Quantum Data Structures

Shashvat Shukla

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of
University College London.

Department of Physics and Astronomy
University College London

September 23, 2026

Declaration

This thesis is a presentation of original work. This work has not previously been presented for a degree or other qualification at this university or elsewhere. All input or assistance in the creation of the academic work other than from the supervisory team (including AI) has been acknowledged below. Any text or data in the thesis that has been presented for publication (including in review) elsewhere is declared below.

Data sources, if not collected yourself: n/a

Software code, instrumentation development, etc: n/a

AI: Generative AI was used to transcribe some handwritten drafts into LaTeX code, generate TeX figures from text prompts, and as part of the proofreading and editing process.

Other: n/a

Publications: Material from the following papers has been reproduced in this thesis:

1. **Shukla, S.**, "A good first impression for quantum algorithms" *IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 3, pages 129–133, 2025.

2. **Shukla, S.**, "Explaining quantum phase estimation using data structures" *American Journal of Physics* (under review).

Abstract

Designing quantum algorithms that outperform the best classical algorithms has been a grand challenge in computer science. In this thesis, we explore an approach to designing quantum algorithms that begins by first thinking about the data type in the problem and how it may be encoded in quantum states. We call this approach, "quantum data structures"– after the analogous concept in classical computing: data structures, which are often the key to designing efficient classical algorithms. We first give a revised and illuminating presentation of some standard quantum algorithms by highlighting the data structures used in them. We then survey the disparate literature on quantum data structures to identify the key challenges and opportunities in this approach. This thesis then defines new quantum data structures, drawing on mathematical tools like hypergraph states and classical error correcting codes. Lastly, we outline the scenarios in which quantum data structures achieve an advantage over classical data structures, and directions for future work.

Impact Statement

Quantum computers will be more efficient and more secure than the computers we have now. These are important themes in our present moment when the energy and other resource requirements of computation are straining societies and the Earth, particularly in the case of AI data centres. Also, as more of the world is digitalised, weaknesses in information security are risks to the stability of social systems and public infrastructure. Therefore, by this first approximation, any progress in quantum computing is of public benefit. More specifically, this thesis advances public benefit in two ways: it improves the *accessibility* of quantum computing theory, and it advances our understanding of the *uses* of quantum computers by proposing new algorithmic primitives for designing quantum algorithms. Beyond material included in this thesis, I have also worked on raising awareness on the *misuses* of quantum computers in breaking cryptography and on the potential *abuse* of sensitive data within quantum computer systems.

Some of the novel ideas presented in this thesis are the result of my own struggles in making sense of quantum algorithms throughout my undergraduate and postgraduate studies. I felt for many years that while I could do the algebraic computations necessary to prove that a quantum algorithm worked correctly and efficiently, I did not have the intuitive grasp of quantum algorithms as I did of classical algorithms. This turned out to be the situation for many of my peers as well. The frameworks developed in this thesis make quantum computing more accessible because they provide simple and conceptual ways of describing quantum algorithms. This has already had an impact in the teaching of quantum algorithms in UCL, where material that was previously split between multiple lectures is now

being taught in just a single lecture, because laborious algebraic derivations that needed to be explained before can now be avoided. There are many public benefits of this simplicity. For example, if these ideas are expressed simply, then the quantum programs we will write to express them will also be simpler to analyse, which means they will be easier to use, check, and maintain. I am also hopeful that such simplifications will allow quantum computing to be better adopted in those regions and organisations that will not have access to the world's top experts in quantum computing.

Public investment in quantum computing will be justified if we find applications of quantum computing. We are currently seeing steady progress in all aspects of scaling up quantum computer systems, however, one grand challenge that remains is the design of new practical quantum algorithms beyond the relatively small set of practical quantum algorithms that are currently known. In my PhD research, I sought to specifically invent new quantum algorithmic *primitives*, because I felt that that was what was most lacking in society's effort to build useful quantum computers. The algorithms and data structures developed in this thesis are new primitives because they work in entirely new ways. They rely on new mathematical connections to achieve quantum advantage, and new ways of thinking about the design of quantum algorithms. I hope that this framework and the algorithms that it inspires will be one path to quantum advantage.

I also believe it is important to anticipate and proactively mitigate potential harms and misuses of one's research. By working in quantum computing, we are advancing a technology that we know will weaken cryptographic infrastructure. While not part of the content of this thesis, I have also been working on public outreach of the quantum threat to cryptography through my Quantum Time Capsule project [2], which has allowed me to build interest and raise awareness of this topic among a wide variety of stakeholders. In the project, we invite members of the general public to purposely encrypt messages with cryptographic schemes like RSA that can be broken by quantum computers. These messages become like time

capsules that can be sent to others, who will wait until a quantum computer is built to decrypt and read them.

Beyond the harms caused by quantum computing, responsible innovation also involves building technologies in such a way that users can adopt them with confidence. One particular area that struck me during my PhD was how cybersecurity concerns in classical computer science often only arose many years after the deployment of a technology, rather than being considered carefully during the development of the technology. This would never be acceptable in other engineering projects such as in the construction of a railroad or a bridge. It led me to wonder what cybersecurity vulnerabilities could be anticipated in quantum computer systems, so that they can be designed with security in mind right from the beginning. In another work that is not included in this thesis, I have identified a side-channel vulnerability [3] that is likely to arise in fault-tolerant quantum computers, because the data that is sent to a quantum error correction decoder system, contains enough information to reconstruct the circuit or algorithm being run on the quantum computer.

I will continue working on the accessibility, uses, misuses and abuses of quantum computers in my post-doctoral research.

Acknowledgements

To everyone who supported me and wished me well along this journey — thank you! I would like to give two people special mentions.

I am grateful to my PhD supervisor, Dan Browne, for his unconditional, generous and patient supervision over the past four years. He created an environment in which I could develop the curiosities and talents that came to me naturally. From Dan, I have learnt that it is possible to be incredibly effective while being gentle.

I am grateful to Jayne Thompson for her mentorship and support throughout my PhD, even though her formal obligations to supervise me ended after the first year. I am especially grateful for her regular check-ins, her many invitations to events, and for hosting me for a research visit in Singapore.

Statement of contribution

All of the material in this thesis was primarily developed by me, but I have benefited from discussions with mentors and collaborators.

I am grateful to Stergios Koutsioumpas for introducing me to the power of code automorphisms and for discussions that led to the writing of Chapter 4.

I am grateful to Dan Browne for reviewing all my work in the past four years, and providing helpful feedback. I am grateful to Shin Nishio, Jayne Thompson, Jansen Zhikuan Zhao and Joseph Fitzsimons for many helpful discussions.

Contents

1 Introduction **14**

1.1 Motivation for quantum data structures 14

1.2 What is a data structure? 16

1.3 Outline of Thesis 20

2 A data structure perspective on quantum algorithms **22**

2.1 Quantum information as a data structure 22

2.2 Quantum state equality 28

2.3 Loading classical data onto quantum computers 31

2.4 Quantum decision search 33

2.4.1 Computational Tasks 34

2.4.2 Classical Algorithm 35

2.4.3 Quantum Algorithms 36

2.4.4 Relationship to Deutsch-Jozsa algorithm 41

2.5 Axiomatic approach to quantum search 42

2.5.1 Using two queries 42

2.5.2 Axiomatic characterisation of two query search 44

2.6 Quantum Phase Estimation explained through data structures 46

2.6.1 Data structures for phase estimation 47

2.6.2 Quantum Phase Estimation 52

3 Theory of quantum data structures **57**

3.1 Limits on compressing data with quantum information 57

3.1.1 Space advantages with weaker requirements 59
3.2 Quantum Fingerprinting 60
3.2.1 Data structure 61
3.2.2 Advantage over classical data structures 62
3.3 Graph state data structure 64
3.4 Quantum Counter 66
3.4.1 Constructions of quantum counters 68
3.5 Other directions in quantum data structures 72
3.5.1 Bounds on the maximum space advantage in unrestricted settings 72
3.5.2 Streaming algorithms 72
3.5.3 Finite automata 73
3.5.4 Resource advantages in communication complexity 74
3.5.5 Time advantages with quantum data structures 74
3.5.6 Quantum Random Access Memory 75
3.5.7 Classical data structures in superposition 75

4 Quantum Data Structures from Error Correcting Codes 77
4.1 Background on Error Correction 77
4.1.1 Parameters of a Code 78
4.1.2 Families of Codes 79
4.1.3 Code Equivalence and Automorphisms 80
4.1.4 Example: Automorphism in a Reed–Muller Code 81
4.2 Data structure framework 83
4.2.1 Data structure values 83
4.2.2 Data structure operations 85
4.2.3 Requirements on Error-Correcting Codes 90
4.3 Data Structures 93
4.3.1 Bitstrings with XOR (Binary linear codes) 93
4.3.2 Bitstrings with reverse, flip and append (Reed–Solomon) . . 96
4.3.3 Polynomials in point–value form (Reed–Solomon) 100

4.3.4 Graphs (Reed–Muller RM(2)) . . . 104
4.4 Discussion . . . 106

5 Quantum Data Structures from Hypergraph states 109
5.1 Encoding a list of bitstrings as a graph state . . . 109
5.1.1 Data structure encoding . . . 110
5.1.2 Data structure operations . . . 111
5.1.3 Discussion . . . 113
5.2 Hypergraph data structure . . . 115
5.2.1 Classical data structures . . . 115
5.2.2 Data structure definition . . . 117
5.3 A polynomial data structure from hypergraph states . . . 121
5.3.1 Polynomial Learning explained with data structures . . . 125
5.4 Discussion . . . 129

6 Algorithms using quantum data structures 130
6.1 Replacing the data structure in standard algorithms . . . 130
6.1.1 String Processing . . . 131
6.2 Message passing scenarios . . . 135
6.3 Memory checking . . . 138
6.4 Discussion . . . 139

7 Discussion 141
7.1 Further questions . . . 141
7.1.1 Readout operations beyond equality testing . . . 141
7.1.2 Practicality of quantum data structures . . . 142
7.1.3 Hybrid data structures . . . 143
7.1.4 Arbitrary group data structures . . . 144
7.1.5 Exploring generative AI as a tool for data structure and algorithm design . . . 145
7.2 Conclusion . . . 147

Appendices **149**

A Construction of Finite fields for Quantum Fingerprinting **149**

Bibliography **151**

List of Figures

2.1 False-negative probabilities of different algorithms for decision search 38
2.2 Circuit for quantum decision search algorithms 39
2.3 Deutsch–Jozsa circuit 41
2.4 Implementation of the phase oracle V_f using the bit-flipping oracle U_f 43
2.5 Illustration of point-value representation of polynomials 48
2.6 A subroutine within the quantum Fourier transform 53
2.7 Recursive definition of the quantum Fourier transform circuit 53
2.8 Circuit for the quantum Fourier transform 53
2.9 Quantum Phase Estimation algorithm 55

3.1 Simultaneous Message Passing model 64

5.1 Example of graph state list data structure 110
5.2 Example of a hypergraph 116
5.3 Example hypergraph state preparation circuit 117

6.1 A modified Simultaneous Message Passing scenario where one branch has two contributors 136
6.2 A modified Simultaneous Message Passing scenario where one branch has n contributors 137

List of Tables

1.1 Graph Abstract Data Type 16
1.2 Comparison of graph data structures 17
1.3 List Abstract Data Type (ListADT) 18
1.4 Stack Abstract Data Type (StackADT) 18

2.1 Quantum State Abstract Data Type (QSADT) 27
2.2 Function Abstract Data Type 32
2.3 Data structures for bit-valued functions 33
2.4 Data structures for polynomials 47

3.1 Fingerprint Abstract Data Type 61
3.2 Comparison of adjacency matrix and graph state as a data structure. 65
3.3 Graph state data structure 67
3.4 Counter Abstract Data Type Operations 67

4.1 Summary of the bitstring-with-XOR data structure. 93
4.2 Bitstring-with-XOR ADT. 94
4.3 Summary of the Reed–Solomon bitstring data structure. 96
4.4 Bitstring with reverse, flip and append ADT. 97
4.5 Point-value polynomial data structure. 100
4.6 Polynomial ADT. 101

5.1 Graph state list-of-bitstrings data structure. 113
5.2 Example space complexities for hypergraphs 119
5.3 Rank-k hypergraph state data structure 121
5.4 Multivariate polynomial over $\mathbb{F}_2$ data structure 126

Chapter 1

Introduction

1.1 Motivation for quantum data structures

Computers are information processing systems, which store and manipulate information in devices known as bits. Quantum computing is an emerging technology [4, 5] in which *quantum bits* or *qubits* are used to store and process information, instead of classical bits. Each classical bit can only be in one of two possible states (0 or 1), but a collection of n qubits can be in a state that requires 2^n complex numbers to describe. Quantum computing allows faster [6, 7], more space efficient [8] and more secure computations [9] than classical computing.

Quantum algorithms do not have an advantage over classical algorithms for all problems [10]. For a quantum computer to outperform classical computers on a specific task, an efficient quantum algorithm for that specific task has to be designed. Such an algorithm will leverage properties of quantum information (such as superposition, interference and entanglement) in a problem-specific way, to solve that problem more efficiently than is possible on classical computers. It can be hard to find such speedups, and it can also be hard to prove that the speedup holds over all possible classical algorithms and not just the best known classical algorithm.

Even when quantum computers do provide a speedup it is highly dependent on the details of the problem how large this speedup will be. A key challenge for the emerging quantum technology industry is the design of quantum algorithms. Relatively few are known, and even fewer achieve practically relevant speedups for

important problems in science and industry [11, 12]. The design of new quantum algorithms has been a difficult challenge for three decades since the inception of quantum computing as a research field [13, 10, 14].

In this thesis, we take a new approach to designing quantum algorithms. Most existing quantum algorithms are designed and reasoned about in low-level detail, either by describing them as a quantum circuit or by fully detailing and tracking the quantum states and unitary matrices that would be needed to implement the algorithm. In contrast, classical algorithms are often studied at a higher level of abstraction. Classical algorithms are rarely described in terms of bits and operations that individually read and flip the bits. Instead, *data types* are defined that model how a collection of bits and bit operations work together to provide a computer implementation of familiar mathematical objects. Defining data types helps encapsulate the details of recurring algorithmic techniques and allows the programmer to reason about more complicated algorithms that build upon these mathematical objects and the operations they support.

In this thesis we will also seek to define such high-level data types for quantum algorithms so that we may make it easier to reason about the data that quantum states represent, how this data can be transformed, and what information can be extracted from these states. We call this perspective *quantum data structures*, in analogy to the field of data structures in classical computing which studies data types and different schemes of how to represent, modify and extract information from them.

This choice of perspective also makes sense from a conceptual analysis of *algorithms*. Algorithms are defined as any automatable procedure that transforms some input information into some output information. For the automation to be possible, algorithms have to make assumptions on the format of the input information. For example, the long multiplication algorithm that one learns in school requires the numbers to be represented in a place value system [15]. It is not possible to perform long multiplication directly on Roman numerals or the written name of the number in English. Indeed before place value systems, multiplication algorithms took other

forms such as the Egyptian multiplication algorithm [16]. Place value systems were a data structure innovation that enabled the more efficient multiplication algorithm.

If every classical algorithm is making some assumptions about how its data is formatted, what assumptions are quantum algorithms making about their data? How should data be represented in quantum states for efficient storage and processing? What algorithms can we design using these quantum data structures? This thesis will make progress on these three questions.

We may summarise the motivation of this thesis as follows:

Premise 1: Quantum computing needs algorithms;

Premise 2: Algorithms need data structures.

Conclusion: Quantum computing needs data structures.

1.2 What is a data structure?

Function	Description	Formal Description
`create()`	Create an empty graph.	$(V,E) \leftarrow (\emptyset,\emptyset)$
`.add_vertex(v)`	Add a vertex with ID `v`.	$(V,E) \rightarrow (V \cup \{v\}, E)$
`.delete_vertex(v)`	Delete the vertex with ID `v`.	$(V,E) \rightarrow (V \setminus \{v\}, E \setminus \{e \in E \mid v \in e\})$
`.add_edge(u, v)`	Add an edge between vertices `u` and `v`.	$(V,E) \rightarrow (V, E \cup \{\{u,v\}\})$
`.delete_edge(u, v)`	Delete the edge between vertices `u` and `v`.	$(V,E) \rightarrow (V, E \setminus \{\{u,v\}\})$
`.is_edge(u, v)`	Check whether there is an edge between vertices `u` and `v`.	Return whether $(u,v) \in E$

Table 1.1: Graph Abstract Data Type

A data structure is a format of storing information together with a list of procedures for creating, modifying or accessing this information [17, 18]. Any data structure is defined to meet the requirements of a data type such as integer, list or graph. More precisely, an abstract data type (ADT) is a description of the requirements for a successful computer implementation of some data type, such as a mathematical object like a number or a graph, or other kinds of data such as

a text string or an image [17]. The abstract data type specifies the operations that need to be supported by the data structure. These operations are generally creation, insertion, deletion, comparison, composition, mutation and readout operations of various kinds. For example, a basic abstract data type for graphs might include operations as detailed in Table 1.1.

Abstract data types say nothing about how the specification they set is practically achieved. Data structures are implementations that meet the specification of an abstract data type. There are many data structures that can meet this specification. This diversity is significant because different data structures for the same data type can vary on their space complexity and time complexity for various operations. This gives us different options to choose from depending on the scenario in which we are applying the data type. To illustrate how one ADT can have multiple data structures, three examples of data structures for the Graph ADT are given in Table 1.2.

Operation	**Ordered Edge List**	**Adjacency Matrix**	**Adjacency Lists**
Space complexity	$O(V+E)$ space	$O(V^2)$ space	$O(V+E)$ space
`add_vertex(v)`	$O(1)$	$O(V^2)$	$O(1)$
`delete_vertex(v)`	$O(\log E)$	$O(V^2)$	$O(V+E)$
`add_edge(u, v)`	$O(\log E)$	$O(1)$	$O(1)$
`delete_edge(u, v)`	$O(\log E)$	$O(1)$	$O(\text{degree}(u))$
`is_edge(u, v)`	$O(\log E)$	$O(1)$	$O(\text{degree}(u))$

Table 1.2: Comparison of graph data structures.

Most discussions of this kind in classical computing focus on abstract data types that support a "universal" set of operations. In the Graph example presented above, the abstract data type has the basic set operations from which it is possible to implement any other operation one might want to perform on the graph. In this case, the universality comes from the fact that the data type requires the ability to perform arbitrary insertions, deletions and lookups on both the vertex and edge set. While there's no precise definition of this concept, a *universal* data structure is characterised by supporting arbitrary read and write operations on its internal representation, and unambiguously representing each possible value of the data structure by a unique internal representation.

Function	Description	Formal Description
`create()`	Create an empty list.	$L \leftarrow []$
`.insert(i, x)`	Insert element x at index i.	$L \rightarrow (L[0], \ldots, L[i-1], x, L[i], \ldots, L[n-1])$
`.delete(i)`	Remove the element at index i.	$L \rightarrow (L[0], \ldots, L[i-1], L[i+1], \ldots, L[n-1])$
`.get(i)`	Return the element at index i.	Return $L[i]$
`.set(i, x)`	Update the element at index i to x.	$L[i] \leftarrow x$
`.size()`	Return the number of elements in the list.	Return $\|L\|$

Table 1.3: List Abstract Data Type (ListADT).

Function	Description	Formal Description
`create()`	Create an empty stack.	$S \leftarrow []$
`push(x)`	Push element x onto the top of the stack.	$S \rightarrow (S[0], \ldots, S[n-1], x)$
`pop()`	Remove the top element.	$S \rightarrow (S[0], \ldots, S[n-2])$
`peek()`	Return the top element.	Return $S[n-1]$
`size()`	Return the number of elements in the stack.	Return $\|S\|$

Table 1.4: Stack Abstract Data Type (StackADT).

It can be useful to study non-universal data structures for two reasons. Firstly, limiting the set of operations can help programmers by better highlighting a particular algorithmic strategy that the data structure supports and make programs easier to reason about. Table 1.3 shows a universal ListADT and Table 1.4 shows the non-universal StackADT which is defined by removing certain capabilities from ListADT. The set of operations in StackADT are all either directly available in ListADT, or they can be simulated by using the universal operations in the right way. For example, pushing onto a Stack can be simulated by inserting into the last position in a List. However, StackADT does not allow all of the operations available in ListADT. For example, it does not allow the user to input an element into the middle of the Stack. An omniscient programmer would not require the special definition of StackADT as they could always use a List like a Stack. However, the use of StackADT serves many practical benefits - it can nudge programmers towards *first-in-last-out* as a

problem solving strategy, it can make programs easier to reason about, and it can help ensure that the constraints of a Stack are enforced in the program and its execution.

Secondly, in some situations it may be known in advance that a data type will be used in a particular way. For example, it may be known that a counter data type will not be used to check the exact count, but only to get an order-of-magnitude estimate of the total count. Furthermore, in such settings, some small probability of failure can usually be tolerated, since access operations on the data structure will only be performed one or a few times. In such cases, a more efficient data structure can be usually defined that only supports those features, keeps only as much information as is required to answer those questions.

An introductory example of a data structure that exploits this principle is the *Morris counter* [19]. A Morris counter is a data structure that supports two operations: incrementing a counter, and querying its approximate value. Rather than storing the exact count n, a Morris counter stores only an exponent k such that 2^k is an estimate of n. Each increment operation increases k by one with probability $\frac{1}{2^k}$, so that the expected value of 2^k tracks n accurately. This representation requires only $O(\log\log n)$ bits, compared to the $O(\log n)$ bits required to store the exact count. The cost of this space saving is that the data structure can no longer return the exact count, only an approximation, and it cannot support operations such as decrementing or comparing two counters exactly. By restricting the supported operations in this way, the Morris counter achieves an exponential improvement in space at the cost of giving up universality.

For related reasons which will be elaborated in later chapters, the quantum data structures we discuss in this thesis will not be universal data structures. For example, quantum fingerprinting [8] is a technique to encode a bitstring into a quantum state on a very small number of qubits. However, once the bitstring has been encoded, the only kind of readout operation that can be performed on the bitstring is to check for equality with another bitstring. It will not be possible to retrieve a particular bit from the bitstring.

Note that "quantum data structures" may also refer to other topics such as the study of classical data structures that represent objects of interest in quantum science (such as tensor networks [20], or stabilizer tableaus [21]), or the study of data structures that are implemented when they arise as subroutines within quantum computations (such as in the case of quantum random access memory [22], or quantum walks over data structures [23]). While these are interesting and highly important areas of research as well, they are not the focus of this thesis. This thesis focusses on *inherently* quantum data structures, where the data storage format and operations are inherently quantum, rather than classical data structures for quantum science or coherent versions of classical data structures.

1.3 Outline of Thesis

This chapter has introduced the concepts of abstract data types and data structures, as well as set up the key research question for this thesis, namely the design of quantum data structures. The rest of this thesis is structured as follows:

- **Chapter 2** contains novel formulations of textbook quantum algorithms in terms of quantum data structures. This shows that data structures are a useful perspective for understanding quantum algorithms at a conceptual level.
- **Chapter 3** collects disparate strands of literature together into a review of prior work relating to quantum data structures. It also presents the main background material on quantum fingerprints and graph states which will be developed in this thesis.
- **Chapter 4** develops new quantum data structures through a framework based on classical error correcting codes and their automorphisms.
- **Chapter 5** develops new quantum data structures using graph and hypergraph states.
- **Chapter 6** explores the computational settings in which the various presented data structures can be used, and the conditions under which they achieve a quantum advantage over classical methods.

- **Chapter 7** evaluates the findings of the thesis and charts a plan for further research in quantum data structures.

Chapter 2

A data structure perspective on quantum algorithms

This chapter presents quantum computing and quantum algorithms from a data structures perspective, and argues that thinking in terms of quantum data structures can make existing quantum algorithms easier to understand and work with. Section 2.1 lays out the foundational concepts of quantum information, framing quantum registers as an abstract data type. Section 2.2 introduces the subroutine of ”quantum state equality”, a key operation supported by most of the data structures in this thesis. Section 2.3 discusses the oracle access paradigm for loading classical data into quantum computers. Section 2.4 presents a simple search problem, which we argue offers a gentler presentation to the Deutsch-Jozsa problem and algorithm, and whose explanation is conceptually led by a data structure rather than a long algebraic derivation. Section 2.5 builds on this framework to give an axiomatic approach motivating Grover's search algorithm. Finally, Section 2.6 explains the Quantum Phase Estimation algorithm in terms of data structures for polynomials, again avoiding a long algebraic derivation.

2.1 Quantum information as a data structure

Quantum bits (qubits) are devices that store quantum information. Collections of qubits act as the memory of quantum computers. The features of qubits such as what sort of information they store, how they can be modified and how classical

information can be extracted from them are typically presented as the physical postulates describing quantum mechanics in finite-dimensional systems.

To set up the theme of this thesis, we will instead frame quantum memories as data structures that implement a specific abstract data type (ADT) called the quantum state abstract data type. This perspective allows a classical computer scientist to abstract the details of physics and engineering, and simply view quantum computers as regular computers equipped with additional powerful data structures that can be implemented on a hardware level through the principles of quantum mechanics. The most general of these data structures, which we cover next, describes what qubits can do. Over the course of the rest of this thesis, we will focus on specific sets of quantum states and operations, allowing us to define more specialised data structures for specific tasks. We find that this data structure approach gives a new perspective on where quantum computation derives its power, and new directions in seeking quantum advantage.

The quantum state abstract data type (QSADT) has the following specification:

1. **Data Storage:** The possible states of a quantum memory (typically denoted with a variable name surrounded by a ket, as $|\psi\rangle$) correspond to vectors of complex numbers. For some fixed size of QSADT, given by the dimension d,

$$\text{provisionally we can write, } |\psi\rangle \in \mathbb{C}^d. \tag{2.1}$$

However, there is an extra detail here: the data type does not distinguish vectors that are equal up to a scalar. Instead, the possible states of the data type are equivalence classes of vectors differing only by a non-zero complex scalar factor. Thus, two vectors

$$|\psi\rangle \quad \text{and} \quad \alpha|\psi\rangle, \tag{2.2}$$

where $\alpha \in \mathbb{C} \setminus \{0\}$, are associated with the same quantum state. Thus to be geometrically precise, quantum states correspond not to vectors, but to lines passing through the origin in complex vector space. Algebraically, the set of quantum states is the

complex projective space, and so we write:

$$|\psi\rangle \in \mathbb{P}(\mathbb{C}^d). \tag{2.3}$$

Alternatively, as is more common, we can focus only on the normalised vectors:

$$|\psi\rangle \in \mathbb{C}^d \text{ such that } \langle\psi|\psi\rangle = 1. \tag{2.4}$$

This extra detail on normalisation might seem strange to a beginner who would have hoped for just a data structure for a vector. Here we can momentarily break the abstraction to remind them that quantum information and its laws were discovered in nature and not designed by people. The mathematical structure of quantum mechanics emerged from physical observations rather than from computational convenience. However, to a first approximation and in most colloquial discussions, we do simply think of quantum states as vectors.

2. **Data Modification:** Quantum states are vectors that can be modified by linear maps, but there are caveats about which operations can be applied efficiently and with a high probability of success.

The most friendly are the invertible linear maps, which can be applied deterministically. If we follow convention and focus on quantum states as normalised vectors, then the set of invertible linear maps that keep the vectors normalised are precisely the unitary transformations, U satisfying:

$$U^\dagger U = UU^\dagger = I. \tag{2.5}$$

Under unitary evolution, a quantum state $|\psi\rangle$ transforms according to:

$$|\psi\rangle \to U|\psi\rangle \tag{2.6}$$

where the normalization is preserved since $\langle U\psi|U\psi\rangle = \langle\psi|U^\dagger U|\psi\rangle = \langle\psi|\psi\rangle = 1$.

While the set of all unitary transformations forms a continuous group, not all unitaries can be efficiently realized in practice. Quantum computation typically relies

on finite gate sets of elementary unitary operations that can be implemented by the model of computation in a unit time step. Universal gate sets, such as the combination of single-qubit rotations and the CNOT gate, can approximate any unitary transformation to arbitrary precision. However, most unitaries require exponentially many elementary gates to implement accurately, thus for time-efficient quantum algorithms, one must only use those unitaries that can be efficiently decomposed into elementary gates.

