computer. *Physical Review Letters*, 86(22):5188–5191, 2001.

[96] R. Raussendorf, D. Browne, and H. Briegel. Measurement-based quantum computation on cluster states. *Physical Review A*, 68(2):022312, 2003.

[97] R. Raussendorf, D. E. Browne, and H. J. Briegel. The one-way quantum computer–a non-network model of quantum computation. *Journal of Modern Optics*, 49(8):1299–1306, 2002, `quant-ph/0108118`.

[98] O. Regev. A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space. 2004, `quant-ph/0406151`.

[99] O. Regev and L. Schiff. Impossibility of a quantum speed-up with a faulty oracle. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I*, pages 773–781, 2008.

[100] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, pages 358–366, 1953.

[101] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[102] D. Robinson. *A Course in the Theory of Groups*. Graduate Texts in Mathematics. Springer-Verlag, 1996.

[103] S. Roman. *Fundamentals of Group Theory: An Advanced Approach*. Springer, 2011.

[104] S. Roman, S. M. Roman, and S. M. Roman. *Advanced linear algebra*, volume 3. Springer, 2005.

[105] H. E. Rose. *A Course on Finite Groups.* Springer, 2009.

[106] D. J. Rosenbaum. Beating the generator-enumeration bound for solvable-group isomorphism. December 2014, `1412.0639`. Submitted to Transactions on Computation Theory.

[107] D. J. Rosenbaum. Bidirectional collision detection and faster deterministic isomorphism testing. April 2013, `1304.3935`.

[108] D. J. Rosenbaum. Breaking the $n^{\log n}$ barrier for solvable-group isomorphism. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1054–1073, January 2013, `1205.0642`.

[109] D. J. Rosenbaum. Optimal quantum circuits for nearest-neighbor architectures. In *Eigth Conference on the Theory of Quantum Computation, Communication and Cryptography*, volume 22, pages 294–307, May 2013, `1205.0036`.

[110] D. J. Rosenbaum. Quantum algorithms for tree isomorphism and state symmetrization. August 2010, `1011.4138`.

[111] D. J. Rosenbaum and F. Wagner. Beating the generator-enumeration bound for p-group isomorphism. December 2013, `1312.1755`. Submitted to Theoretical Computer Science.

[112] J. Rotman. *An Introduction to the Theory of Groups.* Graduate Texts in Mathematics. Springer, 1995.

[113] C. Savage. *An $O(n^2)$ algorithm for Abelian group isomorphism.* Computer Studies Program, North Carolina State University, 1980.

[114] Á. Seress. *Permutation Group Algorithms.* Cambridge Tracts in Mathematics. Cambridge University Press, 2003.

[115] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Annual Symposium on Foundations of Computer Science*, 1994.

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

[117] C. Sims. Computation with permutation groups. In *Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*, pages 23–28. ACM, 1971.

[118] D. Slepian. On the number of symmetry types of boolean functions of n variables. *Canadian Journal of Mathematics*, 5(2):185–193, 1953.

[119] D. Spielman. Faster isomorphism testing of strongly regular graphs. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of computing*, pages 576–584, 1996.

[120] M. Sudan. Algebra and computation. Lecture notes, 2005.

[121] Y. Takahashi and S. Tani. Collapse of the hierarchy of constant-depth exact quantum circuits. 2011, `1112.6063`.

[122] Y. Takahashi, S. Tani, and N. Kunihiro. Quantum addition circuits and unbounded fan-out. *Quantum Information and Computation*, 10(9):872–890, 2010, `0910.2530`.

[123] B. M. Terhal and D. P. DiVincenzo. Adaptive quantum computation, constant depth quantum circuits and Arthur-Merlin games. 2002, `quant-ph/0205133`.

[124] R. Van Meter and K. M. Itoh. Fast quantum modular exponentiation. *Phys. Rev. A*, 71(5):052320, 2005.

[125] N. Vikas. An $O(n)$ algorithm for Abelian p-group isomorphism and an $O(n \log n)$ algorithm for Abelian group isomorphism. *Journal of Computer and System Sciences*, 53(1):1–9, 1996.

[126] F. Wagner. On the complexity of group isomorphism. *Electronic Colloquium on Computational Complexity*, 2011.

[127] F. Wagner. On the complexity of group isomorphism. *Electronic Colloquium on Computational Complexity*, 2012. Revision 2.

[128] R. Wilson. *The Finite Simple Groups*. Springer, 2010.

[129] Y. Wong. Hierarchical circuit verification. In *Proceedings of the Twenty-Second ACM-IEEE Design Automation Conference*, pages 695–701, 1985.