When the desired linear map is not invertible, the situation becomes more restrictive. Such operations can only succeed with some probability of error. When a linear map A is applied to a quantum state $|\psi\rangle$, the probability of successful application is $p_{\text{succ}} = \frac{1}{\|A\|^2}\langle\psi|A^\dagger A|\psi\rangle$, where $\|A\| = \sqrt{\lambda_{\max}(A^\dagger A)}$ is the largest eigenvalue of $A^\dagger A$, and after application the quantum state is renormalised to be in the state $\frac{1}{\sqrt{p_{\text{succ}}}}A|\psi\rangle$.

The above transformations are special cases of the general framework of quantum channels. The complete mathematical description of physically realizable quantum operations is provided by quantum channels, which can be expressed through sets of measurement operators $\{K_i\}$ satisfying the completeness relation $\sum_i K_i^\dagger K_i = I$. When a quantum channel acts on a state $|\psi\rangle$, the system evolves probabilistically: with probability $p_i = \langle\psi|K_i^\dagger K_i|\psi\rangle$, the system transitions to the normalized state $\frac{K_i|\psi\rangle}{\sqrt{p_i}}$.

3. **Data Retrieval:**

The most unusual aspect of the quantum state ADT is how its data readout operations work. In contrast to classical data structures, simply reading out the stored vector components is impossible. Instead, information stored within quantum states can only be accessed by sampling from a probability distribution that is defined in terms of the quantum state.

Another peculiarity of quantum information is that after measurements, the quantum state is altered. Unlike classical data that can be copied and accessed repeatedly without modification, quantum information is changed each time it is read, and notably some of the stored information is lost upon measurement.

The simplest kind of retrieval operation is computational basis measurement, which allows for sampling from a probability distribution defined in terms of the quantum state's vector components (known as *amplitudes*). Performing this measurement destroys the quantum state. When measuring the quantum memory in the computational basis, we obtain index $i \in \{0, 1, \ldots, 2^n - 1\}$ as the result of the measurement with probability:

$$p(i) = |\psi_i|^2, \tag{2.7}$$

where ψ_i is the i^{th} component of vector $|\psi\rangle$. This is known as *Born's rule*.

More generally, quantum measurement follows the quantum channel framework described above. By carefully choosing sets of measurement operators $\{M_k\}_k$ such that each outcome corresponds to a meaningful result, more sophisticated probability distributions over different kinds of outcomes can be created.

The above features of QSADT are summarised in Table 2.1. One could implement a data structure for this abstract data type on a classical computer. However, if one stores each complex number explicitly, with classical data storage technologies that we use today, we would only use at most a few petabytes of information for one quantum state and this would limit us to quantum states of dimension roughly 10^{17}.

The key postulate of quantum information is that a device called a quantum computer can be built that realises the QSADT using only n computational elements (called qubits) to implement a 2^n-dimensional QSADT. This exponential scaling represents one of the most remarkable features of quantum computation: a relatively small number of qubits can store and manipulate an exponentially large amount of information. For instance, just 300 qubits can theoretically represent more complex numbers than there are atoms in the observable universe ($2^{300} \approx 10^{90}$). This exponential resource scaling is what enables quantum algorithms to potentially solve certain problems exponentially faster than classical computers. Quantum systems are in general extremely difficult to simulate with classical computers because the memory

requirement to represent all the complex numbers quickly becomes prohibitive for even tens of qubits.

Viewed through the lens of abstract data structures, quantum computing can be appreciated as computation on a naturally discovered, physically implemented and remarkably efficient data structure. This framing positions quantum computation as part of familiar concepts and paradigms in classical computing. Although this data structure seems unfamiliar and restrictive, it is what allows all the dramatic speedups associated with quantum computing. For instance, algorithms leveraging precisely this structure, such as Shor's algorithm [7], demonstrate exponential speed-ups over the best-known classical algorithms for problems like integer factorization. In the data structures that follow in this thesis, we present more specialised quantum data structures through their abstract data types allowing algorithm design without the need to understand the underlying principles of quantum information.

Function	**Description**
`init(`$\mathbf{z}$`)`	Store a non-zero complex vector $\mathbf{z} \in \mathbb{C}^{2^n}$ as a quantum state with the equivalence rule $\mathbf{z} \sim \alpha\mathbf{z}$ for all $\alpha \in \mathbb{C} \setminus \{0\}$. Commonly normalized such that $\langle\psi|\psi\rangle = 1$.
`apply_unitary(`U`)`	Evolve the state deterministically by a unitary transformation U satisfying $U^\dagger U = UU^\dagger = I$. The state transforms as $\|\psi\rangle \to U\|\psi\rangle$ with normalization preserved.
`apply_non_unitary(`A`)`	Apply a non-invertible linear map A to the state. Returns a boolean indicating success/failure. Success probability is $p_{\text{succ}} = \frac{1}{\|A\|^2}\langle\psi|A^\dagger A|\psi\rangle$. On success, the state becomes $\frac{1}{\sqrt{p_{\text{succ}}}}A|\psi\rangle$.
`compose(`$\|\psi_1\rangle, \|\psi_2\rangle$`)`	Form the tensor product state $\|\psi_1\rangle \otimes \|\psi_2\rangle$, creating a composite quantum register of dimension $2^{n_1+n_2}$ from individual n_1 and n_2 qubit states.
`sample()`	Perform computational basis measurement, returning index $i \in \{0, 1, \ldots, 2^n - 1\}$ with probability $p(i) = \|\psi_i\|^2$. The quantum state is destroyed after measurement.
`channel(`$\{K_i\}$`)`	Apply a general quantum channel using measurement operators $\{K_i\}$ satisfying $\sum_i K_i^\dagger K_i = I$. Returns measurement outcome i with probability $p_i = \langle\psi|K_i^\dagger K_i|\psi\rangle$, and the state becomes $\frac{K_i|\psi\rangle}{\sqrt{p_i}}$.

Table 2.1: Quantum State Abstract Data Type (QSADT).

2.2 Quantum state equality

The Quantum State Abstract Data Type (QSADT) describes the full generality of what quantum states can represent and how they can be manipulated. In principle, it allows us to model any process or transformation that a quantum computer can perform on quantum data. However, this level of generality can be overwhelming. While it is theoretically complete, it is often too broad to provide practical guidance on how to actually design quantum algorithms. To make progress, it is helpful to focus on specific restricted families of operations that are especially useful for solving particular classes of problems.

We begin with one of the most basic yet essential primitives: the quantum analogue of checking whether two pieces of data are equal. In classical computing, equality testing is straightforward. Given two bit strings, we simply compare them bit-by-bit until we find a mismatch or verify that they are identical. The quantum setting is different because of the linearity of quantum mechanics and the destructive nature of quantum measurement. Nevertheless, it is still possible to devise a procedure that, with some probability of error, answers the question: *Is the given quantum state equal to a target state?*

Problem statement

The *Quantum State Equality* (QSE) problem can be stated as follows:

We are given (1) a quantum register prepared in an unknown state $|\psi\rangle$, and (2) a classical description of a known quantum state $|\phi\rangle$. Our task is to determine whether $|\psi\rangle = |\phi\rangle$.

Here, "classical description" could come in various forms that give us the ability to determine any fact about the state, for example we could be given all the amplitudes of $|\phi\rangle$ in the computational basis, or we could be given a small computer program that given an index i, outputs the i-th amplitude of $|\phi\rangle$. Note that it means that we do not have an actual physical copy of $|\phi\rangle$ to use as a resource, but since we know everything about the state, we could prepare the state if we wanted to, or we could construct operators that depend on its amplitudes.

Solution via projective measurement

A natural approach to solving the QSE problem is to perform a projective measurement designed to detect the state $|\phi\rangle$. This involves the two-outcome measurement $\{M_\phi, I - M_\phi\}$, where:

$$M_\phi = |\phi\rangle\langle\phi|$$

is the projector onto $|\phi\rangle$, and I is the identity operator on the Hilbert space of the system.

The meaning of this measurement is as follows:

- If the outcome corresponding to M_ϕ occurs, the post-measurement state of the system is guaranteed to be $|\phi\rangle$.
- If the outcome corresponding to $I - M_\phi$ occurs, the post-measurement state is orthogonal to $|\phi\rangle$.

Because we have a classical description of $|\phi\rangle$, we can implement this measurement explicitly. This can be done by measuring $|\psi\rangle$ in any basis including $|\phi\rangle$. A straightforward way to do this is to apply $U^\dagger_{|\phi\rangle}$ onto $|\psi\rangle$ and then measure on the computational basis, where $U_{|\phi\rangle}$ denotes any circuit that prepares $|\phi\rangle$ (i.e. $U_{|\phi\rangle}|0\rangle = |\phi\rangle$). If the output is $|0\rangle$ then QSE returns YES, otherwise it returns NO.

Correctness and error probability

The correctness of the procedure follows directly from the postulates of quantum information. The probability of obtaining the M_ϕ outcome is:

$$\begin{aligned} p_\phi &= \langle\psi|M_\phi|\psi\rangle \\ &= \langle\psi|\left(|\phi\rangle\langle\phi|\right)|\psi\rangle \\ &= \left(\langle\psi|\phi\rangle\right)\left(\langle\phi|\psi\rangle\right) \\ &= \langle\phi|\psi\rangle^*\langle\phi|\psi\rangle \\ &= |\langle\phi|\psi\rangle|^2. \end{aligned}$$

We can distinguish two main cases:

1. If $|\psi\rangle = |\phi\rangle$ exactly, then $\langle\phi|\psi\rangle = 1$ and therefore $p_\phi = 1$. The measurement will always return the M_ϕ outcome.

2. If $|\psi\rangle \neq |\phi\rangle$, then $0 \leq |\langle\phi|\psi\rangle|^2 < 1$. In this case:

   - If the $I - M_\phi$ outcome occurs, the post-measurement state is orthogonal to $|\phi\rangle$, and we know with certainty that $|\psi\rangle \neq |\phi\rangle$.
   - If the M_ϕ outcome occurs, we cannot be sure whether $|\psi\rangle$ and $|\phi\rangle$ were identical. This is because even two distinct states can have a large inner product, leading to a significant probability of obtaining this outcome by chance.

Therefore, this procedure has *one-sided error*. If the states are equal, the test will never incorrectly declare them to be different. However, if the states are different, the test can still incorrectly produce the M_ϕ outcome, leading to an inconclusive result. The probability of this inconclusive case is exactly $|\langle\phi|\psi\rangle|^2$.

Comparison with classical equality testing

Quantum state equality testing differs from its classical analogue in two key respects:

1. **Probabilistic outcomes.** In a classical equality check, the result is deterministic. Given two classical bit strings, a direct comparison yields an unambiguous answer. In the quantum case, measurement outcomes follow a probability distribution determined by the overlap between the two states. As a result, the quantum test can make conclusive negative declarations ($\neq$) but positive declarations ($=$) are subject to an error probability.

2. **Destructive measurement.** Classical data can be compared without altering the original data, and the process can be repeated as many times as needed. In quantum mechanics, the act of measurement generally changes the state. After our equality test:

   - If the M_ϕ outcome occurs, the system is left in the state $|\phi\rangle$, regardless of what $|\psi\rangle$ was before.

- If the $I - M_\phi$ outcome occurs, the $|\phi\rangle$ component is completely removed from the system's state, making it impossible to reconstruct the original $|\psi\rangle$.

These two properties are not peculiar to this problem but are characteristic of many operations in quantum computing. They have important implications for algorithm design, particularly in the context of quantum data structures. In later sections, we will see how one-sided error can be mitigated by repeated tests, and how state destruction can be managed by carefully structuring the computation so that measurements occur only when their consequences are acceptable.

2.3 Loading classical data onto quantum computers

Early quantum algorithms were motivated by quantum parallelism - the idea that a quantum computer could explore multiple computational branches in superposition to arrive at results more efficiently. In order to formalise how a quantum computer would explore branches in parallel, the pioneers of quantum computing defined computational models in which the input data set could be probed in superposition.

An *oracle* is a mathematical model of how data can be accessed in a computational process. Formally, an oracle implements a unitary operator U_f that acts on quantum states according to:

$$U_f : |x\rangle \otimes |y\rangle \mapsto |x\rangle \otimes |y \oplus f(x)\rangle$$

where $f : \{0,1\}^n \to \{0,1\}^m$ is the function encoded by the oracle, and $\oplus$ denotes addition modulo 2. This allows quantum algorithms to query f coherently on superposition states $\sum_x \alpha_x |x\rangle$, enabling simultaneous evaluation of f on multiple inputs.

It is important to understand how oracular problems are a useful formal mathematical notion. A common misconception is that oracles are theoretical constructs with little bearing on practice. In fact, oracles can model two realistic computational scenarios. First, in database access scenarios, where $f(x)$ returns the value stored

at index x in a database, the oracle can be implemented through QRAM (Quantum Random Access Memory), which provides coherent access to classically stored data via quantum addressing mechanisms. Second, in function evaluation scenarios, where f represents some mathematical function with efficient classical computation (such as modular arithmetic in Shor's algorithm), the oracle U_f can be implemented as a polynomial-size quantum circuit. In both cases, the oracle abstraction separates the complexity of accessing information from the complexity of processing it algorithmically.

One common use of an oracle is to prepare an encoding of a function in a quantum state,

$$\frac{1}{\sqrt{|D|}} \sum_{x \in D} |x\rangle |f(x)\rangle,$$

which places the values of f in superposition across all possible inputs. This idea is used in many quantum algorithms such as the Deutsch-Jozsa algorithm [24] and Simon's algorithm [25]. In this view, the oracle acts as a data-loading subroutine that embeds classical information about f into a quantum state which will then be used in subsequent algorithmic steps to operate on all entries in parallel.

We can treat this state as a *data structure* for a function ADT, as defined in Table 2.2. The operation `create` initialises the quantum representation of a function, given information about the domain of the function and the oracle that returns the values of the function. The `equals` operation performs an equality check between functions relying on the QSE subroutine of the previous section.

Function	**Description**	**Formal Description**
`create(D, O)`	Create a function data structure for a function f defined on domain D, using oracle O for evaluation.	Store the set $\{(x, f(x)) \mid x \in D\}$ representing the mapping $f\colon D \to \mathcal{R}$.
`.equals(f2)`	Test whether the stored function f equals another function f_2. The procedure destroys the data structure after execution.	Return "$f = f_2$" if $\{(x, f(x)) \mid x \in D\} = \{(x, f_2(x)) \mid x \in D\}$, otherwise return "$f \neq f_2$" with some probability of failure. Afterwards, the stored set $\{(x, f(x))\}$ is erased.

Table 2.2: Function Abstract Data Type.

This abstraction has multiple advantages: it allows us to articulate quantum algorithms using higher level concepts, it facilitates a comparison to classical algorithms and it highlights specific capabilities of quantum information which can be adapted for solving new problems. For example, consider different ways we may encode a bit-valued function, as shown in Table 2.3. The bit and sign encodings are common from various oracular quantum algorithms. Note that the angle encoding used here is not the same as the one used in variational quantum algorithms. We are not aware of the angle encoding appearing in other works, but it was inspired by Ref. [26]. These data structures also differ in performance, specifically their probability of failure on the equality test operation and marginally on the number of qubits used. In the next section we will see how having all these different data structures is useful and how they can be prepared given access to U_f.

Name	**Encoding**	**FNP**
Bit encoding	$\frac{1}{2^{n/2}}\sum_x \lvert x\rangle\lvert f(x)\rangle$	$(1-r)^2$
Sign encoding	$\frac{1}{2^{n/2}}\sum_x (-1)^{f(x)}\lvert x\rangle$	$(1-2r)^2$
Angle encoding	$\frac{1}{2^{n/2}}\sum_x \lvert x\rangle \otimes [X^{f(x)}R_y(\theta)\lvert 0\rangle]$	$(1-r+r\sin\theta)^2$

Table 2.3: Data structures for bit-valued functions. FNP stands for False Negative Probability and shows the probability of incorrectly outputting that two functions are the same when they are actually different, in terms of a quantity r, which is the ratio of inputs on which the two functions give different outputs.

2.4 Quantum decision search

This section presents a simple search problem and a quantum algorithm for it that uses the quantum data structures for functions. We focus on the *decision version* of search, in which, given access to a function f one has to determine whether an x exists such that $f(x) = 1$ (i.e. deciding whether a function is constant-zero or *non-constant*).

We first explain the optimal classical algorithm for this problem. Then we explain quantum algorithms in which a state that encodes the function f is prepared, and this state is compared for equality with a reference state that encodes the constant-

zero function. These algorithms only make false negative errors so we analyse their performance using false-negative probability (i.e. probability of outputting that the function is constant-zero when it is actually non-constant). The first of these quantum algorithms is a natural quantum extension of the classical algorithm. The second quantum algorithm is a simplified treatment of the generalised Deutsch-Jozsa algorithm [26].

The contribution of this section is to show how these algorithms can be effectively motivated and taught through high-level algorithmic concepts (namely, *encoding* a function and *equality testing*). These concepts are particularly friendly to students with a computer science background. This presentation also motivates the axiomatic approach to quantum search in the next section. Moreover, our presentation reveals that the Hadamard transform is not essential to the Deutsch-Jozsa algorithm contrary to how it is categorised as an algorithm based on the "Fourier transform" in Nielsen and Chuang [5] (section 1.4.5).

I have used this presentation in a quantum algorithms course that I gave at the UCL Centre for Doctoral Training in Quantum Computation and Communication [27].

2.4.1 Computational Tasks

Instead of starting with the Deutsch-Jozsa problem, we propose starting quantum computing courses with the following more natural problem:

Problem 1 (Decision Search) *Given access to an oracle for a function* $f : \{0,1\}^n \mapsto \{0,1\}$*, determine whether there exists an input* x *such that* $f(x) = 1$.

As usual, the quantum version of this problem is defined in terms of a quantum oracle:

Problem 2 (Quantum Decision Search) *Given access to a quantum oracle* U_f *that implements a function* $f : \{0,1\}^n \mapsto \{0,1\}$ *via the transformation* $U_f : |x\rangle|z\rangle \mapsto |x\rangle|z \oplus f(x)\rangle$*, determine whether there exists an input* x *such that* $f(x) = 1$.

We will first focus on solving this problem given only a single query to either the function f or the quantum oracle U_f. We define the quantity $r_f = \frac{|f^{-1}(1)|}{2^n}$ as the ratio

of inputs that the function f maps to 1, or alternatively interpreted as the average of the function's output values. It is worth noting that $|f^{-1}(0)| + |f^{-1}(1)| = 2^n$. These quantities appear in the analysis of the failure probability of the algorithms.

We will also consider a variant of the above problems in which it is promised that if the input is a YES-instance (i.e. a non-constant function) then it has an average of R, where R is a quantity known in advance and satisfies $R \geq 0.5$. Stated mathematically, the promise is that $r_f \in \{0, R\}$ for some $R \geq 0.5$.

The algorithms we consider will always output **NO** when the input is a NO-instance (i.e. when $f(x) = 0$ for all x). We focus our analysis on the *false-negative probability*, which is the probability of incorrectly outputting **NO** when the input is actually a YES-instance.

2.4.2 Classical Algorithm

If we are allowed only a single classical query, since there is no structure to exploit in the black-box function f, the simple and optimal strategy (Algorithm 1) is to evaluate the function on a random element y and output **YES** iff $f(y) = 1$. If the input is a NO-instance, then our algorithm always outputs **NO** correctly. Otherwise, if the input function is a YES-instance, then the probability of failing to detect this from the random query is $1 - r_f$. Note that there is no way to solve this problem with lower false negative probability if we are promised that $r_f \in \{0, R\}$. We will see that quantum algorithms can exploit such a promise.

Algorithm 1: Decision Search Algorithm (one query)

Output: YES if the sampled point is a witness, otherwise **NO**
Randomly sample y;
if $f(y) = 0$ **then**
  output NO;
end
else
  output YES;
end

2.4.3 Quantum Algorithms

We present two quantum algorithms for this problem. Algorithm 2 is a quantum analogue of the random sampling algorithm above and it achieves a quantum advantage by having a smaller false-negative probability on all possible YES-instance input functions f. Algorithm 3 is an extension which can be tuned to have zero error probability when it is known in advance that $r_f \in \{0, R\}$ for some $R \geq 0.5$.

Lower false negative probability on all inputs

The classical algorithm in the previous section samples a random input to the function. A natural quantum extension of this is to query the oracle in superposition over all inputs. Doing this prepares the *bit encoding* of the function, which is the state $|f\rangle = \frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} |x\rangle|f(x)\rangle$. To check whether f is a NO-instance, we check that this state is the same as the state encoding the constant zero function, $|c_0\rangle = \frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} |x\rangle|0\rangle$. If they are the same, our algorithm will always detect this, otherwise it will have some failure probability.

When comparing the prepared state $|f\rangle$ with $|c_0\rangle$ for equality the probability of mistaking them to be equal when they are different is:

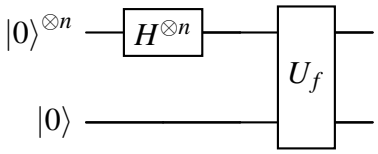
$$
\begin{aligned}
&|\langle c_0|f\rangle|^2 \\
&= \left| \left(\frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} \langle x|\langle 0| \right) \left(\frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} |x\rangle|f(x)\rangle \right) \right|^2 \\
&= \left| \frac{1}{2^n} \sum_{x\in\{0,1\}^n} \langle 0|f(x)\rangle \right|^2 \\
&= \left| \frac{1}{2^n} \left(|f^{-1}(0)| \cdot 1 + |f^{-1}(1)| \cdot 0 \right) \right|^2 \\
&= (1 - r_f)^2. \qquad (2.8)
\end{aligned}
$$

This straightforward approach (Algorithm 2) achieves a quantum advantage on all inputs as it has a false-negative probability $(1 - r_f)^2$, which is less than the false-negative probability of $1 - r_f$ achieved by Algorithm 1. Interestingly, the "square" in

Born's rule is what provides this quantum advantage. One quantum query attains the square of the failure probability of a classical query, or alternatively, two independent classical queries are required to match the success probability of one quantum query.

Algorithm 2: Quantum Decision Search Algorithm (one query)

Output: YES if the oracle is non-zero on some input, otherwise **NO**
Prepare the quantum state $|f\rangle$ using the circuit:



if $|f\rangle \underset{QSE}{=} |c_0\rangle$ **then**
  output NO
end
else
  output YES
end

Fig. 2.1 compares this quantum algorithm with the classical algorithm and other quantum algorithms we will discuss. The plot shows how the false-negative probability (probability of a YES-instance being mistaken as NO-instance) varies with r_f, the ratio of inputs on which the function outputs 1.

There is an interesting way to see the quantum state as providing two "attempts" at solving the decision problem. The first measurement is performed on the second register to hope to see the value 1. If this happens successfully, we are done. However, if we get the outcome 0, we have projected the first register into a subspace where only the NO-inputs remain. We can now perform a quantum state equality test between a uniform superposition state and this remaining state.

Zero failure probability with a promise

In this section we present the version of decision search in which it is known in advance that $r_f \in \{0, R\}$ for any $R \geq 0.5$ and present an algorithm [26] that uses this promise to succeed in one query without any probability of error. Even if this promise

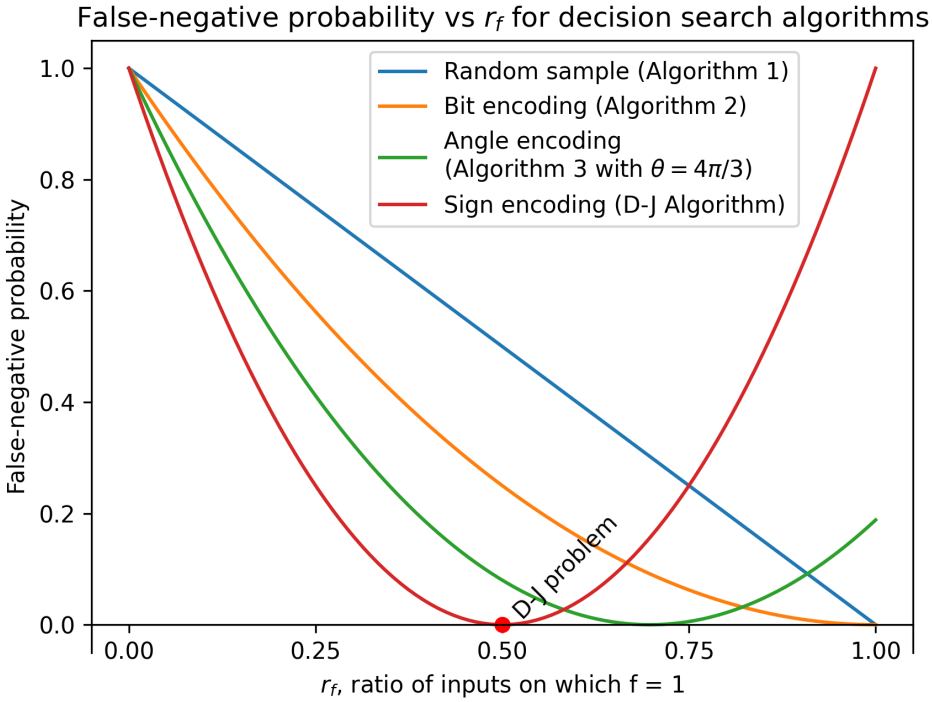


Figure 2.1: This graph compares the probability of a YES-instance being mistaken as NO-instance for different one-query decision search algorithms as a function of r_f, the ratio of inputs on which the function outputs 1. The labelled red point is the special case of the Deutsch-Jozsa algorithm employed to solve the Deutsch-Jozsa problem.

is not available, this algorithm can be seen as achieving a lower error probability in some cases but performing worse on others.

In the last section, we saw that the false-negative probability of the quantum algorithm depends on the inner product between the encoding of the unknown function and the encoding of the constant-zero function. One may naturally consider strategies for reducing this inner product to get a better algorithm.

For example if we took inspiration from quantum interference and encoded the function in a *sign encoding* as $\left|f^{(\pm)}\right\rangle = \frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)} |x\rangle$, and the reference state as $\left|c_0^{(\pm)}\right\rangle = \sum_{x\in\{0,1\}^n} |x\rangle$, then the false-negative probability is:

$$
\begin{aligned}
&|\left\langle c_0^{(\pm)} \middle| f^{(\pm)} \right\rangle|^2 \\
&= \left| \left(\frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} \langle x| \right) \left(\frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)} |x\rangle \right) \right|^2 \\
&= \left| \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{f(x)} \right|^2 \\
&= \left| \frac{1}{2^n} \left(1 \cdot |f^{-1}(0)| + (-1) \cdot |f^{-1}(1)| \right) \right|^2 \\
&= (1-2r_f)^2. \qquad (2.9)
\end{aligned}
$$

Notice that when using the sign encoding, the false-negative probability is zero when $r_f = 0.5$. Thus under the promise that $r_f \in \{0, 0.5\}$, this algorithm is deterministic. This case is almost the Deutsch-Jozsa problem, except that the task is to distinguish the *constant-zero* function from balanced[1]. The promise restricts the inputs to the cases where the quantum algorithm is deterministic and it makes this algorithm have an exponential advantage over deterministic classical algorithms. It achieves an exponential speedup in the worst case running time as the deterministic classical algorithm might take up to $2^{n-1}+1$ queries.

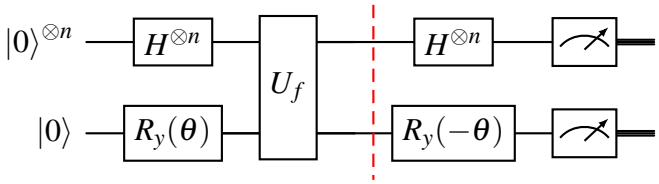
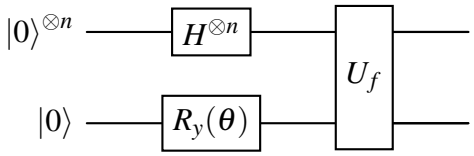


Figure 2.2: Circuit for quantum decision search algorithms. When $\theta = 0$, this performs the algorithm with the bit encoding of the function. When $\theta = 3\pi/2$ this performs the algorithm with the sign encoding of the function, and is equivalent to the Deutsch–Jozsa circuit. For all other values of θ this corresponds to various angle encodings of the function. The dashed vertical line indicates the boundary between the function-encoding stage and the quantum-state-equality stage.

[1]The algorithm presented here would still be able to solve the Deutsch-Jozsa problem.

Algorithm 3: Quantum Decision Search (one query, promise $r_f \in \{0, R\}$ with $R \geq 0.5$) [26]

Prepare $\left|f^{(\theta)}\right\rangle$ where $\sin\theta = \frac{R-1}{R}$:



if $\left|f^{(\theta)}\right\rangle \overset{QSE}{=} \left|c_0^{(\theta)}\right\rangle$ **then**
  return NO
else
  return YES
end

A natural question is how to move this determinism to another value of R. Algorithm 3 achieves this using an *angle encoding* of the function as the state $\left|f^{(\theta)}\right\rangle = \frac{1}{2^{n/2}}\sum_{x\in\{0,1\}^n}|x\rangle[X^{f(x)}R_y(\theta)|0\rangle]$, and comparing it to the reference state $\left|c_0^{(\theta)}\right\rangle = \frac{1}{2^{n/2}}\sum_{x\in\{0,1\}^n}|x\rangle[R_y(\theta)|0\rangle]$ in which case the false-negative probability is:

$$
\begin{aligned}
&|\left\langle c_0^{(\theta)}\middle|f^{(\theta)}\right\rangle|^2 \\
&= \left|\left(\frac{1}{2^{n/2}}\sum_{x\in\{0,1\}^n}\langle x|(\langle 0|R_y(\theta)^\dagger)\right)\left(\frac{1}{2^{n/2}}\sum_{x\in\{0,1\}^n}|x\rangle X^{f(x)}R_y(\theta)|0\rangle\right)\right|^2 \\
&= \left|\frac{1}{2^n}\sum_{x\in\{0,1\}^n}\langle 0|R_y(\theta)^\dagger X^{f(x)}R_y(\theta)|0\rangle\right|^2 \\
&= \left|\frac{1}{2^n}\left(1\cdot|f^{-1}(0)| + \sin\theta\cdot|f^{-1}(1)|\right)\right|^2 \\
&= (1 - r_f + r_f\sin\theta)^2. \qquad (2.10)
\end{aligned}
$$

When $\sin\theta = \frac{r_f-1}{r_f}$, the false-negative probability is zero. Thus if the promise is $r_f \in \{0, R\}$, the algorithm is deterministic if we choose θ such that $\sin\theta = \frac{R-1}{R}$. By choosing $\theta \in [\frac{3\pi}{2}, 2\pi]$, we can get a deterministic algorithm for any $R \in [0.5, 1]$.

Though the quantum algorithm uses only a single query, a deterministic classical algorithm would require $(1-R)\cdot 2^n + 1$ queries in the worst case. Note that the sign encoding ($\theta = \frac{3\pi}{2}$) and bit encoding ($\theta = 2\pi$) are special cases of the angle encoding. The quantum circuit for all these algorithms is given in Figure 2.2.

2.4.4 Relationship to Deutsch-Jozsa algorithm

The Deutsch-Jozsa algorithm [24, 28, 29] is commonly taught as one of the first quantum algorithms in university courses and textbooks (for example, see [5, 30, 31, 32]). It is significant as an early example of an exponential advantage achieved by deterministic quantum algorithms over deterministic classical algorithms [24]. In the Deutsch-Jozsa problem, the input is a function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ that is either constant (i.e. for all inputs, f returns the same output) or balanced (i.e. for exactly half the inputs f returns 0, and for the other half f returns 1), and the goal is to determine which of these is the case. The Deutsch-Jozsa algorithm is illustrated in Fig. 2.3. The measurement outcome of running the circuit will be $|0\rangle$ if and only if f is constant.

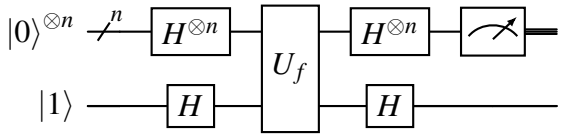


Figure 2.3: Deutsch–Jozsa circuit.

The Deutsch-Jozsa problem is widely acknowledged as being an artificial problem whose primary importance is to show an exponential separation between quantum and classical algorithms. It may not meet the expectations of newcomers expecting to learn faster quantum algorithms for familiar computer science problems. Moreover, the Deutsch-Jozsa algorithm is often just stated and proved to be correct by a derivation rather than being naturally motivated from any high-level concepts. Sometimes, it is explained by reference to the broad principle that interference is an important feature of quantum algorithms, or that the Hadamard transform is a common technique in many quantum algorithms.

The use of two Hadamard transforms surrounding the oracle has sometimes led to the misconception that the Deutsch–Jozsa algorithm fundamentally relies on

the Hadamard transform (which is a type of Fourier transform) [5]. However, our explanation here shows that the Hadamard or Fourier transform is not essential to the algorithm's operation. The Hadamard transform may be replaced by any unitary U that maps $|0\rangle$ to the uniform superposition state. In this case, U can be used to prepare the initial state and $U^\dagger$ can be used for the equality check.

2.5 Axiomatic approach to quantum search

In the last section we showed how the Deutsch-Jozsa algorithm emerges naturally from first seeking a quantum analogue of the one-query decision search algorithm, and then trying to improve the performance of that algorithm. In this section we motivate Grover's search algorithm from an axiomatic perspective. We build upon the previous section and first consider decision search with two queries. Quantum decision search with two queries allows for a new approach beyond classical algorithms. We consider the general form of such two query algorithms and by setting some basic conditions as axioms, we are able to derive the two-query version of a decision version of Grover's search algorithm. If we extend the two-query strategy to multiple queries, we get the full Grover's search algorithm. Note that the axiomatic characterisation does not uniquely pick out Grover's algorithm, but we leave it as an open question how this approach can be extended to obtain a full axiomatic characterisation of Grover's algorithm.

2.5.1 Using two queries

Consider the problem of decision search with two classical queries. An optimal classical algorithm would query the function f at two distinct, randomly chosen points. If either query returns 1, the answer is YES. The queries are independent events; the outcome of the first does not influence the strategy for the second, beyond avoiding the same input. In the limit of a very large function domain, we can ignore this detail about avoiding the same input, as two randomly chosen inputs are overwhelmingly likely to be different. This is akin to running the single-query algorithm twice, slightly improving the success probability but without fundamentally changing the approach. The probability of failure (a false negative) simply decreases from

$(1-r_f)$ to $(1-r_f)^2$, assuming sampling with replacement for simplicity. Without more knowledge of the function, there is no way to combine the queries to create a more sophisticated interaction with the function's structure.

In the quantum setting, two queries unlock a new approach. There is a possibility of modifying the state produced by one query before applying the query oracle again onto the same state. Classically we cannot do this as the output of an oracle query is just a single bit, but in the quantum setting we retain something more in the quantum state after a single query, and we may be able to build a more useful state by performing two queries onto the same quantum state. As we saw in the previous section, inner products measure the performance of the algorithm because the false-negative probability of the algorithm depends on the inner product between the state we prepare and the uniform superposition state that we check equality with. We will try to use the two queries to prepare a state that has a smaller inner product with the uniform superposition state.

For simplicity of presentation and to minimise the inner products, we use the "phase oracle" instead of the "bit-oracle" from the previous section. We denote it as V_f. It acts on a single register of qubits as follows:

$$V_f|x\rangle = (-1)^{f(x)}|x\rangle.$$

This phase oracle V_f can be implemented using a single query to the bit-oracle U_f as shown in Figure 2.4.

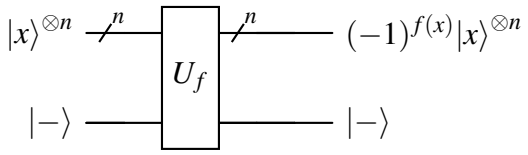
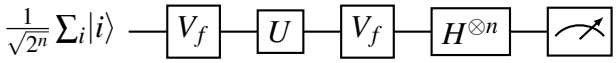


Figure 2.4: Implementation of the phase oracle V_f using the bit-flipping oracle U_f. The ancilla qubit is prepared in the state $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

2.5.2 Axiomatic characterisation of two query search

Our two query algorithm will apply the oracle to the uniform superposition state, perform some intermediate unitary operation on the state, and then apply the oracle again. The key question is what unitary to apply in between the query operations.

We need to find a unitary U such that the following circuit enhances our ability to learn whether f is a YES- or NO- instance.



If f possessed additional structure we could tailor U to that structure. Consider having the promise that either $f(x) \equiv 0$, or $f(x) = 1$ iff $x \equiv c \mod 4$ for a fixed $c \in \{0,1,2,3\}$, and the computational task being to determine which of these two scenarios is the case. In the worst case a classical algorithm requires four queries for this problem, but a quantum algorithm can use just two. The U in between the two queries can be the $+1$-permutation ($|i\rangle \mapsto |i+1\rangle$). Then the state prepared will be either the uniform superposition state (if $f(x) \equiv 0$), or a state which has half the terms having a coefficient +1, and the other half having -1. These two possibilities are orthogonal and performing a quantum state equality procedure to check whether the outcome is the uniform superposition state will solve the problem.

However, f could be any function. If we pick an intermediate unitary U tailored to any particular function, then our algorithm will perform well if that function is indeed the input, but our worst case performance will suffer. In order to not be biased towards any particular input functions, we want to choose a matrix U that is permutationally symmetric.

That is,

$$\sigma^T U \sigma = U, \qquad \text{for every permutation matrix } \sigma.$$

This implies that all diagonal elements of U are equal since,

$$\langle i|\sigma^T U \sigma|i\rangle = \langle \sigma(i)|U|\sigma(i)\rangle = \langle i|U|i\rangle,$$

and $\sigma(i)$ can be chosen to be any index since this equality must hold for every σ. It also implies all off-diagonal elements of U are equal since for $i \neq j$,

$$\langle i|\sigma^T U \sigma|j\rangle = \langle \sigma(i)|U|\sigma(j)\rangle = \langle i|U|j\rangle,$$

and there exists a σ to map each coordinate (i, j) to every other off-diagonal coordinate.

Thus we can write

$$U = c|s\rangle\langle s| + \Delta I \quad |s\rangle := \frac{1}{\sqrt{2^n}} \sum_i |i\rangle.$$

As we will perform an equality check with the uniform superposition state, in the case that f is the constant-zero function, we want the circuit to output the uniform superposition state and so we require that the unitary operation U should have the uniform superposition as an eigenstate,

$$U|s\rangle \;=\; |s\rangle,$$

Taking the above two conditions together,

$$\begin{aligned} U|s\rangle &= (c|s\rangle\langle s| + \Delta I)|s\rangle = |s\rangle \\ &= (c+\Delta)|s\rangle = |s\rangle \\ &\implies c + \Delta = 1. \end{aligned}$$

Now consider that U is unitary. Let $|t\rangle$ denote a vector orthogonal to $|s\rangle$. Then,

$$\begin{aligned} U|t\rangle &= (c|s\rangle\langle s| + \Delta I)|t\rangle \\ &= \Delta|t\rangle = (1-c)|t\rangle. \end{aligned}$$

This implies that $1-c$ is an eigenvalue of U. For unitarity, $|1-c| = 1$, which implies that either $c = 2$, or $c = 0$. We reject the solution $c = 0$, as it just gives U as

the identity matrix. Thus (up to a global phase) we may take,

$$U \;=\; 2(|+\rangle\langle+|)^{\otimes n} - \mathbb{I}_{2^n}.$$

This is the Grover diffusion oracle [6] that can be used even when we have access to more queries.

2.6 Quantum Phase Estimation explained through data structures

The quantum phase estimation algorithm is a fundamental quantum computing technique used in physical simulation and linear algebraic tasks. It is typically taught by stating the quantum circuit for quantum phase estimation and then proving its correctness by algebraic symbol manipulation. This approach rigorously establishes the correctness of the algorithm, but it does not provide a high-level algorithmic perspective on what the algorithm is doing. This section explains the phase estimation algorithm using high-level concepts, by showing that quantum phase estimation employs elementary polynomial data structures from classical computer science. Using the polynomial data structure concepts, the explanation of the quantum phase estimation algorithm can be stated in just a few lines.

The contribution of this section is to provide such an alternative explanation for quantum phase estimation, that is stated in terms of algorithmic ideas that are well-known in classical computer science. We show how data structures for polynomials and a divide-and-conquer framing explain the quantum phase estimation algorithm in a simple way. Note that data structures are an essential concept in the study of classical algorithms. A common strategy for designing a new efficient algorithm is to invent a clever data structure that makes the storage and processing of information required for that computational task more efficient. Although quantum phase estimation has not been stated in this way before, we show how such a retrospective data structure approach simplifies its presentation and points to the source of the quantum advantage. Similarly, divide-and-conquer is a key algorithmic

approach in computer science for designing efficient algorithms. In quantum phase estimation, divide-and-conquer appears within the subroutine known as the quantum Fourier transform (QFT). While not all unitary transformations have efficient circuits, it is possible to efficiently perform QFT because the circuit for it can be constructed using a divide-and-conquer approach on the unitary matrix for QFT.

2.6.1 Data structures for phase estimation

This section introduces the concepts that feature in our explanation of quantum phase estimation. We first review classical data structures for representing polynomials, then show how these motivate the Discrete and Fast Fourier transforms, and finally explain the quantum analogues of these ideas. Table 2.4 condenses these concepts and how they are related.

	Coefficient Representation	**Point-value Representation**
Classical	$[a_0 \quad a_1 \quad \dots \quad a_{N-1}]$	$[p(1) \quad p(\omega_N) \quad \dots \quad p(\omega_N^{N-1})]$
Quantum (Amplitude Encoding)	$\frac{1}{\|a\|}\sum_{j=0}^{N-1} a_j \|j\rangle$	$\frac{1}{\sqrt{N}\|a\|}\sum_{j=0}^{N-1} p(\omega_N^j) \|j\rangle$

Table 2.4: Data structures for storing the polynomial $p(x) = a_0 + \ldots + a_{N-1}x^{N-1}$. The two classical data structures (top row) are stored as a list of values or a vector. The two quantum data structures (bottom row) are quantum states known as the *amplitude encodings* of these vectors. We can convert from the coefficient representation to the point-value representation by applying the Discrete Fourier Transform (DFT) in the classical case, and the Quantum Fourier Transform (QFT) in the quantum case. To convert in the other way, the inverse-DFT or inverse-QFT may be used.

Polynomial data structures

The material in this subsection can be motivated by asking students to consider how polynomials might be represented in a classical computer. The straightforward approach, the **coefficient representation** of a polynomial [18] involves storing a vector (or list) of all its coefficients. For example, the polynomial $p(x) = a_0 + \ldots + a_{N-1}x^{N-1}$ is stored as the vector $a = [a_0 \ldots a_{N-1}]$.

There are other data structures for polynomials. Consider the **point-value representation** of a polynomial [18], which uses the fact that a polynomial, p, with degree less than N is uniquely determined by a set of N distinct points

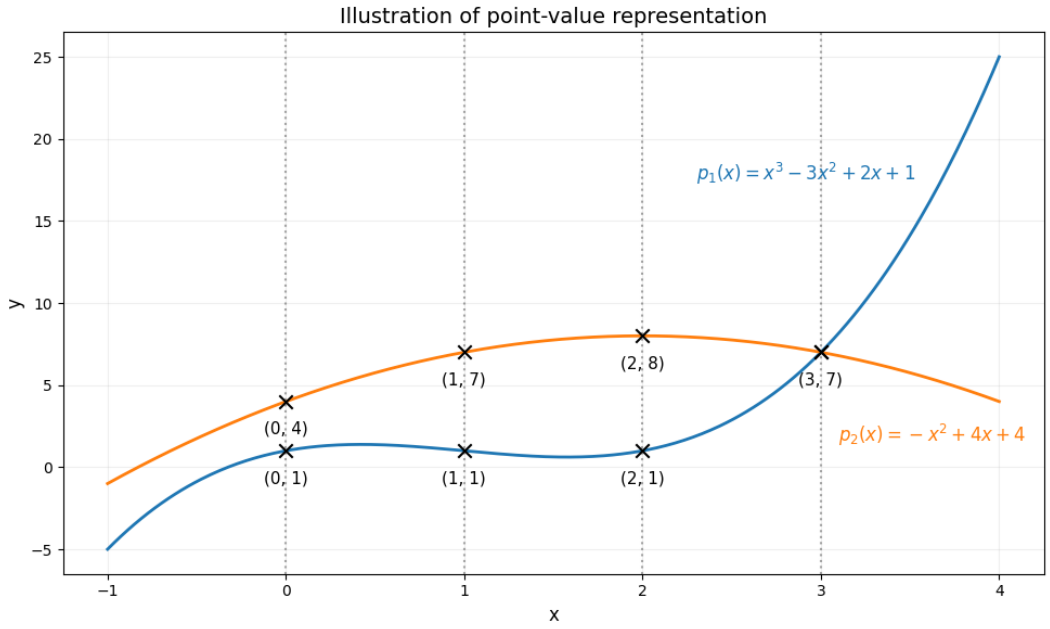


Figure 2.5: There is only one polynomial of degree less than four that passes through the four points $\{(0,1),(1,1),(2,1),(3,7)\}$, and it is x^3-3x^2+2x+1. Similarly the polynomial $-x^2+4x+4$ is the only polynomial of degree less than four that passes through the four points $\{(0,4),(1,7),(2,8),(3,7)\}$. Thus we may represent the polynomials by these sets of points. If we use inputs $x=0,1,2,3$ to create point-value representations, then we may simply store p_1 as $[1,1,1,7]$, and p_2 as $[4,7,8,7]$.

$\{(x_0,p(x_0)),\ldots,(x_{N-1},p(x_{N-1}))\}$. For example, storing two points represents the unique line that passes through them. For a polynomial of degree less than N, we can evaluate it at N different points and store those points. Note that we are free to choose any set of N distinct input values for this purpose, and regardless of our choice these points uniquely identify the polynomial. It is convenient to use the same set of inputs for all the different polynomials we are storing. If we use the inputs $\{x_0,\ldots,x_{N-1}\}$ to create point-value representations, then any polynomial, p, of degree up to N can be stored as the vector $[p(x_0)\ldots p(x_{N-1})]$. This is explained with examples in Figure 2.5.

The point-value representation is significant in classical algorithms because multiplying two polynomials that are represented in this way is efficient. For two polynomials p and q, their product $r=p\cdot q$ can be obtained simply by multiplying

their stored values coordinate-wise:

$$[r(x_0),\ldots,r(x_{N-1})] = [p(x_0)q(x_0),\ldots,p(x_{N-1})q(x_{N-1})]. \tag{2.11}$$

This procedure works as long as the polynomials p and q each have degree less than $N/2$ so that the resulting product r still has degree less than N (and can thus be uniquely determined by N points). Multiplication of polynomials in the point-value representation only requires $\mathcal{O}(N)$ multiplications of numbers. In contrast, multiplication of polynomials in the coefficient representation is done by long multiplication [33], which requires $\mathcal{O}(N^2)$ additions and multiplications of numbers to combine and sum cross-terms.

The coefficient representation is the more common and natural choice, so one would hope to store polynomials in the coefficient representation but use a method to convert between the two representations in order to take advantage of the faster multiplication algorithm in the point-value representation. However, this idea only works if we convert between representations efficiently. The naive approach to converting from coefficients to point values requires multiplying by a Vandermonde matrix [34]. For example, given $p(x) = a_0 + a_1 x + \cdots + a_{N-1}x^{N-1}$ and fixed evaluation points $\{x_0,\ldots,x_{N-1}\}$, the point value representation is obtained by:

$$\begin{bmatrix} p(x_0) \\ p(x_1) \\ \vdots \\ p(x_{N-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{N-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N-1} & x_{N-1}^2 & \cdots & x_{N-1}^{N-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{bmatrix}. \tag{2.12}$$

In general, multiplying a matrix onto a vector requires $\mathcal{O}(N^2)$ elementary operations, and thus the naive approach to convert between the coefficient and point-value representations requires $\mathcal{O}(N^2)$ time. However, directly performing the multiplication in the coefficient representation also takes $\mathcal{O}(N^2)$ time, and thus a slow $\mathcal{O}(N^2)$ conversion can offer us no shortcuts to multiplying polynomials.

However, it is possible to convert representations in $\mathcal{O}(N\log N)$ time if we choose a special set of evaluation points to construct the point-value representation.

Specifically, if we choose the N-th roots of unity (i.e. powers of $\omega_N = e^{i\frac{2\pi}{N}}$) then we get the following Vandermonde matrix:

$$F_N = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \omega_N^2 & \cdots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^{2\cdot 2} & \cdots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \cdots & \omega_N^{(N-1)(N-1)} \end{bmatrix}. \tag{2.13}$$

This matrix (F_N) is called the discrete Fourier transform (DFT) matrix, and it is important in signal processing, functional analysis and abstract algebra. Applying it to the coefficient vector $[a_0, \ldots, a_{N-1}]$ computes the values $p(1), p(\omega_N), \ldots, p(\omega_N^{N-1})$. The Fast Fourier Transform (FFT) is a divide-and-conquer method to multiply F_N onto a vector in only $\mathcal{O}(N \log N)$ time, making the conversion between coefficient and point-value representations more efficient.

The key idea is to split the N coefficients $(a_0, \ldots, a_{N-1})$ into their even- and odd-indexed parts, forming two smaller polynomials of size $N/2$. These smaller polynomials are only evaluated at even powers of ω_N and so only need to be evaluated at half as many inputs. Define $p_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + \ldots$ and $p_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + \ldots$, so that $p(x) \equiv p_{\text{even}}(x^2) + x \cdot p_{\text{odd}}(x^2)$.

Notice that:

$$\omega_N^{2(i+N/2)} = \omega_N^{2i}\, \omega_N^N = \omega_N^{2i}, \tag{2.14}$$

which means,

$$p_{\text{even}}((\omega_N^{i+N/2})^2) = p_{\text{even}}((\omega_N^i)^2), \text{and} \tag{2.15}$$

$$p_{\text{odd}}((\omega_N^{i+N/2})^2) = p_{\text{odd}}((\omega_N^i)^2). \tag{2.16}$$

Thus we only need to evaluate $p_{\text{even}}(x^2)$ (or $p_{\text{odd}}(x^2)$) at half the inputs ω_N^i, namely when $i = 0, 1, \ldots, N/2 - 1$, as the remaining values will be repetitions. We can then combine the evaluations using simple additions and multiplications as:

$$p(\omega_N^k) = p_{\text{even}}(\omega_N^{2k}) + \omega_N^k\, p_{\text{odd}}(\omega_N^{2k}), \qquad k = 0, \ldots, N-1. \tag{2.17}$$

The time complexity of this approach is thus given by the recursive function $T(N) = 2 \cdot T(N/2) + \mathcal{O}(N)$, as each problem of evaluating the polynomial is divided into two polynomial evaluation problems of half the size, and a linear time combination step. Solving this recursion [35] shows that the Fast Fourier Transform reduces the time complexity of converting polynomial representations from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$.

Quantum data structures

The concepts in this subsection can be motivated as the quantum analogues of the classical polynomial data structures from the previous subsection.

Quantum computers are capable of performing some computations on large vectors efficiently by encoding the data into the amplitudes of a quantum state. Since an n qubit quantum state is described using a vector of 2^n amplitudes, computations on vectors encoded as quantum states are capable of achieving large advantages in space and time complexity. The standard encoding, known as the **amplitude encoding** of a vector, is the quantum state obtained by normalising the vector and putting the coefficients into amplitudes of a quantum state, for example as: $v = [v_0 \dots v_{N-1}] \rightarrow \frac{1}{\|v\|} \sum_{i=0}^{N-1} v_i |i\rangle$. Our explanation of QPE features the amplitude encodings of the coefficient and point-value representations of polynomials.

The primary way that quantum computers manipulate amplitude encoded vectors is by unitary operations. When normalised by a $\frac{1}{\sqrt{N}}$ factor, the Discrete Fourier Transform is a unitary matrix. Just as the Discrete Fourier Transform converts the coefficient representation to the point-value representation, its quantum circuit version converts the *amplitude encoding* of the coefficient representation to the *amplitude encoding* of the point-value representation (see Table 2.4). This circuit, and the transformation it implements, are both called the Quantum Fourier Transform (QFT). The circuit can be implemented efficiently by using a divide-and-conquer approach just like the Fast Fourier Transform [36]. The action of QFT on an N-dimensional computational basis state can be decomposed into some simple operations and a smaller instance of QFT on an $N/2$-dimensional computational basis state as:

$$QFT_N|j\rangle_{\text{n qubits}} = \frac{1}{\sqrt{N}}\begin{pmatrix} 1 \\ \omega_N^j \\ \vdots \\ \omega_N^{j(N-1)} \end{pmatrix} = \frac{1}{\sqrt{N/2}}\begin{pmatrix} 1 \\ \omega_N^{2j} \\ \vdots \\ \omega_N^{j(N-2)} \end{pmatrix} \otimes \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ \omega_N^j \end{pmatrix}$$

$$= QFT_{N/2}|j \mod (N/2)\rangle_{\text{n-1 qubits}} \otimes \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ \omega_N^j \end{pmatrix}, \qquad (2.18)$$

where $n = \log_2 N$. This decomposition can be repeatedly applied until the smallest QFT circuit is reached, which is QFT_2 and it is equal to a single Hadamard gate. The circuit to prepare a qubit in the state $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ \omega_N^j \end{pmatrix}$ starting from $|j\rangle$ is shown in Figure 2.6. It relies on the base-2 representation of $|j\rangle$ as $|b_{n-1}\dots b_0\rangle$, and the product $\omega_N^j = \omega_{2^1}^{b_{n-1}} \dots \omega_{2^{n-1}}^{b_1} \omega_{2^n}^{b_0}$.

Figure 2.7 shows how this divide-and-conquer approach gives a recursive definition of the QFT circuit. When this circuit equality is repeatedly applied to decompose the circuit for QFT_N into gates, a sequence of single qubit state preparation circuits is obtained. These state preparation operations can be decomposed further into one- and two- qubit gates by replacing them with the circuit shown in Figure 2.6. Figure 2.8 shows an explicit definition of the QFT circuit in terms of elementary logic gates once all these decomposition steps are performed.

The QPE algorithm uses the inverse-QFT circuit. This can be obtained by simply reversing the QFT circuit and inverting every gate.

2.6.2 Quantum Phase Estimation

Problem

This subsection motivates the Quantum Phase Estimation problem as a natural quantum analogue of a classical eigenvalue problem. Since quantum computers natively operate large matrices and vectors, the use of quantum computers for linear algebraic tasks has been an active field of research.

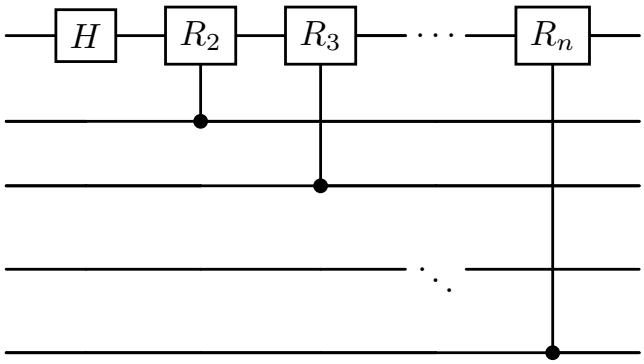


Figure 2.6: The quantum circuit that uses the input computational basis state $|j\rangle$ to prepare the first qubit in the state $\frac{1}{\sqrt{2}}\begin{pmatrix}1\\ \omega_N^j\end{pmatrix}$. In this notation, $R_k = |0\rangle\langle 0| + \omega_{2^k}|1\rangle\langle 1|$. The remaining qubits are left in the state $|j \mod N/2\rangle$, which is the bitstring for j with its most significant bit removed.

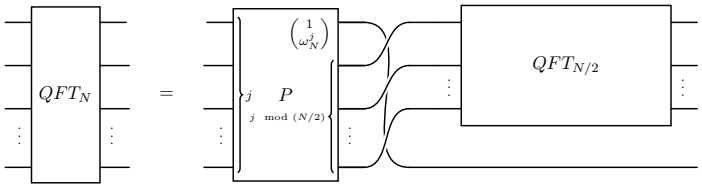


Figure 2.7: Recursive definition of the quantum Fourier transform circuit, which shows how QFT_N can be defined in terms of $QFT_{N/2}$ and some other simple operations. The circuit for the operation P is shown in Figure 2.6. This definition can be repeatedly applied to decompose the QFT_N operation into a quantum circuit. Note that the "base case" (at which the decomposition is no longer needed) is QFT_2, which is equal to the Hadamard gate.

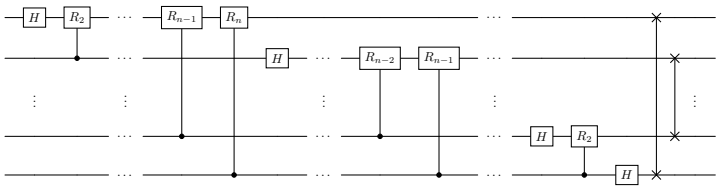


Figure 2.8: Explicit quantum circuit for the quantum Fourier transform, which can be obtained by repeatedly applying the decomposition in Figure 2.7, and substituting the operation P with the circuit shown in Figure 2.6.

Classical algorithms take $\mathcal{O}(N^3)$ time to compute [37] all the eigenvalues of a given matrix. On the other hand, if an eigenvector v of a matrix A is known, the *corresponding eigenvalue* can be found in $\mathcal{O}(N^2)$ time by computing $\frac{v^\dagger Av}{v^\dagger v}$.

Quantum phase estimation is the quantum analogue of the task of computing the *corresponding eigenvalue* when given a matrix and an eigenvector. A key difference between the QPE problem and its classical counterpart is how the data is provided. In QPE, the eigenvector is given as a quantum state in which the vector is amplitude encoded, and the matrix is given as a quantum circuit of which we are given a description. Note that this complicates the comparison of the classical and quantum algorithms, as they solve analogous problems but not exactly the same problem. One way to put them on an equal footing is to view QPE as an extra tool for the corresponding eigenvalue problem that can be used if the input data meets the extra requirements of the quantum algorithm. This can occur when the matrix and eigenvector are too large to operate on directly but still have some structure that makes it easy to work with them on a quantum computer. Notably, this occurs in the case of Hamiltonian simulation, where the unitary operations corresponding to the time evolutions of many interesting Hamiltonians can be implemented as efficient quantum circuits [38]. We may state the QPE problem formally as follows:

Problem 3 *Let U denote a unitary matrix, and let $|\psi\rangle \in \mathbb{C}^N$ be an eigenvector of U with corresponding eigenvalue $\omega_N^k = e^{i2\pi k/N}$ for some $k \in \{0,1,\ldots,N-1\}$, the objective is to determine k given:*

1. *$|\psi\rangle$ as one copy of the quantum state, and*

2. *access to U via black-box calls to a qudit-controlled-U^d gate.*

We define the qudit-controlled-U^d gate as the gate that acts in the following manner: $(C_d U^d)|j\rangle|\phi\rangle = |j\rangle U^j|\phi\rangle$. This is easily motivated as a generalisation of the standard controlled-U gate, which either applies the U gate 0 or 1 times, depending on the value of the controlling qubit. Qudits are d-level quantum systems that generalise qubits, and have a computational basis $\{|0\rangle,|1\rangle,\ldots,|d-1\rangle\}$. The

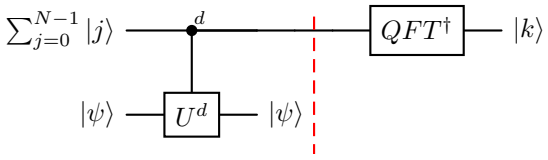


Figure 2.9: High level circuit explaining the Quantum Phase Estimation algorithm. The algorithm has two steps, separated by the dashed line. Step 1 prepares $\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}(\omega_N^k)^j|j\rangle = \frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}(\omega_N^j)^k|j\rangle$, which is the amplitude encoding of the point-value representation of the polynomial x^k. Step 2 converts to the amplitude encoding of the coefficient representation, yielding k.

qudit-controlled-U^d gate applies the U gate j times, if the value of the controlling qudit is j. This allows us to apply U different numbers of times in a controlled way.

Note that all unitaries have eigenvalues of the form $e^{i\theta}$, since unitaries are norm-preserving linear maps. Here we have focused on the case where the eigenvalue to be determined has an angle that is precisely one of the N-th roots of unity. This greatly simplifies the analysis of the algorithm while still preserving its conceptual core.

Algorithm

This subsection shows how the high-level concepts above can be used to explain the QPE algorithm in just a few lines.

The Quantum Phase Estimation algorithm is a quantum circuit (Figure 2.9) that uses the qudit-controlled-U^d gate and the quantum state $|\psi\rangle$ to determine the eigenvalue of $|\psi\rangle$ with respect to U. This circuit has two parts. The first part **prepares an amplitude encoding of the point-value representation of the polynomial** x^k. This can be shown as follows:

$$\begin{aligned} & C_d U^d\left[\tfrac{1}{\sqrt{N}}\textstyle\sum_{j=0}^{N-1}|j\rangle|\psi\rangle\right] \\ &= \tfrac{1}{\sqrt{N}}\textstyle\sum_{j=0}^{N-1}(\omega_N^k)^j|j\rangle|\psi\rangle \quad (\text{Since } U|\psi\rangle = \omega_N^k|\psi\rangle) \\ &= \tfrac{1}{\sqrt{N}}\textstyle\sum_{j=0}^{N-1}(\omega_N^j)^k|j\rangle|\psi\rangle. \end{aligned} \tag{2.19}$$

Notice that the resultant state has amplitudes equal to taking each root of unity to the k-th power, and is thus an encoding of the polynomial x^k.

The second part **converts to the amplitude encoding of the coefficient representation of the polynomial** using the inverse-QFT, giving us the outcome $|k\rangle$, since the encoded polynomial was x^k.

The running time of this algorithm depends heavily on the circuit complexity of implementing the qudit-controlled-U^d operation. When this can be done efficiently in time that is polylogarithmic in N, the QPE algorithm is capable of achieving exponential speedups over classical algorithms that use time polynomial in N to operate on the explicit matrix, U.

When the eigenvalue is not precisely an N-th root of unity, two strategies apply. First, a higher-dimensional qudit register can be used to approximate the eigenvalue more closely. It might turn out that for a larger choice of N, the eigenvalue is exactly a root of unity. However, increasing N requires a larger controlling qudit register, more applications of U, and a larger inverse-QFT circuit. Second, even when the eigenvalue is not exactly a root of unity, the algorithm outputs the nearest root of unity with high probability, since the fidelity between the true state prepared and the ideal state (i.e. amplitude encoding of point-value representation of x^k) is large. The success probability increases as the eigenvalue approaches a root of unity. Thus increasing N improves both the precision of the approximation and the probability of success, at the cost of a larger quantum circuit. The detailed analysis of this can be found in standard algebraic presentations [5].

Chapter 3

Theory of quantum data structures

This chapter conducts a literature review on techniques for storing and manipulating data in a quantum state. It synthesises disparate lines of research to present a theory of quantum data structures. Section 3.1 outlines important no-go results that constrain what quantum data structures are possible. Then we present a few quantum data structures from the literature, namely quantum fingerprints for bitstring storage and comparison (Section 3.2), graph states to store and manipulate graphs (Section 3.3) and a highly space-efficient quantum counter (Section 3.4). Finally, Section 3.5 presents an overview of other important results and research directions within quantum data structures that were not a focus in this thesis.

3.1 Limits on compressing data with quantum information

When first encountering quantum computing, one would expect that quantum states allow for information compression because n qubits have more possible states than n bits do. There are 2^n possible states that n classical bits can be in, which are the elements of the set $\{0,1\}^n$. On the other hand, the set of possible (pure) states of n qubits is $\mathbf{P}(\mathbb{C}^{2^n})$, which is an infinite set[1]. Moreover, when working with qubits, we may use the computational basis to do anything that can be done with bits. Imagine Alice wants to send Bob m bits of information. One would expect that there is a

[1]This is the set of non-zero complex vectors of dimension 2^n, with equivalence up to a constant factor (including both normalisation and global phase).

protocol using states beyond the computational basis by which Alice can encode the bits into a quantum state of less than m qubits and send this quantum state to Bob so that Bob may perform some measurement procedure to determine Alice's message.

It turns out that this is not possible, and is ruled out by the Holevo bound (Ref. [5] §12.1.1, or [39, 40, 41, 42]). Suppose Alice's message is a symbol in the set $\mathcal{X} = \{1, 2, ..., 2^m\}$, and she sends each message i with probability p_i. Suppose for maximum generality that Alice encodes each message i as ρ_i, a (mixed) quantum state from the set of density operators, $\mathcal{D} = \left\{\rho \in \mathbb{C}^{n\times n} \;\middle|\; \rho = \rho^\dagger,\ \rho \geq 0,\ \mathrm{Tr}(\rho) = 1\right\}$. Suppose also for maximum generality that Bob performs a POVM measurement $\{E_j \mid E_j \geq 0, \sum_j E_j = I\}$ and takes the measurement outcome obtained to be his guess of Alice's message. If we model Alice's message as the random variable X, and Bob's guess as the random variable Y, then the Holevo bound states that

$$I(X;Y) \leq S(\sum_i p_i \rho_i) - \sum_i p_i S(\rho_i).$$

Here I(X;Y) denotes the mutual information between X and Y, and S denotes the von Neumann entropy. If Alice sends an n qubit system to Bob then applying the above bound and using basic properties of the von Neumann entropy ($0 \leq S(\rho) \leq n$, for an n qubit state) to the above inequality, we get that at most n bits can be communicated with n qubits. This no-go theorem rules out a straightforward space complexity advantage in encoding information in quantum states.

Nevertheless, information compression with quantum computing remains a natural idea, and one attempt to side-step the "no-go" Holevo bound is to weaken the requirement on what Bob needs to be able to extract from the bitstring. Suppose that instead of recovering all n bits, Bob only needs to recover a particular bit. All the bits are still "encoded" because Bob makes the choice of which bits to read out after receiving the message. An idea like this was initially explored under the name of conjugate coding [43], and later rediscovered in its present incarnation under the name of Quantum Random Access Codes (QRAC) [44]. Quantum Random Access codes are able to outperform classical Random Access Codes (RACs) but unfortunately, the quantum advantage in this setting is limited [44, 45]. A $m \xrightarrow{p} n$

(Q)RAC stores m bits in an n (qu)bit system, such that any of the m bits can be retrieved with probability of success p. For example, it is known that QRACs $2 \xrightarrow{\approx 0.85} 1$ and $3 \xrightarrow{\approx 0.79} 1$ exist, but there are no RACs with parameters $2 \xrightarrow{p} 1$ and $3 \xrightarrow{p} 1$. Nayak's bound [44, 45] states that $n \geq m(1-H(p))$, where H is the binary entropy function. However, there exists a family of *classical* RAC constructions with $n = (1-H(p))m + \mathcal{O}(\log m)$ [44]. Thus the maximum advantage obtainable by QRAC is an additive logarithm factor. A generalisation where any subset of k bits can be read out by Bob was studied in Ref. [46]. The work also proposed XOR-QRAC in which the parity of any subset of k bits could be read out. This can be generalised to f-QRAC [47] in which an arbitrary bit-valued function, f, applied to any subset of k-bits can be read out from the data structure.

As a no-go result, Nayak's bound also allows ruling out the existence of particular quantum data structures, quantum finite automata and quantum communication protocols, if a reduction can be found from the object to a QRAC with parameters that violate Nayak's bound. In Ref. [44], it is shown by a reduction to[2] QRAC that languages $L_n = \{s0 \mid s \in \{0,1\}^*, |s| \geq n\}$ can only be recognised by 1-way Quantum Finite Automata with a number of states that grows exponentially in n. A reduction from quantum communication protocols to Nayak's Theorem was used for lower bound proofs in Refs. [48, 49, 50]. QRACs and Nayak's bound have also been used as targets for reduction in the theory of classical computation [51, 52, 53].

Despite this, the small advantage achieved in QRAC has been investigated in many settings and applications. QRACs enable communication protocols related to Linear Network Coding that are not possible classically [54, 55]. The lower bound on the number of qubits needed for making a QRAC can also be recast as an upper bound on samples needed to learn quantum states [56]. QRAC has also led to ideas for protocols in quantum key distribution [57] and randomness expansion [58].

3.1.1 Space advantages with weaker requirements

As discussed in Chapter 1, there are important non-universal data structures, called *sketches*. Sketches are characterised by using less space or time than a universal data

[2] actually to an even looser kind of encoding called a serial encoding.

structure needs, at the cost of introducing some probability of failure in its operations and lacking a full set of operations. The need for sketches in practice arises when performing computations on very large data sets where it would be prohibitive to store or process all the data. Sketches can also be a useful tool when one knows exactly the operations they will be performing on their data. Imagine Alice sends Bob an m-bit message x, in an n-qubit quantum state, and Bob does not need to retrieve all m bits (as in the Holevo bound setting), nor even one particular bit of Bob's choice (as in the QRAC setting). Rather, Bob needs to be able to compute one of two possible functions $f_1(x)$ or $f_2(x)$. This is a *much weaker* requirement and such settings (with possibly more functions f_i) do allow for larger quantum space advantages.

The vision of quantum state data structures is to design data structures that involve quantum computation and are able to achieve a performance that no classical data structure can achieve. Quantum state data structures involve quantum computation in two ways: (i) at least some of the data is stored as a quantum state, and (ii) at least one of the operations of the data structure involves performing a non-classical quantum channel. They achieve a performance that no classical data structure can achieve in the sense that if we look at the space complexity of storing the data and the time complexities of some set of operations in the quantum state data structure, no classical data structure can match or beat all of those complexities. The next few sections consider examples of such data structures for bitstrings, graphs and counters.

3.2 Quantum Fingerprinting

In this section we look at quantum fingerprinting, which is an early example of a quantum data structure or a quantum sketch. This example illustrates that weakening the requirements on what information is extractable from the data structure allows us to avoid the no-go results of Holevo and Nayak, and makes it possible to achieve large space advantages.

3.2.1 Data structure

A quantum fingerprint [8, 59] of an n-bit string is a quantum state on $\mathcal{O}(\log n + \log(1/\varepsilon))$ qubits that allows equality comparisons between bitstrings to be made with false positive probability ε. The ADT satisfied by quantum fingerprints, which we may call FingerprintADT is defined in Table 3.1. Note that the bitstrings are irrecoverable once they have been encoded as quantum fingerprint states due to the Holevo bound. The idea of the construction is to associate to each of the 2^n values of the bitstring, one vector in the Hilbert space, where these vectors are chosen such that any pair has a small inner product.

Function	Description	Formal Description
`create(x)`	Create a fingerprint of an n-bit string input `x` to be used for later equality comparison.	$f \leftarrow x$
`.equals(y)`	Output whether the stored value is equal to y, with bounded error-probability, then destroy the data structure.	Return f==y, with b. e. p. ε

Table 3.1: Fingerprint Abstract Data Type.

The encoding works as follows. Each bitstring x is first considered as a polynomial (over a field F, for example $\mathbb{Z}_q$, the field of integers modulo a prime q):

$$x = x_1 \ldots x_n \mapsto p_x(z) = x_n + x_{n-1} z + \ldots + x_2 z^{n-2} + x_1 z^{n-1}.$$

This polynomial is then mapped to a quantum state as:

$$p_x(z) \mapsto |f_x\rangle = \frac{1}{\sqrt{|F|}} \sum_{z \in F} |z\rangle |p_x(z)\rangle.$$

Note that for bitstrings $x \neq y$, there can only exist at most $n-1$ values of z for which $p_x(z) = p_y(z)$. This ensures that $\langle f_x | f_y \rangle \leq \frac{n-1}{|F|}$. We may thus calculate the size of field to use that will guarantee $\frac{n-1}{|F|} < \varepsilon$ for our target error probability ε. Appendix A discusses some technical details of how the field can be chosen.

Given a quantum fingerprint that encodes bitstring x, and another bitstring y stored in a classical memory, it is possible to check whether $x = y$ using the Quantum State Equality procedure (see 2.2).

Although the QSE procedure usually consumes one copy of the quantum state, it was shown in Ref. [59] how the data structure can actually support n^c consecutive equality check operations with a bounded error-probability by only using c times more qubits.

3.2.2 Advantage over classical data structures

We now explore the precise nature of the quantum advantage offered by quantum fingerprinting. We will see that it is quite subtle. A common criticism of quantum computing algorithms and protocols like this is to see them as only providing advantages in very specific settings and regimes. This is then used to argue that quantum computing is unlikely to be practically useful. Here we take the dispassionate approach of spelling out the precise conditions in which quantum computers achieve advantages, with two objectives in mind: (1) to better understand the limits and possibilities of computation as they are, as an end in itself, without the urgency of applications; and (2) to set the ground for the discovery of genuinely practical quantum computing techniques which may build upon the insights developed in less practical theoretical techniques. Here we highlight how the discovery of (practically relevant) Shor's algorithm involved building upon the (theoretically relevant) Deutsch-Jozsa and Simon's algorithms. An interesting historical note is that Simon's algorithm was initially not accepted for publication and was known to Peter Shor only because he was one of the reviewers [60]. This example shows how careful work on quantum algorithms even in theoretical settings can be simultaneously very important for further development but also routinely overlooked as uninteresting.

Firstly, consider a "near-miss" computational task on which quantum fingerprinting does not achieve a quantum advantage. In this task, Alice and Bob each possess a large bitstring and they wish to check that they have the same bitstring, while minimising communication. We may assume that Alice will send a message to Bob, after which Bob will perform a computation to determine whether the bitstrings

are equal and alert Alice if they are not. Such situations arise in practice when multiple computers share a large computer file, media corpus, blockchain, sensor data or database and have to ensure they are synchronised.

In practice, this problem is heuristically solved using deterministic hash functions. It is close to certain that if two messages differ, then their hashes will also differ. In the theory of randomised algorithms however, one studies algorithms that have a low failure probability that is independent across runs, and does not depend on the input. Every hash function, h, has infinitely many pairs of inputs x, y such that $h(x) = h(y)$. Thus algorithms and protocols using deterministic hash functions to distinguish bit strings do not meet the bounded error-probability requirement of randomised algorithms. One way to meet this requirement is to design a hash or fingerprint function that takes a random seed value as an input. Such techniques are very efficient and match the $\mathcal{O}(\log n + \log(1/\varepsilon))$ space complexity of quantum fingerprinting. Thus quantum fingerprinting does not achieve an advantage in this setting.

The advantage arises in communication games when Alice and Bob are unable to directly communicate with one another. Quantum Fingerprinting was shown to achieve a space advantage over classical methods in a communication game with three parties, Alice, Bob and a Referee. Alice and Bob receive two bit strings, x and y respectively, and they can only send a single message each to the Referee. This model is called the Simultaneous Message Passing (SMP) model [8, 61]. The collective goal of the three parties is to compute some function $f(x, y)$, which the Referee must output. Quantum Fingerprinting achieves an exponential space advantage in the SMP setting when Alice and Bob do not have access to any shared random bits, and where the goal is to compute the equality function, $f_{eq}(x, y) = [x == y]$. This prevents the use of efficient classical fingerprinting techniques which rely on shared randomness.

Note that even in the SMP model if the goal is only to compare equality, they could just use a deterministic hash and would solve the problem in a heuristic sense. However, to achieve the formal bounded error-probability it has been shown

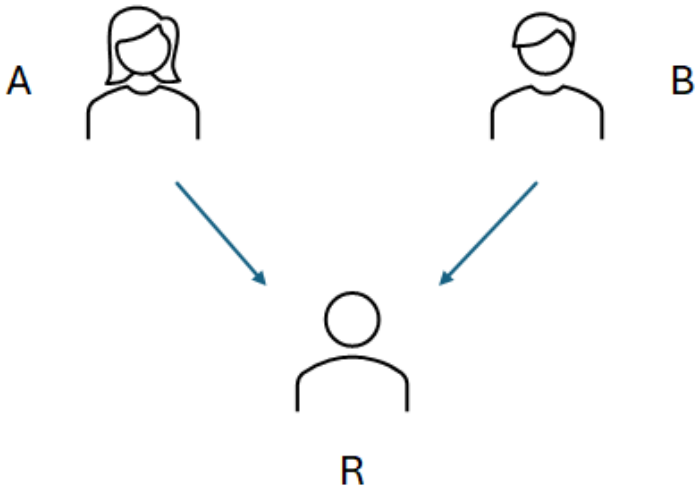


Figure 3.1: In the Simultaneous Message Passing model, the Referee receives a message from Alice and Bob each, and has to compute something from both messages.

that the classical lower bound is $\Omega(\sqrt{n})$. Thus quantum fingerprinting achieves an exponential space advantage in this setting.

Distributed computing settings in which the goal is not only to check equality but also be able to later change the bitstring were explored in Ref. [62]. They also explored the use of graph states, quantum fingerprints and approximate Haar-random states as strategies to achieve a quantum advantage.

3.3 Graph state data structure

The settings above concern quantum data structures that after creation only support some access operations (such as retrieving a bit in a QRAC, or testing for equality in quantum fingerprinting). Following classical data structures, it is natural to wonder about quantum data structures that also support some data modification operations. These may be applied after the data structure is created, to update the data value or perform some procedure upon the initial data value to transform it in some meaningful way. We turn now to the work on quantum data structures for graphs which support both mutation and readout operations.

This topic was first studied in Ref. [63], where three axioms were given that any quantum state encoding of graph must satisfy. The work then showed how multiple families of quantum states such as graph states, Gaussian cluster states, quantum

random networks, and PEPS (Projected Entangled Pair States) all fit the framework. Ref. [64] independently developed the graph-state data structure, with a quadratic space advantage, a range of mutation operations, and several comparison and read-out operations. A fragment of their graph data structure and how it compares to the adjacency matrix data structure is given in Table 3.2.

Features	**Formal Description**	**Graph State** (Quantum) [64]	**Adjacency Matrix**
Storage space	N.A.	$O(V)$ space	$O(V^2)$ space
Comparing two graphs	$G_1, G_2 \rightarrow$ Output $G_1 == G_2$	$O(V)$, for constant failure prob.	$O(V^2)$
Neighbourhood loop toggle	$v,(V,E) \rightarrow (V, E \oplus \{\{n,n\} \mid n \in N(v)\})$	$O(1)$	$O(V)$
Vertex Comparison	$v_1, v_2, G \rightarrow$ Output $N(v_1) == N(v_2)$	$O(1)$	$O(V)$
Checking presence of edge	$(u,v),(V,E) \rightarrow$ Output $(u,v) \in E$	$O(n)$ copies needed	$O(1)$

Table 3.2: Comparison of adjacency matrix and graph state as a data structure.

The graph state encoding of graphs is straightforward. A graph $G = (V,E)$ on $N = |V|$ vertices is represented by its graph state $|G\rangle$ on N qubits, one per vertex, where every qubit is initialised in $|+\rangle$ and a controlled-Z (CZ) gate is applied for each edge. This encoding purports to achieve a quadratic space advantage since a classical data structure must distinguish all $2^{\mathcal{O}(N^2)}$ graphs on N vertices and hence needs at least $\mathcal{O}(N^2)$ bits.

Similar to quantum fingerprints, equality tests can be performed on graph states either by the Quantum State Equality procedure or the quantum SWAP test. This is because graph states are stabilizer states, so $|\langle G'|G\rangle| \leq \frac{1}{\sqrt{2}}$ whenever $G \neq G'$ [21], ensuring a bounded error-probability for equality test operations. The error probability can be exponentially suppressed by storing multiple copies of the graph state as the data structure. Also similar to before, the data structure is destroyed when equality operations are performed.

The fact that some one- and two-qubit gates transform graph states to other graph states was first understood in the study of the mathematics of graph states [65, 66]. Ref. [64] built upon these and added other transformation operations to see graph states as data structure, and to see these operations as data structure operations. For example, a Pauli Z on qubit v toggles the self-loop at v (loop-complementation), and a Pauli X loop-complements all neighbours of v. A CZ gate on qubits a, b toggles

the edge (a,b) and a CNOT controlled on a and applied to b complements all edges between a and the neighbourhood of b.

A more typical definition of GraphADT would have an operation for checking the presence of an edge between two given vertices. This turns out to be a more difficult operation to support in graph states, requiring many copies of the state. With $\mathcal{O}(N)$ copies of the state there is a procedure to retrieve the whole graph, as shown in the paper. By Nayak's bound we know that it would not be too many fewer to retrieve a particular edge, so it seems better to just retrieve the whole graph in every such case.

The operations supported by the graph state data structure and its corresponding ADT are not standard graph operations and thus may not feature in many existing algorithms. It is still important to understand what new efficiencies quantum computers give, and to document and understand these well so they can be varied or extended to eventually become practical algorithms.

There are two kinds of advantages that graph state data structures achieve, as pointed out in Ref. [64]. One is a memory advantage, and the other is a time complexity advantage. The graph state data structure is able to represent N vertex graphs using just N qubits, whereas a full classical description of the graph would require N^2 bits. We explore the precise nature of the computational tasks that are able to exploit this principle in Chapter 6. The time advantage specifically concerns the ability to be efficient on two operations simultaneously. They prove that no classical graph data structure can offer both $O(1)$ edge complementation and $O(1)$ vertex comparison, whereas the quantum data structure can.

3.4 Quantum Counter

Natural numbers are one of the simplest data types in computer science. This section describes quantum data structures for them, building up to the result of Ref. [67].

We consider a simple abstract data type for natural numbers: a counter which represents numbers modulo n, and allows increment and decrement operations. This data type follows the strategy of quantum fingerprinting and graph states to seek a

Operation	Formal Description	Implementation	Complexity
`create(n)`	$G \leftarrow (\{1,\ldots,n\},\emptyset)$	Prepare $\|+\rangle^{\otimes n}$	$O(V)$ space
`.edge_toggle(i,j)`	$G \rightarrow (V, E \oplus \{i,j\})$	$CZ_{i,j}$ or Z_i if $i = j$	$O(1)$
`.delete_vertex(i)`	$G \rightarrow (V \setminus v, E\|_{V\setminus v})$	Z meas. , success on $\|0\rangle$	$O(1)$ and 0.5 failure prob.
`.equals_graph(H)`	Output $G == H$	QSE (sec. 2.2)	$O(E)$, 1 copy, one-sided error.
`.equals_graph(`$\|H\rangle$`)`	Output $G == H$	SWAP test	$O(V)$, 1 copy, one-sided error
`.vertex_equals(i,j)`	Output $N(i) == N(j)$	SWAP test on qubits i and j	$O(1)$
`.nlt(i)`	$G \rightarrow (V, E \oplus \{\{n\} \mid n \in N(i)\})$	X_i	$O(1)$
`.vnt(i,j)`	$G \rightarrow (V, E \oplus \{\{i,v\} \mid v \in N(j)\})$	$CNOT_{i,j}$	$O(1)$
`.all_degrees_even()`	Output $\forall v : \|N(v)\| \equiv 0 \bmod 2$	X meas. all qubits on 2 copies	$O(V)$, 2 copies, 0.5 false positive prob.

Table 3.3: Graph state data structure [64].

quantum space advantage by packing many data codewords into a low dimensional Hilbert space. Thus, the quantum data structure will not allow us to perfectly read the number out of the quantum state due to the Holevo bound. Just like the above examples, it will only have the ability to check whether the stored value is equal to some specific value, with a small error-probability. For simplicity, we will assume only increment and decrement operations because these can be reversibly implemented. By repeated application of increment or decrement, the data structure supports addition and subtraction. Table 3.4 shows the operations of the Counter ADT.

Operation	Description	Formal Description
`create()`	Create a counter initialized to zero.	$c \leftarrow 0$
`.increment()`	Increment the counter by one.	$c \leftarrow c+1$
`.decrement()`	Decrement the counter by one.	$c \leftarrow c-1$
`.equals(i)`	Check if the counter equals a given integer `i`. Return `true` if $c = i$, otherwise return `false` and destroy the data structure.	Return c==i

Table 3.4: Counter Abstract Data Type Operations

The CounterADT specification can be turned into axioms, or constraints, on the form of the quantum state data structure. The first of these captures the idea that our data structure will represent integers from 0 to $n-1$, for some choice of modulus n, and that each of these integers will be assigned a quantum state of some fixed dimension d.

Axiom 1 *There exists an encoding map* $E : \{0,...,n-1\} \to \mathbb{C}^d$. *We denote the quantum state* $E(j)$ *as* $|[j]\rangle$.

The next basic requirement is that we should be able to determine whether two quantum states are encoding the same number. As discussed, expecting the ability to decode from $|[j]\rangle$ to j would defeat any quantum space advantage, due to Holevo's bound. We will only require the ability to check for equality with high probability. If both values that we are checking for equality between are in the quantum state encoding, we may use the SWAP test [8]. The SWAP test successfully distinguishes two different states $|\phi\rangle$ and $|\psi\rangle$ with probability $\frac{1}{2} - \frac{1}{2}|\langle\phi|\psi\rangle|^2$. If one value (call it v_0) is in quantum state encoding, but the other value (call it v_1) is given explicitly in a classical representation, then we can perform the quantum state equality procedure by measuring $|[v_0]\rangle$ on a basis including $|[v_1]\rangle$ (see 2.2). This can also be done by applying the decrement operator v_1 times and checking if the result is $|[0]\rangle$. If $v_0 = v_1$ we will get $|[0]\rangle$ with probability 1. Otherwise, we will get $|[0]\rangle$ with probability $|\langle[v_0]|[v_1]\rangle|^2$. The key figure of merit is thus the inner products $\langle[i]|[j]\rangle$, which should be small if we want to maximise the probability of distinguishing the states.

Axiom 2 *There exists* ε *s.t. for all* $i \neq j$, $\langle[i]|[j]\rangle < \varepsilon$.

The quantities n, d and ε are the three parameters that capture the performance of a specific quantum state data structure for the CounterADT, w.r.t space complexity and error probability for the equality checking operation. We are not interested for now in the time complexity of implementing the operations.

Lastly, we require a reversible increment operation. By applying the inverse of the increment operation, we also have a decrement operation.

Axiom 3 *There exists a unitary* $U \in \mathbb{C}^{d\times d}$, *s.t.* $U|[j]\rangle = |[(j+1) \mod n]\rangle$.

3.4.1 Constructions of quantum counters

To construct a quantum data structure for CounterADT, we need to select a quantum state $|[0]\rangle$, to be the data codeword representing 0, and a unitary operation, U, that implements the increment operation. The axioms from above help to restrict the space of possible choices of $|[0]\rangle$ and U that we need to consider.

Note that because the increment operation is a unitary (and hence normal) matrix, U can be diagonalised as $U = P^\dagger DP$, for diagonal D and unitary P. Together with the requirement that $U^n = I$, this gives us that $D^n = I$, or in other words the diagonal entries of D are all n-th roots of unity. The set of data structure codewords are $\{|[i]\rangle = P^\dagger D^i P|[0]\rangle\}$, and the failure probability of the equality test operations depends on the quantities $\langle[i]|[i+k]\rangle = \langle[0]|P^\dagger D^{-i} P P^\dagger D^{i+k} P|[0]\rangle = \langle[0]|P^\dagger D^k P|[0]\rangle$ for all k. Notice that if we chose $P|[0]\rangle$ as our codeword for 0, and D as our increment unitary, we would still get the same inner products. That is, without loss of generality, the choice of P can be absorbed into the choice of $|[0]\rangle$. Thus, to construct a quantum Counter we only need to find a diagonal unitary D whose elements are all n-th roots of unity, and a state $|[0]\rangle$. Notate the entries of D and $|[0]\rangle$ as:

$$D = \begin{pmatrix} \Omega_0 & 0 & \cdots & 0 \\ 0 & \Omega_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Omega_{d-1} \end{pmatrix}, \quad |[0]\rangle = \sum_{i=0}^{d-1} \alpha_i |i\rangle.$$

Furthermore, notice that quantities we seek to minimise, to reduce failure of the equality operation, have the following form:

$$\begin{aligned} &\{|\langle j_2|j_1\rangle|\}_{0\le j_1,j_2\le n-1, j_1\neq j_2} \\ &\{\langle[0]|D^{-j_2}D^{j_1}|[0]\rangle\}_{0\le j_1,j_2\le n-1, j_1\neq j_2} \\ &\{\langle[0]|D^{j}|[0]\rangle\}_{0\le j\le n-1} \\ &= \{\sum_{i=0}^{d-1} \Omega_i^j |\alpha_i|^2\}_{0<j\le n-1} \end{aligned}$$

Since only $|\alpha_i|$ appears in this expression and not α_i, we may further assume for simplicity that our starting state $|[0]\rangle$ has only real amplitudes in the computational basis, i.e. $\alpha_i \in \mathbb{R}$, for all i. Thus we see that the only choices to be made to construct the data structure are the roots of unity in D and the real vector $|[0]\rangle$. Moreover there is a permutation symmetry in this choice since any consistent reordering of Ω's and α's does not affect the inner products.

We can now state an optimisation problem to find the construction of a quantum Counter that obtains the minimum ε for some fixed d and n.

$$\begin{aligned} \underset{p,\Omega}{\text{minimise}} \quad & \max_{0<j\leq n-1} \sum_{i=0}^{d-1} \Omega_i^j p_i \\ \text{subject to} \quad & p_i \geq 0, \quad i = 0,\ldots,d-1 \\ & \sum_{i=0}^{d-1} p_i = 1 \\ & \Omega_i^n = 1, \quad i = 0,\ldots,d-1 \end{aligned} \tag{3.1}$$

This optimisation problem has a combinatorial explosion in choices of Ω_i, which are chosen discretely from the set of roots of unity. It is unclear what constructions are optimal or close to optimal for choices of d and n in general. We now consider some simple constructions to see how they perform asymptotically on (n,d,ε).

Orthonormal basis encoding

We may choose to encode each integer j simply as the quantum state $|j\rangle$, and choose U s.t. $U|i\rangle = |i+1 \mod n\rangle$. In the set up of the optimisation above $U = D = \text{diag}(1,\omega_n,\omega_n^2,...,\omega_n^{n-1})$ and $|[0]\rangle = \frac{1}{\sqrt{n}}\sum_{0\leq i<n}|i\rangle$. This construction achieves no space advantage by itself it is essentially the classical counter data structure, but does optimise ε when $d = n$, since it has parameters $(n,n,0)$. We will see that it can be useful in combination with other techniques.

Phase Quantum Counter

The smallest non-trivial construction of a diagonal matrix D satisfying $D^n = I$ (up to global phase) is a one qubit rotation such as $R_Z(\frac{2\pi}{n})$. This data structure maps the number j to the quantum state $|[j]\rangle = R_Z(\frac{2j\pi}{n})|+\rangle$. However, as we increase n, the inner products of these quantum states gets very close to 1. Specifically, the largest inner product modulus, which is between any two consecutive numbers such as 0 and 1, is $|\langle[1]|[0]\rangle| = |\langle+|R_Z(\frac{2\pi}{n})|+\rangle| = |\frac{1+e^{i2\pi/n}}{2}| = 1-\frac{\pi^2}{2n^2}+\mathcal{O}(n^{-4})$. The performance is thus $(n,2,1-\frac{\pi^2}{2n^2}+\mathcal{O}(n^{-4}))$, which has inner products close to 1 for anything except small values of n.

Repetitions of a data structure

One idea to decrease the inner products of the counters for larger n is to store copies of the state. This exponentially decreases the magnitude of the inner product in the number of copies stored as $|(\langle\phi|^{\otimes k})(|\psi\rangle^{\otimes k})| = |\langle\phi|\psi\rangle|^k$. However, it also exponentially grows the number of levels of the quantum system. Repeating the data structure maps an (n,d,ε) construction to a (n,d^k,ε^k) construction.

Least Common Multiple constructions using different Quantum Integers

However, repetitions of the phase counter in particular does not suppress the inner product quickly enough for how many levels it introduces to the quantum system. A more successful approach is to store different phase integers. For example consider $U = R_Z(\frac{2\pi}{3}) \otimes R_Z(\frac{2\pi}{5})$. This choice of U gives $n = LCM(3,5) = 15$, $d = 4$ and $\varepsilon = |\langle +|R_Z(\frac{2\pi}{5})|+\rangle| \approxeq 0.8090$.

Let $U_1,...,U_p$ be p quantum integers with parameters $(n_1,d_1,\varepsilon_1),...,(n_p,d_p,\varepsilon_p)$ respectively such that $n_1,...,n_p$ are distinct primes. We can construct the quantum integer with unitary $U_1 \otimes ... \otimes U_p$ which will have parameters $(\Pi_i n_i, \Pi_i d_i, \max_i \varepsilon_i)$

A Least Common Multiple construction can also be obtained by taking the direct sum. Let $U_1,...,U_p$ be p quantum integers with parameters $(k_1,d_1,\varepsilon_1),...,(k_p,d_p,\varepsilon_p)$ respectively such that $k_1,...,k_p$ are distinct primes. We can construct the quantum integer with unitary $U_1 \oplus ... \oplus U_p$ which will have parameters $(\Pi_i k_i, \Sigma_i d_i, \max_i \frac{p-1+\varepsilon_i}{p})$

The above constructions give us a flavour for quantum Counters which have simple constructions. While the optimal constructions are not known, it is known [67] from a randomised argument that constructions with parameters $(p,\mathcal{O}((\log p)/\varepsilon),\varepsilon)$ exist for primes p. The argument considers $|[0]\rangle = \sum_{0\leq i<d}|i\rangle$ and D is chosen by taking roots of unity at random along the diagonal. To relate d and ε, the key idea is to analyse the expected cancellation of the various phases. This works because the expected sample mean of a random set of phases is 0, so the variance is very quickly suppressed as the sample size (i.e. d) is increased. The arguments can also easily be extended beyond primes p, to any integers n, and it was only not done in the original

paper because of the finite automata setting they were considering in which only counters with a prime modulus get an advantage.

Note that any classical counter would have to use a p-level system to have a different encoding for every possible value of the counter, but the quantum data structure is able to have a different encoding for every value of the counter but still have a bounded error probability, ε in its equality operation with just a $\mathcal{O}((\log p)/\varepsilon)$-level system.

3.5 Other directions in quantum data structures

We briefly summarise other important work and directions of research that relate to the phrase "quantum data structures". Each of these are worthy investigations to continue in their own right.

3.5.1 Bounds on the maximum space advantage in unrestricted settings

Large quantum space advantages arise in restricted models of computation such as streaming, finite automata or communication games like simultaneous message passing. For general computation, it is known that the space complexity of unbounded-error quantum Turing machines equals that of unbounded-error randomised Turing machines [68, 69], and unbounded-error randomised space can be simulated deterministically with only a quadratic overhead [70]. Consequently, quantum computation can offer at most a quadratic space advantage over deterministic classical computation in the unrestricted setting. This explains why the search for large quantum space advantages, and hence for quantum data structures that exploit them, is naturally conducted in online and communication-restricted models.

3.5.2 Streaming algorithms

Streaming algorithms naturally involve encodings that can be viewed as data structures which support both modification and a limited form of readout. In a streaming or online setting, the input is provided in a serial manner but is too large to be stored entirely. This might be a good setting for quantum advantage as quantum computers

are not large enough for us to load all our data into them, but it is conceivable even in the short-to-medium term to imagine using a quantum working memory in combination with a large read-only classical memory, or a stream of data that is either received over a network or procedurally generated. The space advantage could mean using a small quantum computer where instead a high performance computer with large virtual memory would be required.

A formalisation of the streaming model of quantum computation is presented in Ref. [71]. There are multiple known exponential quantum space advantages for streaming computational tasks such as checking the disjointness of two sets encoded as bitstrings [71], detecting divisibility by a prime for a number in unary encoding [67, 72], a variant of the Boolean Hidden Matching Problem [73], and a variant of MAX-CUT [74, 75], and a range of classification and dimensionality reduction tasks [76]. Other problems for which quantum space advantages have been found in the streaming setting include approximating triangle count in graphs [77] and frequency moments of a vector [78, 79]. The problem of recognising well bracketed strings with two types of brackets has also been studied in the streaming setting [80], and looser quantum lower bounds have been found but no quantum advantage has been found yet.

3.5.3 Finite automata

Some streaming algorithms have also been explored in the research area called Quantum Finite Automata [81]. Quantum finite automata (specifically those that are classified as 1-way and "realtime"[3]) read symbols of the input string one at a time and update a state. Once the entire string is read, a measurement is performed to determine the output of the computation: a single bit denoting whether the input string is in the language recognised by the QFA. This is a special case of streaming computational tasks. Of particular interest to us are the "succinctness" or "conciseness" results - where it is shown that a quantum automata can recognise the same language using fewer states than any classical equivalent. It is known that there is no space advantage for Quantum Finite Automata over Deterministic Finite

[3]There are many variants of QFAs, for a comprehensive review see Ref. [81].

Automata on decision problems without any promise and with 0-error [49]. Thus to find a practical advantage we have to consider either bounded-error computational tasks or those with a promise. Comparison of PFAs and QFAs on unary language promise problems is investigated in Ref. [82].

QFAs achieve space advantages over classical FAs for various decision problems. One general pattern of such problems takes as input some numbers in unary notation, and checks which one of two modular equivalences are satisfied, promised that exactly one will be satisfied [83, 84, 85, 86]. A notable example is checking the divisibility of a single input unary number by a fixed prime [67, 72]. See Ref. [81] for a detailed review of many QFA succinctness results.

3.5.4 Resource advantages in communication complexity

The quantum fingerprinting advantage in the SMP model is one instance of a broader phenomenon studied in quantum communication complexity, where the "data structure" is a quantum message exchanged between parties. For a survey of more quantum communication protocols and the communication games on which they achieve an advantage, see Ref. [87]. These results are relevant to quantum data structures in two directions: communication protocols can often be read as data structures whose creation and readout are split between parties, and conversely communication lower bounds are a standard tool for proving lower bounds for data structures, such as Nayak's bound as we saw above, or more generally using any data structures in the cell probe model [88].

3.5.5 Time advantages with quantum data structures

This thesis focuses on space advantages, but quantum encodings of data can also be sought for time advantages. One famous example is doing arithmetic in the Fourier basis. Encoding an integer in the phases of a register allows addition to be performed with shallow circuits of controlled rotations and without the carry propagation needed in classical addition circuits [89, 90]. This simplifies arithmetic subroutines inside larger quantum algorithms. The graph state data structure of Section 3.3 also provides a time advantage on some operations: it supports $O(1)$

edge complementation and $O(1)$ vertex comparison simultaneously, a combination provably unattainable by any classical data structure [64].

3.5.6 Quantum Random Access Memory

Quantum Random Access Memory (QRAM) is a proposed device for accessing an array of (classical or quantum) data at an address that is itself a quantum state, by implementing the map $\sum_j \alpha_j |j\rangle|0\rangle \mapsto \sum_j \alpha_j |j\rangle|x_j\rangle$ for an array of data x. Many quantum algorithms are defined assuming access to data in QRAM, notably those in quantum machine learning and quantum linear algebra. QRAM is a data structure and the canonical proposal for implementing it called the bucket-brigade architecture [22], which routes queries through a binary tree of qutrits and was argued to require only $\mathcal{O}(\log N)$ active gate operations per query, offering better noise resilience than a naive fan-out circuit. However, a lot more work is still needed before we can practically implement QRAM [91, 92].

Data structures can also be designed on top of QRAM, by specifying what kind of data is specifically stored in the QRAM. A notable example is the KP-trees data structure [93] which allows efficiently preparing amplitude encodings of the array stored in the QRAM. Famously, it was shown that this quantum data structure can be "dequantized" [94] to a classical data structure which also provides an exponential speedup like the quantum data structure promissed, although the classical data structure is still polynomially less efficient.

3.5.7 Classical data structures in superposition

Another area of investigation under the name of quantum data structures is the adaptation of classical data structures so that superpositions over them can still have interference effects. This requires a unique representation for each possible data value. This is not always a requirement for classical data structures, and sometimes they can have different representations of the same object depending on the history of operations that were called on the data structure to get to its current value [95]. If the unique representation property is achieved, then quantum walk algorithms can perform a quantum walk search over different values of a classical data structure

in superposition, such as in the case of the element distinctness algorithm [23]. Quantum walks over data structures have been turned into frameworks to generically design quantum algorithms by designing the right data structure [96, 97]. Designing classical data structures that behave well in superposition is thus an interesting but distinct research direction from the inherently quantum data structures that are the focus of this thesis.

Chapter 4

Quantum Data Structures from Error Correcting Codes

This chapter presents our novel framework of constructing quantum data structures using classical error correcting codes. The data is encoded in the same style as quantum fingerprinting [8, 59] (see section 3.2) by putting the coordinates of a classical error correcting codewords into a quantum superposition. Where we extend quantum fingerprinting is by showing how once the data has been encoded, it can be modified in its quantum state representation. We present some new quantum data structures, and we show a connection between the graph state data structures and data structures from error correcting codes.

4.1 Background on Error Correction

Error-correcting codes were developed to enable reliable communication over noisy channels [98]. In a typical scenario, a sender wishes to transmit a message drawn from some set Σ^k, where Σ is a finite alphabet (for example $\Sigma = \mathbb{Z}_2$ for binary codes). Directly transmitting the message leaves it vulnerable to corruption from channel noise. The core idea of error correction is to *encode* the message into a longer string called a *codeword* drawn from a subset $\mathcal{C} \subseteq \Sigma^n$ in such a way that the original message can still be recovered even if some of the transmitted symbols are corrupted. The separation between codewords, measured by their Hamming distance, determines the number of errors that can be detected and corrected. These days

error correcting codes are useful mathematical objects that have applications beyond communication such as in complexity theory [99] and algorithm design [100]. Our use of error correcting codes is also not for the purposes of communication, but rather as a useful mathematical object that has the properties we need.

Formally, for an alphabet Σ, a code $\mathcal{C}$ of length n is a subset $\mathcal{C} \subseteq \Sigma^n$. An *encoding function*

$$E_{\mathcal{C}} : \Sigma^k \to \Sigma^n,$$

is an injective function that maps each k-symbol message to a codeword. The *code distance* of $\mathcal{C}$ is defined as

$$d(\mathcal{C}) = \min_{\substack{u,v \in \mathcal{C} \\ u \neq v}} \text{dist}_{\text{Ham}}(u,v),$$

where dist_{Ham} denotes the Hamming distance. A larger code distance means more resilience against errors.

4.1.1 Parameters of a Code

A code $\mathcal{C}$ over Σ with length n, message length k, and minimum distance d is denoted $[n,k,d]_{|\Sigma|}$. Several numerical parameters describe its efficiency and robustness:

- **Length:** n is the number of coordinates in each codeword.
- **Dimension:** k is the size of the message before encoding.
- **Rate:** $R = k/n$ is the fraction of symbols devoted to the message rather than redundancy.
- **Distance:** d controls error detection and correction capability: a code can detect up to $d-1$ errors and correct up to $\lfloor (d-1)/2 \rfloor$ errors.
- **Relative distance:** $\delta = d/n$ measures the error tolerance as a fraction of the block length, which is useful for asymptotic analysis.

4.1.2 Families of Codes

For both practical and theoretical purposes, we often work with *families* of codes. A code family is a collection of codes for various choices of k such that for any integer i, there is a code in the family with code dimension greater than i. These codes are over the same alphabet and have a uniform construction, i.e. there is an efficient, explicit procedure to generate the code of a specific size in the family. Studying families allows us to analyse scaling behaviour, such as maintaining constant rate and constant relative distance as $k \to \infty$.

Examples include:

- **Hamming codes** [101], which form a family of binary linear codes with dimension

$$k = 2^m - m - 1.$$

  For each m, the corresponding block length is

$$n = 2^m - 1 = k + m.$$

  and the code distance is always 3. The codewords of the Hamming code are constructed by placing m parity-check bits at the positions in n that are powers of two, with each parity bit chosen to enforce parity over a distinct subset of the coordinates. Each m produces a new code in the family, for example, the $(7,4)$ Hamming code has $m = 3$, $k = 4$, $n = 7$ and the $(15,11)$ code has $m = 4$, $k = 11$, $n = 15$.

- **Reed–Solomon codes** [102] over finite field $\mathbb{F}_q$ (Note: q must be a prime power), whose codewords are constructed by evaluating polynomials of degree-bound[1] k at n distinct field points, with $n \leq q$. For example, choosing $n = 255$ and $k = 223$ over $\mathbb{F}_{256}$ yields the $(255, 223)$ code capable of correcting up to 16 symbol errors.

- **Reed–Muller codes** [103] over $\mathbb{F}_2$, denoted $\mathrm{RM}(r,m)$, whose codewords are constructed by evaluating polynomials in m variables of total degree at most r

[1] A polynomial of degree-bound k is a polynomial whose degree is strictly less than k.

at every point of $\mathbb{F}_2^m$. Since $x^2 = x$ over $\mathbb{F}_2$, it suffices to consider multilinear polynomials. The resulting code has length $n = 2^m$, dimension $k = \sum_{i=0}^{r} \binom{m}{i}$, and minimum distance $d = 2^{m-r}$. For example, choosing $r = 1$ and $m = 5$ yields the $(32, 6)$ code $\mathrm{RM}(1,5)$ with minimum distance 16, capable of correcting up to 7 errors.

4.1.3 Code Equivalence and Automorphisms

There are two kinds of transformations that one can perform on any error correcting code that do not change the pairwise Hamming distances of codewords. Thus applying such a transformation to a code leaves it "essentially the same" [104]. The first is to permute the coordinates of all the codewords in the same way. The second is to remap the symbols at a particular coordinate in all the codewords in the same way.

Two binary codes $\mathcal{C}, \mathcal{D} \subseteq \mathbb{Z}_2^n$ are said to be *equivalent* if there exists a coordinate permutation $\pi \in S_n$ and a bit vector $a \in \mathbb{Z}_2^n$ such that

$$\mathcal{C} \;=\; \big\{ \pi(y) \oplus a \;:\; y \in \mathcal{D} \big\}.$$

Here, $\pi(y)$ permutes the coordinates of y, and $\oplus a$ flips the bits indicated by a. A *code equivalence transformation* is a specific choice of (π, a) that maps $\mathcal{D}$ to $\mathcal{C}$. When $\mathcal{C} = \mathcal{D}$, such a transformation is called a *code automorphism*.

For codes over a larger alphabet Σ (e.g. $\mathbb{F}_q$), the bitwise flip is replaced by a *symbol permutation*. Two q-ary codes $\mathcal{C}, \mathcal{D} \subseteq \Sigma^n$ are *equivalent* if there exists a coordinate permutation $\pi \in S_n$ and a tuple of alphabet permutations $(\tau_1, \ldots, \tau_n)$, where $\tau_j \colon \Sigma \to \Sigma$ is a bijection, such that

$$\mathcal{C} \;=\; \big\{ \tau_1(y_{\pi(1)}) \,\|\, \tau_2(y_{\pi(2)}) \,\|\, \ldots \,\|\, \tau_n(y_{\pi(n)}) \;:\; y \in \mathcal{D} \big\}.$$

In this form, binary equivalence is the special case where each τ_j is either the identity or the NOT map $x \mapsto x \oplus 1$.

Equivalence and automorphisms are often used in classical coding theory for designing efficient decoding algorithms [105], proving symmetry properties, and classifying codes up to relabelling. In our setting, these transformations have a more direct operational meaning: any automorphism of a code corresponds to a reversible transformation that can be applied to a quantum state encoding without leaving the code space. Note that the term code automorphism is often also used in the literature to refer to *permutation automorphisms* which only include transformations that permute the coordinates of the code. There are also multiple other definitions of code automorphisms which mostly assume that the code in question is linear [104].

4.1.4 Example: Automorphism in a Reed–Muller Code

We now illustrate these definitions in the case of the second-order Reed–Muller code on three variables, $\mathrm{RM}(2,3)$. The codewords of this code are computed by evaluating on all inputs, the Boolean polynomials of degree at most two which are in three variables, i.e.

$$f(x_1,x_2,x_3) \in \mathbb{F}_2[x_1,x_2,x_3], \quad \deg f \leq 2,$$

will be evaluated on every point of the cube $\{0,1\}^3$.

- The *length* is $n = 2^3 = 8$, since we evaluate at all 8 input points.

- The *dimension* is $k = 7$, since there are 7 monomials of degree ≤ 2:

$$x = [1,\ x_1,\ x_2,\ x_3,\ x_1x_2,\ x_1x_3,\ x_2x_3].$$

It is well-known that the code distance is 2, thus $\mathrm{RM}(2,3)$ is an $[8,7,2]$ binary linear code. Suppose we have the message bitstring

$$m_f = [0,1,0,0,0,0,1].$$

This can be encoded as the polynomial:

$$f(x_1,x_2,x_3) = m_f \cdot x = x_1 + x_2x_3.$$

Evaluating f at the lexicographically ordered points

$$(000),(001),(010),(011),(100),(101),(110),(111)$$

produces the *codeword*

$$c_f = [0,0,0,1,1,1,1,0].$$

Now we will illustrate how we can apply a permutation σ of coordinates that effectively swaps the variables x_1 and x_2.

$$\begin{aligned} f^\sigma(x_1,x_2,x_3) &= f(x_2,x_1,x_3) \\ &= x_2 + x_1x_3. \end{aligned}$$

The new *message bit string* is

$$m_{f^\sigma} = [0,0,1,0,0,1,0].$$

Swapping variables also permutes the evaluation points:

$$(x_1,x_2,x_3) \;\mapsto\; (x_2,x_1,x_3).$$

Two pairs of monomials have to be swapped: x_1 with x_2, and x_2x_3 with x_1x_3. This induces a permutation of the codeword coordinates:

$$(1,2,3,4,5,6,7,8) \mapsto (1,2,5,6,3,4,7,8).$$

Applying this to the codeword c_f yields

$$\pi(c_f) = (0,0,1,1,0,1,1,0).$$

We can check that this was done correctly by verifying that evaluating the transformed polynomial f^σ directly at the original order points gives exactly the same codeword:

$$c_{f^\sigma} = (0,0,1,1,0,1,1,0) = \pi(c_f).$$

This worked example illustrates how a code automorphism acts on a single codeword by showing how a permutation of coordinates in the codeword transforms the underlying message bit string. Similarly, this transformation will act on all other codewords, mapping them to other codewords. The action on the whole codespace will be bijective.

4.2 Data structure framework

We now build on these error correcting concepts to show a general framework for designing quantum data structures. In overview the error correcting properties of the codes are used to create sets of quantum states with small inner products, and the automorphisms are used to define reversible operations that act directly on the quantum data structures to change the underlying information.

4.2.1 Data structure values

We first describe the encoding, or the format in which data is represented by these data structures. For simplicity, we first consider data structures where the possible values of our data structure are all 2^k possible messages on k bits.

Each message $x \in \mathbb{Z}_2^k$ is first mapped to an n-bit codeword $E(x)$ by the chosen error-correcting code. We then represent $E(x)$ in a quantum function state, which effectively puts the coordinates in superposition just like quantum fingerprinting. The encoding process could be performed on a quantum computer in two stages:

1. **Error correction code encoding:** Map the k-qubit register containing the message into an n-qubit register containing the corresponding codeword:

$$|x,0,\ldots,0\rangle \xrightarrow{\text{Encoder}} |E(x)\rangle.$$

This is a unitary embedding of the k logical qubits into the n physical qubits of the code.

2. **QRAM-style loading:** Prepare an index register $|i\rangle$ and an auxiliary value register initialised to $|0\rangle$. Use a QRAM-like operation [106] to load the i-th coordinate of $E(x)$ into the value register without disturbing the rest of the codeword:

$$|i\rangle|0\rangle \otimes |y\rangle \xrightarrow{\text{QRAM loader}} |i\rangle|y_i\rangle \otimes |y\rangle.$$

Combining these steps, the overall loading procedure is:

$$\begin{aligned}
\sum_i |i\rangle|0\rangle \otimes |x0\ldots0\rangle &\xrightarrow{\text{Encoder}} && \sum_i |i\rangle|0\rangle \otimes |E(x)\rangle \\
&\xrightarrow{\text{QRAM loader}} && \sum_i |i\rangle|E(x)_i\rangle \otimes |E(x)\rangle \\
&\xrightarrow{\text{Discard}} && |[x]\rangle = \sum_i |i\rangle|E(x)_i\rangle.
\end{aligned}$$

If the code alphabet is binary, the registers require:

- $\log n$ qubits for the index register $|i\rangle$,
- 1 qubit for the value register $|E(x)_i\rangle$.

Thus the total space is $\log n + 1$ qubits. Since classically storing x requires k bits, to obtain a quantum space advantage we require $\log n + 1 < k$. For non-binary codes over an alphabet of size D, the value register requires $\log D$ qubits, giving total space $\log n + \log D$ qubits.

If E is an $[n,k,d]$ code, then the states encoding two different messages x and x' have inner product bounded by:

$$\left|\langle [x]|[x']\rangle\right| \leq \frac{n-d}{n}.$$

If we instead encoding the bitstring in a sign encoding, $|[x]\rangle = \sum_i (-1)^{E(x)_i}|i\rangle$, then the bound will be $\frac{n-2d}{n}$. This follows directly from the code's minimum distance d: large

d ensures that distinct codewords differ in many coordinates, so their corresponding quantum states have small overlap and are thus more easily distinguished. The equality comparison procedure can either be done by the quantum state equality procedure (see 2.2) or a SWAP test [8]. This is exactly the property exploited in quantum fingerprinting.

In the most direct interpretation, these messages are simply elements of the bitstring data type, with no additional structure. However, in many interesting cases, the k-bit messages are not arbitrary: they correspond to more structured mathematical objects that arise from the properties of the error-correcting code used. For example, if the code is linear and over a larger alphabet, the message space has a vector space structure and the data structure can be seen as storing vectors rather than bitstrings. If a polynomial code is used, then the data structure can be seen as storing polynomials rather than bitstrings. Ultimately, all classical data can of course be seen as some sort of bitstring, but codes that utilise some algebraic object are more naturally seen as storing that object as data. The additional structure also determines what operations on the stored data can be implemented, helping the interpretation as being of a particular type.

4.2.2 Data structure operations

In this section we study classes of operations that *modify* the stored data in our quantum data structure framework.

We will restrict our attention to those operations that can be implemented as *reversible unitary transformations* on the quantum state. This choice is made so that the data structure modification operations can be performed without some probability of destroying the data structure. This would allow an indefinite number of modification operations to succeed deterministically. Data structures with non-unitary operations can also be considered but are beyond the scope of this chapter.

It is helpful to recognise that the operations can be interpreted as transforming the data structure on three different layers:

1. At the **message** layer, we see the data structure as encoding a k-bit (or k-symbol) message. The transformation will change what message or value the data structure is storing.

2. At the **codeword** layer, we see the error correcting code that data structure uses as part of its construction, and the message is stored as an n-bit codeword produced by applying the encoding function to the original message. The transformation will act as an automorphism on the codeword, changing which codeword the data structure is using.

3. The data is physically stored as a **quantum state**, which is specifically an indexed superposition of the coordinates of the codeword. On this level, we apply a unitary operation to transform the data.

Note that not every transformation on the message space can be realised as a valid operation on this data structure. The first restriction is imposed by *reversibility*: a unitary operation must have a well-defined inverse, so any irreversible map such as resetting the stored value to a specific fixed codeword cannot be implemented.

A second, more subtle restriction follows from the fact that a unitary operation must preserve inner products. For any two messages x and x' with encodings $E(x)$ and $E(x')$, we have:

$$\langle E(x) \,|\, E(x') \rangle \quad \text{is unchanged by any unitary operation.}$$

This means that if a proposed transformation of codewords changes these inner products, it cannot correspond to a physically realisable unitary, even if it is a bijective operation on the message space.

To illustrate this, consider the repetition code with $k = 2$, $n = 4$, whose codewords are

$$\{0000, 0011, 1100, 1111\}.$$

We denote the encoded state for message $m \in \{00,01,10,11\}$ by $|[m]\rangle$, e.g.

$$|[00]\rangle \;=\; \tfrac{1}{2}\big(|0\rangle|0\rangle + |1\rangle|0\rangle + |2\rangle|0\rangle + |3\rangle|0\rangle\big),$$

$$|[01]\rangle \;=\; \tfrac{1}{2}\big(|0\rangle|0\rangle + |1\rangle|0\rangle + |2\rangle|1\rangle + |3\rangle|1\rangle\big).$$

In general, the inner product between two encodings $|[x]\rangle$ and $|[y]\rangle$ is

$$\langle [x] \mid [y] \rangle \;=\; \frac{1}{4} \cdot \#\{i \in \{1,2,3,4\} : x_i = y_i\},$$

i.e. the fraction of positions where the underlying codewords agree.

Now consider the bijective function on the message space f:

$$f : \{0,1\}^2 \to \{0,1\}^2, \quad f = \begin{cases} 00 & \mapsto 01 \\ 01 & \mapsto 10 \\ 10 & \mapsto 00 \\ 11 & \mapsto 11 \end{cases}$$

Suppose for contradiction that there exists a unitary operation U that implements f on the message space, i.e.

$$U|[c]\rangle = |[f(c)]\rangle, \qquad \forall c \in \mathcal{C}.$$

Take $c = 0000$, $c' = 1111$. We have

$$\langle [00] \mid [11] \rangle \;=\; \frac{1}{4} \cdot 0 \;=\; 0.$$

But under f these map to 0011 and 1111, so

$$\langle [01] \mid [11] \rangle \;=\; \frac{1}{4} \cdot 2 \;=\; \frac{1}{2}.$$

This is a contradiction as unitary operations preserve inner products. Thus no such unitary operation exists and this map f on the message space cannot be physically implemented.

The upshot is that the set of physically realisable transformations on the data structure is smaller than the set of all reversible operations on the message space.

Within these constraints, we can identify at least two broad and natural families of operations that can be implemented. Each such operation can be described both in terms of how it transforms the underlying quantum state and how it transforms the classical codeword that defines the data structure.

1. **Permutations of coordinates** — On the quantum state level, this is a unitary that permutes the index register:

$$\sum_i |i\rangle|E(x)_i\rangle \longmapsto \sum_i |\sigma^{-1}(i)\rangle|E(x)_i\rangle.$$

Which can be rewritten as:

$$\sum_i |i\rangle|E(x)_{\sigma(i)}\rangle.$$

On the *codeword level*, this means that the n-bit string

$$E(x) = (E(x)_1, \ldots, E(x)_n)$$

is replaced by the permuted codeword

$$(E(x)_{\sigma(1)}, \ldots, E(x)_{\sigma(n)}).$$

2. **Symbol relabelling operations** — On the *quantum state level*, this is a unitary that applies a relabelling function π_i to the second register independently on each coordinate:

$$\sum_i |i\rangle|E(x)_i\rangle \longmapsto \sum_i |i\rangle|\pi_i(E(x)_i)\rangle.$$

For binary alphabets, the special case is bit-flipping (XOR with a vector v):

$$\sum_i |i\rangle|E(x)_i\rangle \longmapsto \sum_i |i\rangle|E(x)_i \oplus v_i\rangle.$$

On the *codeword level*, this means that each symbol of the codeword is replaced by its image under the relabelling map, i.e.

$$(E(x)_1,\ldots,E(x)_n) \mapsto (\pi_1(E(x)_1),\ldots,\pi_n(E(x)_n)).$$

The operations on the codeword that can be made from composing these two types of operation are specifically the code automorphisms.

Consider an error correcting code with encoding function $E : \mathbb{Z}_D^k \to \mathbb{Z}_D^n$. To construct a data structure operation that acts on the message space according to bijective function f on $\mathbb{Z}_D^k$, we would need to find a code automorphism F on the codespace such that:

$$F\big(E(x)\big) = E\big(f(x)\big) \quad \text{for all } x \in \mathbb{Z}_D^k.$$

At the quantum state level, this correspondence means that there exists a unitary U_f with the action:

$$U_f|[x]\rangle = |[f(x)]\rangle.$$

Note that different code automorphisms might correspond to the same action on the message space. Consider again the repetition code from above with codewords $\{0000, 0011, 1100, 1111\}$. Let σ be the coordinate permutation that swaps $(0 \leftrightarrow 1)$ and $(2 \leftrightarrow 3)$, i.e. $\sigma = (01)(23)$. For any codeword $c = E(m_1m_2)$ we have $\sigma(c) = c$, so both the identity permutation id and σ act identically on the codewords.

We will try to find a diverse set of operations on the message space that can be implemented physically because they are code automorphisms. We will be interested that these operations act in some meaningful way, that may be useful as a data structure operation. The notion of an automorphism group might help us

identify possible operations on the data structure, but we must be careful that many automorphisms will correspond to the same underlying operation on the data.

4.2.3 Requirements on Error-Correcting Codes

Having seen how a classical error-correcting code (ECC) can be used to define a quantum data structure, we now turn to the question of which codes give rise to interesting and performant data structures. We will first determine what data structure parameters we get from some given code parameters $[n,k,d]$. Then looking at the desired data structure parameters, we will infer backwards the desired code parameters.

An $[n,k,d]$ binary code specifies a subspace of $\mathbb{Z}_2^n$ consisting of 2^k codewords, each at least Hamming distance d from any other. From such a code, we can construct a quantum data structure with the following properties:

- The data structure requires $\log n + 1$ qubits in the bit encoding, and $\log n$ qubits in the sign encoding.
- It encodes one element from a set of 2^k possible elements (classically requiring k bits).
- The inner product between any two distinct encoded states is bounded above by $\frac{n-d}{n}$ in the case of the bit encoding (or $\frac{n-2d}{n}$ in the case of the sign encoding).

The same logic applies for codes over a larger alphabet Σ of size D: An $[n,k,d]_D$ code yields a data structure that:

- Uses $\log n + \log D$ qubits,
- Encodes one element from a set of D^k elements (classically requiring k D-ary symbols),
- Has inner products between distinct encodings bounded above by $\frac{n-d}{n}$.

However, notice that the additional space complexity for using a large alphabet code is small. Thus codes on large alphabets should be explored as they will achieve a greater space advantage.

Our target quantum data structures have a constant probability of success in equality comparisons, achieve a space advantage and have computationally meaningful operations. Thus the code properties we desire are:

1. **Constant relative distance:** The relative distance $\delta = d/n$ should be a positive constant, i.e.

   $$d = c \cdot n \quad \text{for some fixed } c > 0,$$

   so that the inner products between distinct encodings are bounded by a constant as k grows. This ensures that the equality comparisons in the data structure can be performed with a constant probability of success. If we want a specific probability of success we can store copies of the data structure and perform replicates of the equality comparison procedure to amplify the probability of success. This is essential for defining a *uniform family* of data structures of arbitrary size with consistent distinguishability properties.

2. **At least a modest rate:** Unlike performant classical codes in communication, we do not require the code rate $R = k/n$ to be a constant as k grows. To store k classical bits (or symbols from a large alphabet), or to store some other data type which takes one of 2^k (or D^k) possible values, a classical data structure cannot use less than k bits (or symbols), otherwise it will have the same encoding for two possible values of the data structure. On the other hand, our quantum data structures will use only $\log n + 1$ (or $\log n + \log D$) qubits but still have a unique encoding for each possible value of the data structure. A discussion of the settings in which the space advantage is achieved is pursued in the next chapter.

   A necessary condition for a space advantage is:

   $$\log n + 1 < k$$

   for the binary case. Or for a code over an alphabet of size D:

   $$\log n + \log D < k.$$

This is because the space complexity of the data structure is $\log n + 1$ qubits (or $\log n + \log D$ qubits for alphabet size D), while a deterministic classical data structure would need k bits, or k symbols. For example a code family where $n = \mathcal{O}(k^c)$ for a constant c would yield a quantum data structure.

3. **Interesting automorphism operations:** We also care about the operations that the data structure supports. The more such transformations a code has, the richer the set of possible data structure operations.

Fortunately, our requirements are more relaxed than those in classical communication theory:

1. We do not require the code to be decodable as we only care about the encoding and the code distance.

2. We do not require constant rate, but only constant relative distance.

3. We are free to use a large alphabet. In classical error correction, large alphabets are only useful as theoretical tools as communication media almost always transmit bits of information. Straightforward conversion from large alphabets to bits leads to a poor code distance [102] and only careful techniques allow the conversion of good large-alphabet codes to good binary codes [107]. In our setting, the symbols are placed in quantum superposition, so the "cost" of a large D appears only as a small additive $\log D$ term in qubit complexity.

These loosened constraints open the door to code families that have not been considered in the usual error correction literature. However, because this precise combination of requirements has not been systematically studied before, there is probably a lot more code design that can be done to construct new code families that optimise both constant relative distance, modest rate and interesting symmetry groups for data structure operations. The properties mentioned above are those that are most important for the purposes of achieving a quantum space advantage. There may also be other practical properties that we may want our data structures to have - for example efficient circuits for encoding and modifying the data.

4.3 Data Structures

We now present examples of quantum data structures that were designed based on our error-correcting codes framework described above. Each subsection describes (i) the stored data type, (ii) the quantum space complexity, (iii) supported operations, and (iv) the underlying error correcting code used by the data structure. Some brief remarks are made about where these data structures may be useful, but longer discussion on this point is in Chapter 6.

4.3.1 Bitstrings with XOR (Binary linear codes)

A linear code is one in which the codespace is a vector space. Notably, the linear combination of codewords is still a codeword in these codes. Error-correcting codes do not have to be linear, but often many error correcting codes are linear, as they are easier to analyse by focussing on a basis set for the vector space. The linearity of these codes, which is often assumed in most error correcting code contexts, is a small amount of additional structure over the requirements on codes set out in the previous section. In this data structure, we will see how this linearity (together with a binary alphabet) can be utilised to support an XOR operation, since XOR-ing two codewords produces the codeword of the two underlying messages XOR-ed together. While any binary linear code may be used, the most space efficient data structures will come from using the highest rate codes, namely the asymptotically good binary linear codes, which are those with constant rate and constant relative distance.

Datatype	Bitstrings of length k
Space complexity	k-bit string encoded in $\mathcal{O}(\log k)$ qubits
Operations	XOR any bitstring onto the stored bitstring
Code used	Binary linear code with constant rate and constant relative distance (e.g. Justesen [107])

Table 4.1: Summary of the bitstring-with-XOR data structure.

Let $\mathcal{C}$ be a binary linear $[n,k,d]$ code with $k = \Theta(n)$ and $d = \Theta(n)$. Write $E\colon \{0,1\}^k \to \{0,1\}^n$ for its encoder. For $x \in \{0,1\}^k$, the data structure is in the quantum state:

$$|[x]\rangle = \sum_{i=0}^{n-1} |i\rangle |E(x)_i\rangle.$$

Function	Description	Formal Description
`create(x)`	Create an instance storing bitstring $x \in \{0,1\}^k$.	$b \leftarrow x$
`.xor(y)`	XOR bitstring y with the stored bitstring.	$b \leftarrow b \oplus y$
`.equals(y)`	Check if the stored bitstring equals y.	Return `true` if $b = y$, else return `false` with high probability and destroy the data structure.
`.equals(|[y]⟩)`	Check if the stored bitstring equals that stored in the state $\|[y]\rangle$.	Return `true` if $b = y$, else return `false` with high probability and destroy both data structures.

Table 4.2: Bitstring-with-XOR ADT.

The total number of qubits in this state is $\lceil \log n \rceil + 1 = \mathcal{O}(\log k)$. This data structure is exponentially more efficient than storing the k classical bits, because the code used has a constant rate. Recall that the constant relative distance of the code guarantees distinguishability, since $|\langle [x]||[y]\rangle| \leq (n-d)/n = \mathcal{O}(1)$ whenever $x \neq y$. Recall that the constant relative distance is essential for the data structure to support the equality comparison operation with a constant success probability for all k. The specific failure probability depends on the rate of the code chosen. The equality checking procedure can be done in one of two ways. If the bitstring y is known explicitly then `.equals(y)` can be implemented by the QSE procedure (see section 2.2). On the other hand, if two instances of the data structure need to be compared, the SWAP test can be used. Choosing binary linear codes whose rate was in a smaller asymptotic class would have led to a data structure needing more space. However, this may be worthwhile for some practical reasons such as if the quantum circuits for working with the constant rate codes are much more complex than the lower rate codes. This encoding was proposed in the original quantum fingerprinting paper [8]. Here we extend it by showing that it also supports the modification operation of XOR-ing any bitstring.

The `create(x)` operation prepares the quantum state encoding of a particular bitstring by first preparing the codeword bits in a quantum register, and then loading them into the superposition, using a circuit that works similarly to the QRAM circuit

[91]. The high-level procedure is as follows:

$$\sum_i |i\rangle|0\rangle|x0\ldots0\rangle \xrightarrow{\text{Encoder}} \sum_i |i\rangle|0\rangle|E(x)\rangle \xrightarrow{\text{QRAM loader}} |[x]\rangle|E(x)\rangle \xrightarrow{\text{Discard}} |[x]\rangle.$$

For $y \in \{0,1\}^k$, the `xor(y)` operation is performed by the reversible quantum operation:

$$U_{\oplus y}\colon |i\rangle|b\rangle \mapsto |i\rangle|b \oplus E(y)_i\rangle.$$

The linearity of E implies $U_{\oplus y}|[x]\rangle = |[x \oplus y]\rangle$. The implementation of this procedure may use the same procedure as the encoding. Note that although n ancilla qubits are needed to perform this operation, these qubits are not part of the storage of the data structure and this memory can thus be used briefly for the loading process and then be yielded to other computational processes. For example, if we had multiple such data structure instances, we would only need a single set of n ancilla qubits to XOR values onto each instance by taking turns. However, if this ancilla requirement needs to be reduced, one could choose a binary linear code with enough structure so that bits can be XOR-ed one at a time. Let e_j denote the bitstring with only a single 1 in the j-th position, and 0's everywhere else. Linearity allows us to decompose a bitstring into e_j's and apply the following operation:

$$U_{\oplus e_j}\colon |i\rangle|b\rangle \mapsto |i\rangle|b \oplus E(e_j)_i\rangle.$$

For a code with some mathematical structure, $E(e_j)_i$ could be computed from a small circuit just based on i and j, rather than relying on looking up memory, or encoding arbitrary codewords in deep quantum circuits.

Thus we see our first example of how data stored in a quantum fingerprint can be modified. This is interesting because the data is stored in a very compressed form, which makes most operations impossible but surprisingly, as we have shown, XOR-ing a known bitstring onto these states is one of the operations that is still possible. This data structure can be used for computations relating to error correcting codes, such as in the study of error correcting codes, it could be a memory efficient way of

exploring which patterns of errors produces a trivial syndrome, and is hence a logical error. In a distributed computing setting, this could be used to check consistency between datasets stored on different computers.

Another data structure extending this one can be defined by replacing $\mathbb{Z}_2$ by $\mathbb{Z}_q$ and a linear q-ary code. This yields a vector data structure supporting addition modulo q, for primes q which generalises XOR.

4.3.2 Bitstrings with reverse, flip and append (Reed–Solomon)

Next, we present another data structure for bitstrings, but this one is constructed using the Reed-Solomon code [102] instead, which is a code whose structure is based on polynomials. Even though both data structures encode bitstrings and achieve a similar asymptotic performance, they support different operations and hence cannot be used in place of one another. This data structure has two notable characteristics. Firstly, the Reed-Solomon code is a code over a large alphabet, one that grows linearly with the code dimension. This makes it impractical for communication setting in the large alphabet form, but if this alphabet is simply converted to a binary representation in a naïve way, the code distance of the code can be very poor. However, as discussed above this is not a problem for our data structure construction and at very little overhead, we can put the coordinates of the codeword in superposition without converting it to a binary codeword, and still preserve the code distance. The second notable characteristic is that the Reed-Solomon code has many automorphisms, which allows the design of data structure operations. These two facts seem related, and codes with lots of algebraic structure and larger alphabets seem to have more code automorphisms, than binary codes that are usually constructed using some randomisation.

Datatype	Bitstrings of length k
Space complexity	$\mathcal{O}(\log k)$ qubits
Operations	Reverse, bitwise flip, append bit.
Code used	Reed–Solomon [102] over $\mathbb{F}_q$ with $q = \Theta(k)$

Table 4.3: Summary of the Reed–Solomon bitstring data structure.

Function	Description	Formal Description
`create(x)`	Create an instance storing bitstring $x = x_1x_2\ldots x_k$.	$b \leftarrow x$
`.append(a)`	Append a new bit a to the end of the stored bitstring.	$b \leftarrow ba$
`.reverse()`	Reverse the order of the stored bits.	$b \leftarrow \text{rev}(b)$
`.flip()`	Flip each bit in the stored string (0→1, 1→0).	$b \leftarrow b \oplus 111...1$
`.equals(y)`	Check if the stored bitstring equals y.	Return `true` if $b = y$, else return `false` with high probability and destroy the data structure.
`.equals(`$\|[y]\rangle$`)`	Check if the stored bitstring equals that stored in the state $\|[y]\rangle$.	Return `true` if $b = y$, else return `false` with high probability and destroy both data structures.

Table 4.4: Bitstring with reverse, flip and append ADT.

To encode a bitstring $x = x_1 \ldots x_k$ in this data structure, we first associate it with the polynomial $p_x(z) = \sum_{j=0}^{k-1} x_j z^{k-j} \in \mathbb{F}_q[z]$. Then the quantum state encoding may be defined as:

$$|[x]\rangle = \frac{1}{\sqrt{q}} \sum_{z \in \mathbb{F}_q} |z\rangle\,|p_x(z)\rangle.$$

A similar encoding was proposed in Ref. [59]. Our contribution here is to connect it to the rest of our results, and to show that the data structure also supports modification operations. Note that one change we make is to use polynomials with no constant term, as this is required to keep the operations unitary. This encoding is a superposition of the pairs in a function[2], and we saw it in Section 2.3 as a way to encode functions. Recall that such states are more distinguishable when the functions have a smaller number of pairs in common. In fact, this encoding is also somewhat similar to the *point-value encoding* from Section 2.6. The difference being that what we saw before stored the values of the function in the phase associated with the state, whereas the encoding here uses a second register. Similar to the point-value encoding, for two distinct bitstrings, x and y, we have that $|\langle[y]|[x]\rangle| \leq k/q$, because the degree bound of the polynomials is k and thus the polynomials can only agree on at most k values. For this quantity to be a constant, we need $q \in \Theta(k)$. Note that the

[2] i.e. when the function is viewed as a set of pairs as in Set Theory.

data structure will also store the value k classically (using $\mathcal{O}(\log q)$ bits) so we know the size of the bitstring, which is needed for some operations later. The number of qubits used is $2\lceil \log q \rceil$, so for $q \in \mathcal{O}(k)$, the space complexity of this data structure is $\mathcal{O}(\log k)$. To implement the `create(x)` operation on the bitstring $x = x_1 ... x_k \in \mathbb{F}_2^k$, the fingerprint can be prepared using Horner's Rule [108, Sec. 4.6.4], by alternating the following two subroutines:

$$V_{x_i} : |z\rangle|w\rangle \mapsto |z\rangle|w + x_i\rangle, \text{ and } U_z : |z\rangle|w\rangle \mapsto |z\rangle|w \cdot z\rangle, \text{for } z \neq 0.$$

With these operations,

$$|[x]\rangle = \frac{1}{\sqrt{|F|}} \sum_{z \in F} |z\rangle|p_x(z)\rangle = U_z V_{x_k} U_z V_{x_{k-1}} U_z ... U_z V_{x_1} \frac{1}{\sqrt{|F|}} \sum_{z \in F} |z\rangle|0\rangle.$$

This implementation of `create()` operation in terms of Horner's rule can be seen as first creating the encoding of an empty bitstring and appending bits one at a time. Therefore, we may define an `append(`x_{k+1}`)` operation which is performed by applying the operation $U_z V_{x_{k+1}}$ to the quantum state and incrementing the stored length of the bitstring ($k \mapsto k+1$), as follows:

$$U_z V_{x_{k+1}} |[x]\rangle = \frac{1}{\sqrt{|F|}} \sum_{z \in F} |z\rangle|z \cdot (p_x(z) + x_{k+1})\rangle = \frac{1}{\sqrt{|F|}} \sum_{z \in F} |z\rangle \left| p_{x||x_{k+1}}(z) \right\rangle = |[x||x_{k+1}]\rangle.$$

Another operation supported by the data structure is `flip()`, which flips all the bits in the bitstring, for example, $x = 00110$ is mapped to $\bar{x} = 11001$. To implement this operation note that $p_{\bar{x}}(z) = p_{1^k}(z) - p_x(z)$, where $p_{1^k}(z) = z + ... + z^k$. Also note that a simpler expression for this latter polynomial can be given as $p_{1^k}(z) = \frac{z^{k+1}-1}{z-1} - 1$.

The flip operation can thus be implemented by a procedure such as:

$$\begin{aligned}
&\frac{1}{\sqrt{|F|}}\sum_{z\in F}|z\rangle|p_x(z)\rangle \\
\mapsto&\frac{1}{\sqrt{|F|}}\sum_{z\in F}|z\rangle|p_{1^k}(z)\rangle|p_x(z)\rangle \\
\mapsto&\frac{1}{\sqrt{|F|}}\sum_{z\in F}|z\rangle|p_{1^k}(z)\rangle|p_{1^k}(z)-p_x(z)\rangle \\
\mapsto&\frac{1}{\sqrt{|F|}}\sum_{z\in F}|z\rangle|p_{1^k}(z)-p_x(z)\rangle \\
=&\frac{1}{\sqrt{|F|}}\sum_{z\in F}|z\rangle|p_{\overline{x}}(z)\rangle
\end{aligned}$$

Finally, we present the `reverse()` operation which maps a bitstring to its reverse, like $x = 01011$ to $\mathrm{rev}(x) = 11010$. This operation is notable for relying on a coordinate permutation to work. Define two subroutines:

$$U_{\mathrm{shift}(a)} : |z\rangle|w\rangle \mapsto \begin{cases} |z\rangle|w\cdot z^a\rangle & \text{if } z\neq 0, \\ |z\rangle|w\rangle & \text{if } z = 0. \end{cases}$$

$$U_{\mathrm{mod_inv}} : |z\rangle|w\rangle \mapsto \begin{cases} \left|z^{-1}\right\rangle|w\rangle & \text{if } z\neq 0, \\ |0\rangle|w\rangle & \text{if } z = 0. \end{cases}$$

The operation can be performed by the following steps:

$$\begin{aligned}
&\frac{1}{\sqrt{|F|}}\sum_{z\in F}|z\rangle\left|x_1z^k+x_2z^{k-1}+\cdots+x_{k-1}z^2+x_kz\right\rangle\\
\overset{U_{shift(q-1-(k+1))}}{\mapsto}&\frac{1}{\sqrt{|F|}}\sum_{z\in F}|z\rangle\left|x_1z^{q-1-1}+x_2z^{q-1-2}+\cdots+x_{k-1}z^{q-1-(k-1)}+x_kz^{q-1-k}\right\rangle\\
=&\frac{1}{\sqrt{|F|}}\sum_{z\in F}|z\rangle\left|x_1z^{-1}+x_2z^{-2}+\cdots+x_{k-1}z^{-(k-1)}+x_kz^{-k}\right\rangle\\
\overset{U_{mod_inv}}{\mapsto}&\frac{1}{\sqrt{|F|}}\sum_{z\in F}\left|z^{-1}\right\rangle\left|x_1z^{-1}+x_2z^{-2}+\cdots+x_{k-1}z^{-(k-1)}+x_kz^{-k}\right\rangle\\
=&\frac{1}{\sqrt{|F|}}\sum_{y\in F}|y\rangle\left|x_ky^k+x_{k-1}y^{k-1}+\cdots+x_2y^2+x_1y\right\rangle
\end{aligned}$$

This highlights how coordinate permutations can enable operations on quantum fingerprint states. The coordinate permutation is done by $U_{\text{mod_inv}}$. After the coordinate permutation, a change of variable ($y = z^{-1}$) allows us to rewrite the state in a familiar form. This data structure could be useful in distributed computational settings where a bitstring needs to be updated by concatenating bits to both its front and back. Bits can be concatenated to the front by reversing the bitstring, performing append and reversing it again. A similar data structure can also be defined beyond bitstrings for vectors of integers modulo q. In this case, the `flip` operation is no longer well-defined, but the data structure encoding together with the `append` and `reverse` operations still form a vector of integers data structure.

4.3.3 Polynomials in point–value form (Reed–Solomon)

Datatype	Degree-$< d$ polynomials over $\mathbb{F}_q$
Space complexity	$\mathcal{O}(\log q)$ qubits
Operations	Add polynomial, scale output, shift input, scale input
Code used	Reed–Solomon

Table 4.5: Point-value polynomial data structure.

The previous data structure uses polynomials as a way to encode bitstrings. However, we could also use a similar construction to implement a polynomial abstract data type. Both states have a similar form and can implement similar operations, but some of the operations have a more natural interpretation on bitstrings such as

Function	Description	Formal Description
`create(p0)`	Create an instance storing polynomial $p(z) \in \mathbb{F}_q[z]$.	$p \leftarrow p_0$
`.add(p2)`	Add another polynomial p_2 pointwise.	$p \leftarrow p + p_2$
`.multiply(p2)`	Multiply by a polynomial p_2 that has no roots in $\mathbb{F}_q$.	$p \leftarrow p \cdot p_2$
`.shift(b)`	Shift the input by b.	$p \leftarrow \lambda z.p(z+b)$
`.scale(a)`	Scale the input by $a \neq 0$.	$p \leftarrow \lambda z.p(az)$
`.equals(q)`	Check if the stored polynomial equals q.	Return `true` if $p = q$, else return `false` with high probability and destroy the data structure.
`.equals(|[q]⟩)`	Check if the stored polynomial equals the one stored in the state $\|[q]\rangle$.	Return `true` if $p = q$, else return `false` with high probability and destroy both data structures.

Table 4.6: Polynomial ADT.

`reverse` and `flip`, while others, that we will see in this subsection, have a more natural interpretation on polynomials.

Let $p(z)$ be a polynomial of degree-bound d over finite field $\mathbb{F}_q$. The function state encoding (see section 2.3) prepares the uniform superposition of all field elements together with the polynomial evaluation at each point:

$$|[p]\rangle \;=\; \frac{1}{\sqrt{q}} \sum_{z \in \mathbb{F}_q} |z\rangle |p(z)\rangle.$$

The encodings of distinct polynomials of degree at most d have inner product at most d/q, which is constant if $q = \Theta(d)$. Appendix A describes how finite fields of the right size can be constructed. There are q^d possible polynomials, which would require $\mathcal{O}(d \log q)$ bits to write down explicitly, and the quantum data structure stores them in $\mathcal{O}(\log q)$ qubits. To prepare this state given the coefficients $c_0, \ldots, c_{d-1}$ of $p(z) = c_0 + c_1 z + \cdots + c_{d-1} z^{d-1}$, we can apply Horner's rule similarly to the previous data structure. In addition to the degree-bound d, the actual degree of the polynomial should also be stored in classical description, as it is needed for some of the other operations to ensure that the degree-bound is not exceeded.

The data structure supports addition of a polynomial onto the data structure by the `add` operation. The polynomial to be added can be evaluated at all the

points in superposition by using Horner's rule after which it is added on and finally uncomputed as:

$$\begin{aligned}\frac{1}{\sqrt{q}}\sum_{z\in\mathbb{F}_q}|z\rangle|p(z)\rangle &\mapsto \frac{1}{\sqrt{q}}\sum_{z\in\mathbb{F}_q}|z\rangle|p(z)\rangle|p_2(z)\rangle \\ &\mapsto \frac{1}{\sqrt{q}}\sum_{z\in\mathbb{F}_q}|z\rangle|p(z)+p_2(z)\rangle|p_2(z)\rangle \\ &\mapsto \frac{1}{\sqrt{q}}\sum_{z\in\mathbb{F}_q}|z\rangle|p(z)+p_2(z)\rangle.\end{aligned}$$

The classically stored degree of the polynomial should also be updated if the polynomial that was added has a larger degree than the stored polynomial. Similarly to this, polynomials can be multiplied onto the data structure, however there is a caveats with multiplication, as labelled below:

$$\begin{aligned}\frac{1}{\sqrt{q}}\sum_{z\in\mathbb{F}_q}|z\rangle|p(z)\rangle &\mapsto \frac{1}{\sqrt{q}}\sum_{z\in\mathbb{F}_q}|z\rangle|p(z)\rangle|p_2(z)\rangle \\ &\overset{(*)}{\mapsto} \frac{1}{\sqrt{q}}\sum_{z\in\mathbb{F}_q}|z\rangle|p(z)p_2(z)\rangle|p_2(z)\rangle \\ &\mapsto \frac{1}{\sqrt{q}}\sum_{z\in\mathbb{F}_q}|z\rangle|p(z)p_2(z)\rangle.\end{aligned}$$

The step labelled (*) can only be performed unitarily if p_2 has no roots in $\mathbb{F}_q$, because only in that case do we have $p_2(z)\neq 0$, for all z, making the operation invertible. The classically stored degree of the polynomial should be increased by the degree of the multiplied polynomial.

The data structure supports two coordinate permutation based modification operations. The first shifts the input variable by adding a constant value, thus mapping the function $\lambda z.p(z)$ to $\lambda z.p(z+b)$. To implement this operation, we apply the coordinate permutation unitary,

$$U_{\text{shift}(b)}\colon |z\rangle|w\rangle \mapsto |z-b\rangle|w\rangle,$$

which relabels the superposition indices as:

$$\frac{1}{\sqrt{q}}\sum_{z\in\mathbb{F}_q}|z\rangle|p(z)\rangle \mapsto \frac{1}{\sqrt{q}}\sum_{z\in\mathbb{F}_q}|z-b\rangle|p(z)\rangle = \frac{1}{\sqrt{q}}\sum_{y\in\mathbb{F}_q}|y\rangle|p(y+b)\rangle.$$

Note that the effect of such a shift on functions over reals is quite simply interpreted as translation of the graph of the function. However, over finite fields it can behave more unexpectedly. As an example, consider $p(z) = z^2 + 1$ over $\mathbb{F}_5$, and shift it by $b = 2$. Then

$$p(z+2) = (z+2)^2 + 1 = z^2 + 4z + 5 \equiv z^2 + 4z \pmod 5.$$

So in this example, `shift(2)` has the effect of modifying the data structure in the following way:

$$\frac{1}{\sqrt{5}}\sum_{z=0}^{4}|z\rangle\left|z^2+1\right\rangle \quad\mapsto\quad \frac{1}{\sqrt{5}}\sum_{z=0}^{4}|z\rangle\left|z^2+4z\right\rangle.$$

The other coordinate permutation operation shifts the input variable by multiplying a non-zero constant to it, mapping $\lambda z.p(z)$ to $\lambda z.p(az)$. For $a \neq 0$, we apply:

$$U_{\mathrm{scale}(a)}\colon |z\rangle|w\rangle \mapsto \left|a^{-1}z\right\rangle|w\rangle,$$

which transforms the state as:

$$\frac{1}{\sqrt{q}}\sum_{z\in\mathbb{F}_q}|z\rangle|p(z)\rangle \mapsto \frac{1}{\sqrt{q}}\sum_{z\in\mathbb{F}_q}\left|a^{-1}z\right\rangle|p(z)\rangle = \frac{1}{\sqrt{q}}\sum_{y\in\mathbb{F}_q}|y\rangle|p(ay)\rangle.$$

Once again, while such a transformation applied to functions over reals would stretch or squeeze the graph of the function, its action on functions over a finite field works quite differently, because of the modular arithmetic. Consider $p(z) = z^2 + 1$ over $\mathbb{F}_5$, and scale by $a = 3$. Then,

$$p(3z) = (3z)^2 + 1 = 9z^2 + 1 \equiv 4z^2 + 1 \pmod 5.$$

So the operation `scale(3)` transforms this state in the following way:

$$\frac{1}{\sqrt{5}}\sum_{z=0}^{4}|z\rangle\left|z^2+1\right\rangle \quad\mapsto\quad \frac{1}{\sqrt{5}}\sum_{z=0}^{4}|z\rangle\left|4z^2+1\right\rangle.$$

A data structure for polynomials over finite fields could be useful in cryptography or coding theory. Compared to the previous data structure, this data structure has a much larger number of coordinate permutation operations, because its two coordinate permutation operations are parametrised by an element from the field $\mathbb{F}_q$, unlike in the previous data structure where there were just two particular coordinate permutations. We discuss why this makes the performance achieved by this data structure harder to achieve with classical data structures in Chapter 6. This data structure can be extended to support a polynomial reciprocal operation, which is the analogue of the `reverse` operation from the bitstring data structure. For a polynomial $p(z)$ of degree d, its reciprocal polynomial is $p^*(z) = z^d p(z^{-1})$. However this extension would need us to change the encoding slightly to include the point at infinity in the set of evaluation points, to preserve unitarity, and some of the other operations would then no longer be supported. This data structure could also be generalised to a multivariate polynomial data structure, which would support more operations such as swaps between variables.

4.3.4 Graphs (Reed–Muller RM(2))

Next we show how our framework can also explain the graph state quantum data structure that was presented in section 3.3 and Ref. [64]. While the original presentation is based on the graph state formalism or the Pauli stabiliser formalism, we show a connection to Reed-Muller codes which allows this data structure to be put within our error correcting codes framework.

For each graph $G = (V, E)$ with $|V| = n$, associate the polynomial $P_G : \mathbb{F}_2^n \to \mathbb{F}_2$, defined as:

$$P_G(x_1, \ldots, x_n) = \sum_{(i,j)\in E} x_i x_j.$$

That is, in this polynomial, the monomial $x_i x_j$ appears in the polynomial if and only if there is an edge (i, j) in the graph. If there is a self-loop on vertex i (i.e. an edge from the vertex to itself), the monomial x_i is present in the polynomial. The coefficients are taken over $\mathbb{F}_2$, so they are either 0 (no edge) or 1 (edge present).

Note that the graph state corresponding to a graph $G = (V, E)$ can be written [109] in a function state form with sign encoding as:

$$\left|[G^{\pm}]\right\rangle \;=\; \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{P_G(x)} |x\rangle.$$

To fit more closely to our framework, we can write it in the bit encoding:

$$\left|[G]\right\rangle \;=\; \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \, |P_G(x)\rangle.$$

Written as vectors, $(P_G(x))_{x \in \mathbb{F}_2^n}$, for all the graphs G on n vertices, are the codewords of the Reed-Muller code of order 2 [109]. We illustrate just one operation to show the connection between graph state operations and code automorphisms. Consider the neighbourhood loop toggle operation (`nlt` from section 3.3). Starting from the graph state $\left|[G^{\pm}]\right\rangle$, we apply the neighbourhood loop toggle operation to vertex i by applying the Pauli X operator on qubit i. This achieves the following permutation automorphism:

$$\begin{aligned} X_i \left|[G^{\pm}]\right\rangle &= \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{P_G(x)} |x_1, \ldots, x_{i-1}, \, 1 \oplus x_i, \, x_{i+1}, \ldots, x_n\rangle \\ &= \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{P_G(x_1, \ldots, x_{i-1}, \, 1 \oplus x_i, \, x_{i+1}, \ldots, x_n)} |x\rangle. \end{aligned}$$

Now note that

$$(1 \oplus x_i)\, x_j \;=\; x_j \oplus (x_i x_j),$$

so replacing x_i by $1 \oplus x_i$ in P_G shifts the phase by $\sum_{j \in N(i)} x_j$. Hence, up to a global phase,

$$X_i|G\rangle \equiv \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{P_G(x)+\sum_{j \in N(i)} x_j} |x\rangle = \Big(\prod_{j \in N(i)} Z_j \Big) |G\rangle.$$

This naturally extends to data structures for hypergraphs of order k, using hypergraph states which can be interpreted in this framework through the code family $RM(k)$. This hypergraph data structure will be discussed in the next chapter.

4.4 Discussion

This chapter has explored how classical error correcting codes can be used to construct quantum data structures. Quantum fingerprinting had shown how the distance properties of error correcting codes provide guarantees that quantum encodings of different messages remain nearly orthogonal, which enables equality testing with a high probability of success. We extended quantum fingerprints by showing that data modification operations can be implemented by using code automorphisms to define reversible operations that are physically realisable and change the underlying value of the data structure. In doing so we obtain a data structure design framework that generalises quantum fingerprinting, allows for the design of new data structures by leveraging code families and automorphisms, and reveals a connection with previously studied graph-state data structures.

These encodings achieve a space advantage by exploiting the fact that a quantum register can exist in a large number of possible states, and in particular $\mathcal{O}(\log k)$ qubits is sufficient space to pick a set of 2^k quantum states which have small pairwise inner products. This allows a one-to-one encoding of the data into a much smaller quantum space than is possible classically, while preserving the ability to perform equality checks, and some modification operations. However, implementation details of the operations and data loading can affect the practicality of the schemes we discussed. Moreover, the successful application of these data structures to achieve a space advantage depends on the specific computational task. It is possible that randomised algorithms, that do not store the data in a one-for-one encoding are able

to solve computational tasks with space complexities as good as the quantum data structure. We explore this in the next chapter.

By treating codes as modular components, we can "plug in" different code families and design new data structures, effectively mining the coding theory literature to generate quantum data structures. Our construction also relaxes many of the constraints that make code design difficult in the classical setting. We do not require efficient decoding, nor do we restrict ourselves to binary alphabets, nor do we insist on constant or particularly high rate. In fact, large-alphabet codes that are awkward for classical communication are particularly attractive here, since their overhead in quantum space is only logarithmic.

The biggest limitation of these data structures is that they are not universal data structures and that they have a limited set of operations. The only readout operation is equality checking with one-sided error. The reversible operations that can be performed rely on symmetries of the underlying code, so the set of supported modifications is relatively narrow compared to universal data structures. Thus the application of these data structures will be limited to specific problems where just the available set of operations is sufficient.

There is an interesting similarity between this work and the design of quantum error correcting codes. Both settings involve embedding information into Hilbert spaces, and in both cases one studies which logical transformations can be performed on the stored information by symmetry-based operations on the codespace. The key difference is that in quantum error correction, the encoding space is larger than the message space and the goal is to increase error resilience, whereas in quantum data structures the encoding space uses fewer qubits than bits in the message and we seek compact encodings.

Future work could look into the following questions:

- Can these data structures be given other forms of readout than equality comparison? Perhaps these could be related to procedures that perform sampling from distributions that are hard to sample from classically?

- Are there code families, particularly non-linear or large-alphabet ones, whose automorphisms yield operations that genuinely resist classical simulation?
- Which data structures among these have efficient circuits for data loading and modification? We could implement them on early fault-tolerant quantum computers to work out the end-to-end engineering of such algorithms.
- Is there a way to connect this framework more directly with quantum data structures where the inner products come from destructive interference of phases (see section 3.4) rather than Hamming distance, so that the different approaches to designing quantum data structures can be unified?
- Which algorithms use quantum data structures to achieve a quantum space or time advantage?

Chapter 5

Quantum Data Structures from Hypergraph states

This chapter looks to define quantum data structures within the graph and hypergraph state formalisms. We first propose a *list of bitstrings* data structure built on top of graph states. We then generalise the graph state data structure to the hypergraph state data structure, which allows for arbitrary polynomial advantages, and may thus be more practical than the quadratic space advantage achieved by the graph state data structure. We also show how the hypergraph data structure allows the definition of a polynomial data structure which helps give a conceptual explanation of the Bernstein-Vazirani algorithm and its generalisation, the polynomial learning algorithm. Finally we discuss further research directions such as stabilizer formalisms that may be explored in future works to define more efficient quantum data structures.

5.1 Encoding a list of bitstrings as a graph state

In this section we present a data structure for the *list of bitstrings* data type. The data structure represents the list of bitstrings as a bipartite graph state.

As we saw in chapter 3, the use of graph states to store graphs results in a quantum data structure that only supports a non-universal set of graph operations but achieves a quadratic space advantage. As this data structure has a lot of operations, and achieves a clear space advantage, a natural question is how this data structure can be extended to define other data structures. We show how a list of bitstrings data

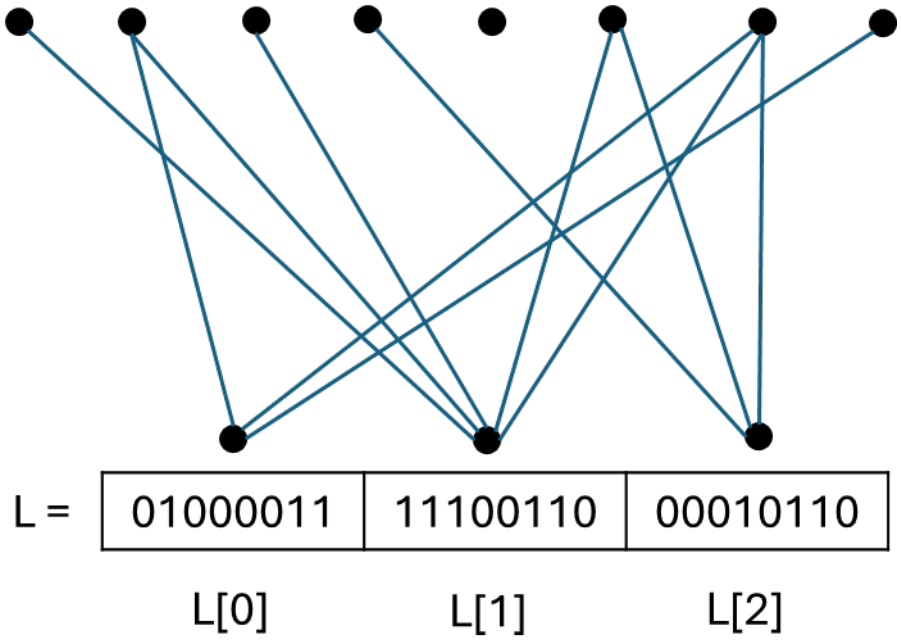


Figure 5.1: Example of the graph state on 8 + 3 = 11 qubits corresponding to the list of bitstrings, $L = [01000011, 11100110, 00010110]$.

structure can be defined using graph states. The construction uses a bipartite graph, where each vertex in one partition corresponds to a place value of the bitstrings, and each vertex in the other partition corresponds to an item in the list.

5.1.1 Data structure encoding

Each list of k-bit bitstrings $L = [s_1, ..., s_n]$ is encoded by the graph state corresponding to the bipartite graph of k and n vertices,

$$G = (\{v_1, ..., v_k\}, \{u_1, ..., u_n\}, \{(u_i, v_j) \text{ if bit } j \text{ is set in } s_i\}).$$

Figure 5.1 shows an example of this encoding.

This graph state can be stored in an indexed memory by storing the k place value qubits in sequence, followed by the n element qubits in sequence. The numbers k and n are stored in classical description so we know which qubits in the register are the k place value qubits and the n list element qubits.

An information-theoretically optimal classical data structure for storing n bitstrings of length k uses $\mathcal{O}(nk)$ bits. It simply stores the bitstrings in a sequence. This

classical data structure is universal. The quantum data structure we present here supports only the operations shown in Table 5.1 but uses just $\mathcal{O}(n+k)$ bits.

5.1.2 Data structure operations

This section explains how the stored list can be transformed in terms of graph state operations from section 3.3. The operations are summarised in Table 5.1

The data structure can be initialised as an empty list with bitstring length k by preparing the state $|+\rangle^{\otimes k}$. As there are no elements in the list, there are no qubits in the other partition (i.e. $n = 0$). The `create(k)` operation that creates such an empty list should also store the values k and $n = 0$ classically for tracking how the qubits in memory are to be interpreted.

Another way to create the list would be to start from an existing and explicitly known non-empty list of bitstrings and to build the quantum state corresponding to it. This can easily be implemented using a list insertion operation, which we define next.

The operation `insert(s,i)` takes a bitstring s and inserts it into the position i in the list. It should first check that `s` has k bits and that `i` is in the range 0 to n. Then, it can prepare a new qubit in the $|+\rangle$ state, and permute the qubits so that this new qubit is now in the i-th position amongst the list element qubits. Finally for each bit s_j that is 1, a CZ gate (i.e. an `edge_toggle` operation in the graph state data structure) is applied between the j-th place value qubit and the new qubit.

The CZ gates or `edge_toggle` graph operations also enable the implementation of a bitstring XOR operation. A known k-bit bitstring b can be XORed onto the list element at the i-th position by the operation `xor_element(i,b)` which applies CZ gates between the i-th list element qubit and the k place value qubits if b has 1 in that place. Edge toggling is not a very usual operation in graph data structures, but in this data structure it is used to implement XOR which is a more common and natural operation.

SWAP gates allow permuting elements of the list in this data structure if applied among the n list element qubits. We can also define an operation that permutes the

place value qubits. This would permute the coordinates of all bitstrings in the list uniformly.

Since the graph state data structure for graphs is able to represent self-loops in the graph, it is possible to encode one more bit of information associated with each qubit in our list data structure. These extra bits can be used in our list data structure as binary flags on each element. One could also similarly define a flag for each place value.

The operation `toggle_flag(i)` toggles the flag for the i-th bitstring in the list. At the gate-level, this corresponds to a Z gate applied to the i-th element qubit.

The neighbourhood loop toggle operation in graphs (`nlt`, see Table 3.3) also allows us to toggle the flags of many list elements by just applying an X gate to the j-th bit position qubit. This is equivalent to toggling the flag for all elements that have the j-th bit equal to 1.

It is also possible to delete elements from the list by using the `delete_vertex` operation in graphs. Note that when `delete_vertex` fails in graph states, it does still delete the vertex, but it can be interpreted as applying the `nlt` operation before deleting the vertex or alternatively, that it applies a Z gate on neighbouring vertex and toggles the self-loops on those vertices. If we have the classical representation of the graph, we can correct this, but we can assume we do not in many cases where we are trying to get a space advantage. Therefore, the `delete_vertex` operation has a 0.5 probability of making a graph state data structure unusable for equality comparisons. In the list of bitstrings data structure, this also happens, but because the graph is bipartite, we know that none of the added loops are on any of the list element qubits. As a result, while the `equals_list` operation is disabled after an unlucky `delete` operation, we can still use the `equal_elements` operation as per normal.

Finally there are three readout operations on this data structure. Two lists can be compared for equality in two different ways. If one list is stored in the data structure and the other is known explicitly, then they can be compared using the quantum state equality procedure as defined in section 2.2. Otherwise, if both lists are encoded

in this data structure, they can be compared for equality by the SWAP test. The data structure also allows us to test if two list elements are equal by means of the `vertex_equals` operation on the graph state data structure.

Function	Description	Formal Description
`create(k)`	Create an empty bitstring list with default bitstring-length k.	$L \leftarrow \emptyset$
`.append(s)`	Insert a bitstring into the list.	$L \leftarrow L\|s$
`.equals_list(L2)`	Check equality of the stored list with another bitstring list.	Return whether $L = L2$
`.equal_elements(i, j)`	Check equality of two elements at positions `i` and `j`.	Return whether $L[i] = L[j]$
`.xor_element(i, b)`	XOR a constant bitstring `b` onto element at position `i`.	$L[i] \leftarrow L[i] \oplus b$
`.permute_elements(p)`	Permute elements of the list according to permutation `p`.	$L \leftarrow [L[p(i)]$ for $i \in \{1,2,\ldots,\|L\|\}]$
`.permute_coordinates(p)`	Permute coordinates of all bitstrings according to permutation `p`.	$L[i] \leftarrow [L[i][p(j)]$ for $j \in \{1,2,\ldots,k\}]$ for all i
`.toggle_flag(i)`	Toggle the flag of element i.	$F[i] \leftarrow F[i] \oplus 1$
`.toggle_if_set(bit)`	Toggle the flag of bitstrings where the specified bit is set.	For all i where $L[i][bit] = 1$, toggle flag of $L[i]$
`.insert_bit(j)`	Create a new coordinate in the j-th position in all the bitstrings with initial value 0	$L[i] \leftarrow L[i][:j]\|0\|L[i][j:]$, for all i
`.delete(i)`	Delete bitstring at position i from the list. If measurement outcome is 1, the `equals_list` operation is now disabled.	$L \leftarrow L[:i]\|L[i+1:]$

Table 5.1: The list of operations supported by the graph state list-of-bitstrings data structure.

5.1.3 Discussion

Like many quantum algorithms, data structures and protocols that seek to articulate where quantum information technologies may be useful, this data structure was motivated from what quantum computation makes possible, rather than what is useful or needed from quantum computers. This explains why the data structure may not seem immediately useful. Nonetheless, we think this data structure could

be useful in two imagined settings. Firstly, it can be used for equality testing and tracking of a large set of files that are changing with permutation and XOR updates. This would be especially powerful if there are many such updates followed by an equality test of two list elements. This is because the data structure can perform these operations efficiently (in $\mathcal{O}(1)$ time) and no classical data structure can do the same [64].

The space advantage from nk to $n+k$ would be most advantageous for regimes of large n and k. For example, if this data structure is used to track 1 million files each of roughly a gigabyte, then it will do in a few million qubits (in theory, ignoring overheads of fault-tolerance and system architecture), what would need the largest data centres in the world to use all their memory. This advantage is modest, especially since the overheads and cost of quantum computer will be substantial. Moreover it is unclear what practical utility this data structure will have and even if this non-universal set of operations is interesting for any practical application, whether that will be in the regimes of n and k that are classically infeasible or for which there would be a quantum advantage after accounting for the overheads of quantum computation. Therefore it is unlikely that such large files will be processed in this way using millions of qubits just for the quantum state to be destroyed by one equality comparison.

The second imagined application is to perform computations involved in the simulation and design of error correcting codes. The data structure naturally stores a Tanner graph: the list of bitstrings may be viewed as a list of parity checks, and the place value qubits can represent the physical bits on which the code is built. The data structure allows three operations that are particularly relevant to error correcting codes—XOR, which corresponds to changing the code space by changing the parity checks, permutations of place value qubits which corresponds to automorphisms of the code, and toggling flags of the list elements based on set bits which corresponds to flipping a physical bit and automatically updating the syndrome of the code.

While this may not be useful in decoding error correcting codes due to the limited readout capability, it could help in computations that help study and design

codes, especially since this is often a very computationally intensive process, but one that only needs to be done once to find the codes. The `toggle_if_set(i)` operation allows us to toggle the flag of those parity checks whose result would be flipped if bit i was flipped. This can for example allow us to efficiently check if a pattern of errors corresponds to an undetectable error in a code.

This section has shown how simple constructions on top of an existing quantum data structure can give completely new possibilities for a standard datatype such as a list of bitstrings. We will see in the next section how extending to hypergraph data structures allows us to achieve bigger polynomial advantages which are more practical, including for a different list of bitstrings data structure.

5.2 Hypergraph data structure

The graph state formalism [65, 66] can be used to define a natural quantum data structure for the graph data type [64]. Analogously, we will use the hypergraph state formalism [110, 111] to define a natural data structure for the hypergraph data type. We will see that the hypergraph data structures allow us to achieve arbitrary polynomial space advantages over all classical data structures on some computational tasks. The degree of the polynomial advantage achieved depends on the maximum hyperedge rank allowed in the hypergraph data structure. However, for a fixed target error probability, the space complexity grows exponentially in the rank, so this advantage is practical in the regime of small rank hypergraphs with large numbers of vertices.

5.2.1 Classical data structures

Hypergraphs are a generalisation of graphs in which edges can be incident on not just two but on any number of vertices. A hypergraph is formally a tuple of two sets $H = (V, E)$ where V is the set of vertices and $E \subseteq \mathcal{P}(V)$ is the set of "hyperedges". The number of vertices that a hyperedge is incident on is called the rank of the hyperedge. In many cases it is useful to focus on hypergraphs of fixed or bounded rank. A hypergraph with all edges of rank k or less is called a rank-k hypergraph. If all hyperedges have rank equal to k, it is called a k-uniform hypergraph. There are

many variants of hypergraphs such as weighted-, coloured- and directed- hypergraphs. Hypergraphs find applications in social network analysis, identifying correlations in clinical datasets, and various other studies in biochemistry and biological systems [112].

Broadly speaking, data structures and algorithms for hypergraphs are more computationally resource intensive than those for graphs, because there are 2^n possible hyperedges on n vertices but only $\binom{n}{2}$ possible edges. A natural data structure for storing hypergraphs is the hyperedge list.

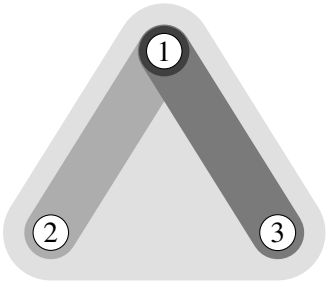


Figure 5.2: The hypergraph $H = (V,E)$ with vertex set $V = \{1,2,3\}$ and edge set $E = \big\{\{1\},\{1,2\},\{1,3\},\{1,2,3\}\big\}$. Each hyperedge is drawn as a shaded region enclosing exactly its own vertices.

Hyperedge lists are good when working with "sparse" hypergraphs. This is when the number of hyperedges $|E|$ is far smaller than the number of possible hyperedges, $|\mathcal{P}(V)| = 2^{|V|}$. Viewed from the lens of coding theory, the hyperedge list data structure is the encoding that gives shorter encodings to the more likely to occur sparse hypergraphs while giving larger encodings to the less likely to occur dense hypergraphs. If dense hypergraphs are unlikely to be encoded in the data type, then the hyperedge list is a good choice.

On the other hand, the "one-hot" encoding would be an unbiased encoding of the hypergraph. In this encoding, a bit is stored for every possible hyperedge, indicating whether it is present or absent in the hypergraph. Since there are $2^{|V|}$ possible hyperedges, each n vertex hypergraph needs 2^n bits to be described in this encoding. This is quickly prohibitive for anything more than a few tens of vertices. As this is the information theoretically optimal data structure in the unbiased case, the only way to do better is to utilise some knowledge or structure of what kind of

hyperedges are more likely to be represented in the data structure (such as a sparsity assumption).

We may consider the one-hot encoding of rank-k hypergraphs, which need only $1+n+\binom{n}{2}+\ldots+\binom{n}{k} \in \mathcal{O}(n^k)$ bits to encode n vertex hypergraphs. This is considerably better than 2^n but it is prohibitive beyond 100 vertices for any values of k beyond 5.

In the rest of the section we will show how hypergraph states can serve as a data structure for hypergraphs and fill a gap by providing a data structure for dense rank-k hypergraphs in regimes of n and k that are classically intractable, but quantumly efficient. These data structure are not universal, but nonetheless provide computational possibilities that are otherwise impracticable with classical computing. For example, it would be practical to implement some computations on rank-7 hypergraphs of 5000 vertices using this quantum data structure, but this would be beyond the reach of classical data structures as $5000^7 > 10^{25}$.

5.2.2 Data structure definition

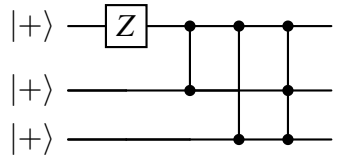


Figure 5.3: Preparation of the three-qubit hypergraph state $|H\rangle$ associated with the hypergraph $H=(V,E)$ on $V=\{1,2,3\}$ with hyperedge set $E=\big\{\{1\},\{1,2\},\{1,3\},\{1,2,3\}\big\}$. Each hyperedge $S\in E$ contributes a generalised controlled-Z gate $C^{|S|-1}Z$ acting on the qubits in S, applied to $|+\rangle^{\otimes 3}$: the singleton $\{1\}$ gives Z_1, the two ordinary edges give $CZ_{1,2}$ and $CZ_{1,3}$, and the three-vertex hyperedge gives $CCZ_{1,2,3}$.

In this data structure, every hypergraph

$$H=(V=\{1,\ldots,n\},\, E\subseteq\mathcal{P}(V)) \tag{5.1}$$

is encoded as the hypergraph state:

$$|H\rangle=\left(\prod_{S\in E} C^{|S|-1}Z_S\right)|+\rangle^{\otimes n}. \tag{5.2}$$

For distinguishability of the data structure values, we need an upper bound on the fidelity between their quantum states. Let H_1 and H_2 be two hypergraphs on n vertices. Note that:

$$|\langle [H_2]|[H_1]\rangle|^2 = \left| \langle +|^{\otimes n} \left(\prod_{S \in E_{H_2}} C^{|S|-1} Z_S \right) \left(\prod_{S \in E_{H_1}} C^{|S|-1} Z_S \right) |+\rangle^{\otimes n} \right|^2 \tag{5.3}$$

$$= \left| \langle +|^{\otimes n} \left(\prod_{S \in E_{H_1} \oplus E_{H_2}} C^{|S|-1} Z_S \right) |+\rangle^{\otimes n} \right|^2 \tag{5.4}$$

$$= |\langle [H_\emptyset]|[H_1 \oplus H_2]\rangle|^2, \tag{5.5}$$

because C^kZ gates are self-inverse. Note $H_1 \oplus H_2$ denotes the hypergraph $(V, E_{H_1} \oplus E_{H_2})$, and $H_\emptyset$ denotes $(V, \emptyset)$.

Hypergraph states are REW states with phases corresponding to Reed-Muller codewords [110, 109]. Specifically, a rank-k hypergraph state has phases corresponding to codewords of $\mathrm{RM}(k,n)$ where the symbols are $+1$ and -1 instead of 0 and 1. This helps us bound the inner products in terms of the known code distance of Reed-Muller code $\mathrm{RM}(k,n)$.

Since $|+\rangle^{\otimes n}$ is a REW state that has all phases $+1$, it corresponds to the codeword of all 0s. Therefore:

$$\begin{aligned} |\langle [H_\emptyset]|[H_1 \oplus H_2]\rangle|^2 &= \left[\frac{1}{2^n} \left(+1 \cdot (2^n - w) + (-1) \cdot w \right) \right]^2 \\ &= \left(1 - \frac{2w}{2^n} \right)^2, \end{aligned}$$

where w is the Hamming weight of the Reed-Muller codeword associated with the state $|[H_1 \oplus H_2]\rangle$. The minimum Hamming weight of a non-zero codeword is equal to the code distance, which is 2^{n-k} for $\mathrm{RM}(k,n)$. Thus, if $H_1 \oplus H_2$ has rank k:

$$\begin{aligned} |\langle [H_2]|[H_1]\rangle|^2 &\leq \left(1 - \frac{2^{n-k+1}}{2^n} \right)^2 \\ &= \left(1 - 2^{1-k} \right)^2. \end{aligned}$$

If we use the data structure to only encode hypergraphs of a preset rank k, then the symmetric difference of the graphs will also be of rank k, so this fidelity bound will hold. To suppress the fidelity for large values of k, an encoding that stores H as $|H\rangle^{\otimes c}$ can be used. That is, c copies of the hypergraph state are stored. Then, the fidelity of two different hypergraph encodings is:

$$|\langle [H_2]|[H_1]\rangle|^{2c} \leq \left(1 - 2^{1-k}\right)^{2c}. \tag{5.6}$$

Table 5.2 shows the bit and logical qubit counts for the classical one-hot encoding and the quantum hypergraph data structure, as well as the pairwise fidelity bound (i.e. error probability) for some sample values of n, k, and c.

Vertices	Rank	Copies	Bits (One-Hot)	Qubits (Hypergraph state)	Error prob.
n	k	c	$\sum_{i=0}^{k} \binom{n}{i}$	cn	$\left(1 - 2^{1-k}\right)^{2c}$
100	3	6	1.67×10^5	600	0.0317
100	5	25	7.94×10^7	2500	0.0397
1000	5	25	8.29×10^{12}	25 000	0.0397
5000	7	100	1.55×10^{22}	500 000	0.0429
10 000	10	1000	2.75×10^{33}	1.00×10^7	0.0200
50 000	15	25 000	2.33×10^{58}	1.25×10^9	0.0473

Table 5.2: Space comparison for classical one-hot encoding and hypergraph state encoding at various scales.

To keep the fidelity bounded by a constant ε, we need $c \in \Omega(2^k)$, since

$$\left(1 - 2^{1-k}\right)^{2c} \approx e^{2c\left(-2^{1-k}\right)}.$$

Therefore, for bounded error, our data structure uses $O(2^k n)$ qubits to encode rank k hypergraphs on n vertices, which would require $O(n^k)$ bits in the one-hot encoding. For any fixed k, this is a k-th order polynomial advantage in space.

Having understood the encoding, and the extent of space advantage, we now turn to the operations supported by the data structure. The operations are summarised in Table 5.3. The data structure can be initialised as an empty hypergraph on n

vertices by preparing the state $|+\rangle^{\otimes n}$. This corresponds to the hypergraph with no hyperedges. The `create(n)` operation should also store the value n classically for tracking how many qubits are in the register.

The basic modification operation is `toggle_edge(`$i_1,\ldots,i_m$`)`, which toggles the presence of a hyperedge incident on vertices $i_1,\ldots,i_m$. At the gate level, this corresponds to applying a generalised controlled-Z gate $C^{m-1}Z$ on the m qubits. For the special case of a singleton hyperedge ($m = 1$), this reduces to a single-qubit Z gate. This operation is the hypergraph analogue of the `edge_toggle` operation in graph states.

The `toggle_link(i)` operation has the effect of toggling the hyperedges in the neighbourhood (aka "link") of i. This is the set of hyperedges that contain i, but with i removed, i.e. $L_H(i) = \{e - \{i\} | e \in E_H, i \in e\}$. This can be implemented by applying a single X gate to qubit i. This generalises the neighbourhood loop toggle operation on graph state.. This is an efficient operation—classically, toggling all hyperedges containing a vertex would require iterating through the entire hyperedge list, but here it is achieved in $\mathcal{O}(1)$ time.

The `tl2(i,j)` operation (toggle link of j controlled on i) applies the transformation that toggles hyperedges of the form $e \cup \{i\}$ for each hyperedge e in the link of vertex j. At the gate level, this is implemented by a $\text{CNOT}_{i,j}$ gate. This operation is useful for manipulating the local structure of the hypergraph around specific vertices.

Vertex permutations are supported via the `permute(`σ`)` operation, which relabels vertices according to a permutation σ. Virtually, this can be achieved in $\mathcal{O}(1)$ time by updating a classical mapping from logical to physical qubits. Physically, if the qubits must be rearranged in memory, this requires a SWAP network with $\mathcal{O}(n)$ gates.

For readout, the `vertex_equals(i,j)` operation checks whether two vertices have identical neighbourhoods (excluding each other). This is implemented via a SWAP test on qubits i and j. The operation has one-sided error and requires $\mathcal{O}(2^k)$ copies for bounded error probability when the hypergraph has rank at most k.

The `delete(i)` operation removes vertex i from the hypergraph, along with all hyperedges incident on it. This is implemented by measuring qubit i in the Z-basis. The operation succeeds (producing the correct post-measurement state) when the outcome is $|0\rangle$, which occurs with probability 0.5. On outcome $|1\rangle$, the operation still deletes the vertex but introduces unwanted phase flips on neighbouring vertices, analogous to the failure mode of vertex deletion in graph states.

Finally, there are two equality testing operations. The `equals(H')` operation compares the stored hypergraph with another hypergraph H' that may be given explicitly or encoded in another quantum register. If H' is known explicitly, quantum state equality (QSE, see section 2.2) can be used. The `equals(`$|H'\rangle$`)` operation compares two quantum-encoded hypergraphs via the SWAP test. Both operations have one-sided error and require $\mathcal{O}(2^k)$ copies for bounded error probability on rank-k hypergraphs.

Operation	Formal Description	Implementation	Complexity
`create(n)`	$H \leftarrow (\{1,\ldots,n\},\emptyset)$	Prepare $\|+\rangle^{\otimes n}$	$O(V)$ space
`.toggle_edge(`$i_1,\ldots,i_m$`)`	$H \to (V, E \oplus \{\{i_1,\ldots,i_m\}\})$	$C^{m-1}Z_{i_1\ldots i_m}$, or Z_i if $m=1$	$O(1)$
`.toggle_link(i)`	$H \to (V, E \oplus L_H(i))$	X_i	$O(1)$
`.tl2(i,j)`	$H \to (V, E \oplus \{T \cup \{i\} \mid T \in L(j)\})$	$\mathrm{CNOT}_{i,j}$	$O(1)$
`.permute(`σ`)`	$H \to (V, \{\sigma(e) \mid e \in E\})$	Relabel qubits, or a SWAP network	$O(1)$ virtual, $O(V)$ physical
`.vertex_equals(i,j)`	Output $L_H(i)\|_{\setminus j} == L_H(j)\|_{\setminus i}$	SWAP test on qubits i and j	$O(1)$, $O(2^k)$ copies, one-sided error
`.delete(i)`	$H \to (V \setminus i, E\|_{V\setminus i})$	Z meas., success on $\|0\rangle$	$O(1)$ and 0.5 failure prob.
`.equals(H')`	Output $H == H'$	QSE (sec. 2.2)	$O(E)$, $O(2^k)$ copies, one-sided error
`.equals(`$|H'\rangle$`)`	Output $H == H'$	SWAP test	$O(V)$, $O(2^k)$ copies, one-sided error

Table 5.3: Rank-k hypergraph data structure implemented as a quantum hypergraph state. Here $L_H(i) = \{e \setminus \{i\} \mid e \in E_H, i \in e\}$ is the link of vertex i, $L(i)|_{\setminus j} = \{e \in L(i) \mid j \notin e\}$, and $E|_W = \{e \in E \mid e \subseteq W\}$.

5.3 A polynomial data structure from hypergraph states

In this section we define a data structure for multivariate polynomials over $\mathbb{F}_2$ by using hypergraph states and show how this data structure explains the Bernstein-Vazirani algorithm [113] and one of its generalisations, the polynomial learning algorithm [114]. While the polynomial learning algorithm is lesser-known, we

think that this data structure presentation makes it easier to include it in quantum algorithms courses, as it requires much less algebraic manipulation to teach and understand.

This data structure encodes polynomials of the form $p : \mathbb{F}_2^n \to \mathbb{F}_2$. Note that any function of type $\mathbb{F}_2^n \to \mathbb{F}_2$ can be stated as a polynomial [115]. We choose to call this a polynomial data structure rather than a function data structure nonetheless because the supported operations and the space complexity of the data structure are more easily interpreted when this function is viewed as a polynomial. Such polynomials and their data structures are important in coding theory and cryptography [116].

A simple classical data structure would store the list of monomials in the polynomial. Just like for hypergraph data structures, this monomial list encoding is good for "sparse polynomials", i.e. those with few monomials, but it can take up very large space if the stored polynomial is instead "dense". In the dense case, it may be better to simply store a one-hot encoding of which monomials are present in the function. In the case of polynomials another option is to store its evaluation on every input, or its truth table.

For example, $p(x_1,x_2,x_3) = x_1 + x_1x_2 + x_1x_3 + x_1x_2x_3$ can be stored as either:

1. Monomial List: The list `[(1,), (1,2), (1,3), (1,2,3)]` indicating the monomials present in the polynomial.

2. Point-value encoding or Truth table: The list $[p(000), p(001), ..., p(111)]$ which evaluates to [0,0,0,0,1,0,0,0].

3. One-hot monomial encoding: The list [0,1,0,0,1,1,0,1] which when dot-producted with the vector $[1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1x_2x_3]$, gives the function p.

Other data structures may also be used depending on what special structure is known about the function p. For example, one may notice that the truth-table for p has very few 1s and for such a function, we may wish to simply enumerate the set $f^{-1}(1)$, as a set or list of bitstrings. Another possibility is to store the Walsh-Hadamard spectrum of the function if it is known to be sparse.

We now present our non-universal quantum data structure for this datatype, which is based on hypergraph states. The key idea is to view the polynomial as a hypergraph by having as many variables as vertices, and associating monomials with hyperedges incident on all the vertices corresponding to the variables in that monomial. For example, the previous example p would be associated with the hypergraph $H = (\{1,2,3\},\{\{1\},\{1,2\},\{1,3\},\{1,2,3\}\})$ (see Figure 5.2)

In general the polynomial,

$$p(x_1,\ldots,x_n) = \sum_{S\subseteq\{1,\ldots,n\}} \left(c_S \prod_{i\in S} x_i \right)$$

is associated to the hypergraph,

$$H = (V = \{1,\ldots,n\}, E = \{S \subseteq \{1,\ldots,n\} | c_S = 1\}).$$

The hypergraph state for this polynomial can be constructed by the circuit shown in Figure 5.3.

In general, p can be encoded by the hypergraph state

$$\left(\prod_{S\in E_H} C^{|S|-1} Z_S \right) |+\rangle^{\otimes n},$$

where H is the hypergraph corresponding to p as described above.

Note that this data structure always uses n qubits to store a polynomial on n input variables. This makes it particularly efficient for encoding "dense" polynomials - i.e. those with many monomials in them.

As with quantum data structures we have seen so far, we have to carefully consider what readout operations are possible in this data structure. Arbitrary functions encoded in this way will use hyperedges of high order. As we have seen in the previous section, this leads to inner products close to 1 between distinct hypergraphs. Therefore, for ease of distinguishability between states encoding distinct polynomials, this data structure should be limited to polynomials of a bounded degree. The bound on the fidelity of the quantum states encoding two distinct polynomials of

degree at most k is $\left(1-2^{1-k}\right)^2$, which is also the bound for hypergraph states of rank k. This shows that our data structure can encode polynomials of a bounded degree and check for equality with bounded error probability, by quantum state equality (see 2.2) or the SWAP test.

Next, we show the modification operations supported by this data structure. The `toggle_edge` operation on hypergraphs can be used to add monomials to the polynomial. For any monomial $x_{i_1}x_{i_2}\ldots x_{i_k}$, we can map $p \mapsto p + x_{i_1}x_{i_2}\ldots x_{i_k}$, by applying $C^{k-1}Z_{i_1 i_2 \ldots i_k}$ to the data structure. Note that the `toggle_edge` operation is not very natural on hypergraphs, in the sense that often algorithms require operations that add or remove edges or hyperedges; it is rarer that toggling an edge or hyperedge is what the algorithm requires. However, when that toggle operation is applied here in the encoding of polynomials, it has a more natural interpretation as the addition of monomials. This is because in the field $\mathbb{F}_2$, adding a monomial toggles its presence in the polynomial since $2 \equiv 0 \mod 2$ and so products of variables can only have the coefficients 0 or 1.

We can also add polynomials onto the stored polynomial in one of two ways. If the polynomial to be added is known explicitly, then each of its monomials can be added onto the data structure one by one. Otherwise, if the polynomial to be added can be accessed by an oracular call, then the oracle can be applied to the data structure.

The `toggle_link` operation, which was also unnatural in hypergraphs has a more natural action on the polynomial data structure. It allows us to add a partial derivative of the stored polynomial to itself. That is, it allows us to perform the map $p \mapsto p + \frac{\partial}{\partial x_i} p$ for our choice of x_i.

Measuring a qubit in the Z-basis gave us a deletion operation which has some failure probability. Here it can be interpreted as an operation that randomly substitutes either 0 or 1 into the polynomial. Given a number of copies this operation also allows us to substitute a few variables in the way we want by rejecting failed substitutions.

It is also possible to add to the stored polynomial a modified polynomial where all instances of a variable are replaced. That is, $p \mapsto p + x_i \frac{\partial}{\partial x_i} p$. In fact, the replacement can be done by any monomial, so we can also implement the operation $p \mapsto x_{i_1} \dots x_{i_k} \frac{\partial}{\partial x_i} p$. This can be done by the `tl2` operation in hypergraphs.

Other than checking for equality between two polynomials, another readout operation, `read_linear` can be defined for this polynomial data structure, that works only when we know that the polynomial is of degree 1. That is, if we know that the stored polynomial is of the form $p(x) = c \cdot x$ for some $c \in \mathbb{F}_2^n$ where x denotes the variable vector $(x_1, \dots, x_n) \in \mathbb{F}_2^n$, then this readout operation returns c. This uses the fact that the hypergraph state corresponding to linear functions are just tensor products of $|+\rangle$ and $Z|+\rangle = |-\rangle$ one-qubit states, depending on whether that variable does not or does appear in the linear function respectively. So, measuring in the X-basis helps us read out whether each variable appears in the linear function or not. This operation can be used in promise settings and we will see one example of this in the next section. Given multiple copies of the state, the `read_linear` operation can also be used to check whether a function is linear by performing the operation multiple times and seeing if consistent results are obtained. For low-degree but non-linear polynomials, only a few copies will be needed to detect that they are not linear.

The ADT and data structure operations corresponding to this polynomial data structure is shown in Table 5.4. It is a remarkable near miss that many of the operations needed to do calculations relevant to breaking lattice based cryptography can be implemented using this data structure [117], but since not all of them can be, this does not at present give any quantum algorithms for weakening lattice based cryptography.

5.3.1 Polynomial Learning explained with data structures

In this subsection, we show how the data structure from the previous section allows us to easily explain the Bernstein-Vazirani algorithm, and more notably, its generalisation, the polynomial learning algorithm [114]. This latter algorithm is not very popular, but we think it is an elegant algorithm in the toolkit of quantum algorithms

Operation	**Formal Description**	**Quantum State**
`create(n)`	Initialise $p(x_1,\ldots,x_n)=0$.	$\|\psi\rangle \leftarrow \|+\rangle^{\otimes n}$
`.add_monomial(`$i_1,\ldots,i_k$`)`	$p \leftarrow p + x_{i_1}\cdots x_{i_k}$	$\|\psi\rangle \leftarrow C^{k-1}Z_{i_1\ldots i_k}\|\psi\rangle$
`.add_partial_derivative(j)`	$p \leftarrow p + \frac{\partial}{\partial x_j}p$	$\|\psi\rangle \leftarrow X_j\|\psi\rangle$
`.add_polynomial(q)`	$p \leftarrow p + q$	$\|\psi\rangle \leftarrow V_q\|\psi\rangle$
`.replace_var(i, j)`	$p \leftarrow p + x_i\frac{\partial}{\partial x_j}p$	$\|\psi\rangle \leftarrow \mathrm{CNOT}_{ij}\|\psi\rangle$
`.replace_var_multi(`$i_1,\ldots,i_k$`, j)`	$p \leftarrow p + x_{i_1}\cdots x_{i_k}\frac{\partial}{\partial x_j}p$	$\|\psi\rangle \leftarrow C^k\mathrm{NOT}_{i_1\ldots i_k j}\|\psi\rangle$
`.substitute_random_value(i)`	$p \leftarrow p\|_{x_i=0}$ or $p\|_{x_i=1}$ u.a.r.	Measure Z_i on $\|\psi\rangle$
`.equals(q)`	Return $q == p$ with bounded error probability.	SWAP test on $\|\psi_p\rangle$ & $\|\psi_q\rangle$, or QSE on $\|\psi_p\rangle \otimes \|\psi_q\rangle$
`.read_linear()`	Return c s.t. $p(x_1,\ldots,x_n) = c\cdot x$, under promise that p has this form.	X measurement on all qubits

Table 5.4: The list of operations supported by the *multivariate polynomial over* $\mathbb{F}_2$ data structure that is implemented as a quantum hypergraph state.

and ought to be better known. We believe it can be included in more quantum computing courses because it is also a natural extension of the commonly taught Bernstein-Vazirani algorithm, and with our presentation its mathematics becomes quite simple, relying only on graphical circuit identities of hypergraph state preparation circuits. Ref. [114] describes the algorithm for multilinear polynomial learning over $\mathbb{F}_d$, not just $\mathbb{F}_2$. This more general algorithm can also be easily explained with data structures by making an extension to the Hypergraph state formalism and to the polynomial data structure presented above. This will be detailed in a forthcoming work but has not been included in this thesis.

The computational task in polynomial learning is to determine a polynomial $p : \mathbb{F}_2^n \to \mathbb{F}_2$ when given only oracular access to U_p, a unitary which implements the map $|x_1\ldots x_n\rangle|z\rangle \mapsto |x_1\ldots x_n\rangle|z \oplus p(x_1,\ldots,x_n)\rangle$. In the Bernstein-Vazirani problem we are promised that p has degree 1, and in the general polynomial learning problem we are promissed that the function has some degree at most d. The objective is to use as few queries of U_p as possible to determine the polynomial.

First note that using a single query to U_p, we may implement V_p which implements the map $|x_1 \dots x_n\rangle \mapsto (-1)^{p(x_1,\dots,x_n)}|x_1 \dots x_n\rangle$, as shown in Figure 2.4. We will use the multivariate polynomial data structure over $\mathbb{F}_2$ (abbreviated here as MPoF_2) defined in the last section for this explanation.

The oracle access can be viewed in the data structure as the ability to call the `add_polynomial(p)` function without knowing p because it has the same effect on the state. The objective is to learn p in as few calls to this operation as possible.

Algorithm 4: Oracle for the (unknown) polynomial p

procedure `V_p`(r : MPoF_2) $\to$ MPoF_2**:**

    r.`add_polynomial`(p)

    return r

The Bernstein-Vazirani algorithm can be stated very simply in terms of the polynomial data structure and the `read_linear` operation. First a new instance of MPoF_2 is created, which is initialised as the zero polynomial. Then the oracle call adds the linear polynomial to be learnt onto the MPoF_2 instance. Finally the read linear operation reads out a bitstring which one-hot encodes which monomials are present in the sum. Here we define a `learn_linear` function which will be used in defining the polynomial learning algorithm. Note that the classical algorithm for learning an n variable linear function with no constant term requires n queries to the classical oracle, but the quantum algorithm only requires a single query .

Algorithm 5: Learning a degree-1 polynomial

procedure `learn_linear`(f : oracle) $\to$ bitstring**:**

    r : $\mathrm{MPoF}_2 \leftarrow$ `create`(n)

    $f(r)$

    return r.`read_linear`()

The polynomial learning algorithm can be defined recursively with `learn_linear` as the base case of the recursion. The recursive step relies on the `add_partial_derivative` operation to compute n partial derivatives of the function with respect to each variable. The partial derivatives will all have degree one less than the original polynomial, thus reducing the problem to a simpler case.

The three lines in Algorithm 6 achieve the transformation on the zero polynomial: $0 \mapsto f(x) \mapsto f(x) + \frac{\partial}{\partial x_i} f(x) \mapsto \frac{\partial}{\partial x_i} f(x)$.

Algorithm 6: Function to prepare the i-th partial derivative of f

```
procedure partial_derivative(i : int, f : oracle, r : MPoF_2) →
 MPoF_2:
    f(r)
    r.add_partial_derivative(i)
    f(r)
    return r
```

To illustrate the recursion, we show both the algorithm for learning degree 2 polynomials, and the general algorithm which works in a similar way.

Algorithm 7: Learning a degree-2 polynomial

```
procedure learn_quadratic(f : oracle) → bitstring:
    res : bitstring ← ε
    for i ← 1 to n do
        res ← res|| learn_linear(λr. add_partial(i, f, r))
    return res
```

Algorithm 8: Learning a degree-d polynomial

```
procedure learn_polynomial(d : int, f : oracle) → bitstring:
    if d = 1 then
        return learn_linear(f)
    else
        res : bitstring ← ε
        for i ← 1 to n do
            res ← res|| learn_polynomial(d − 1, λr. add_partial(i, f,
              r))
        return res
```

5.4 Discussion

Stabilizer formalisms like the graph and hypergraph formalisms are a good framework for designing quantum data structures for a few reasons:

- They focus our attention on a discrete and finite subset of the n-qubit inner product space, into which we can encode the information. This fits the philosophy of data structures, which are structured and particular ways of interacting with information registers.
- The set of states often have some upper bounds on their pairwise inner products, which allows them to be distinguished with bounded error probability.
- The number of states on n qubits is generally more than 2^n, allowing for a space advantage over classical encoding
- The data usually already corresponds to some data type of interest, such as in the above cases to graphs and hypergraphs.
- Stabilizer formalisms can make it easy to determine which operations are closed on the set of states, that is, which operations map stabiliser states to other stabiliser states and do not leave the set. This is helpful in the context of data structures to maintain guarantees on how the state will behave under future modification and read out operations. This is not too dissimilar from the use of stabiliser formalisms in quantum error correction, and the design of logical gates on QEC codes.

Two interesting formalisms, the XS [118] and XP stabilizer formalisms [109], can be explored to make a weighted or coloured hypergraph data structure, which generalises hypergraphs to having different types of edges. Any other interesting classes of quantum states which have some of the above properties can also be considered for the purpose of designing quantum data structures.

Chapter 6

Algorithms using quantum data structures

This chapter considers how the quantum data structures discussed in this thesis can be used to design quantum algorithms. We will see that some quantum data structures can directly be inserted into classical algorithms because they implement the same data type, and discuss the possibility of hybrid data structures leveraging both classical and quantum data structures for their respective strengths. We will then provide a general argument to prove the space advantage of quantum data structures in the Simultaneous Message Passing (SMP) model. Lastly, we will discuss quantum advantages over known classical algorithms on memory checking problems and why we expect these quantum advantages to hold.

6.1 Replacing the data structure in standard algorithms

As described in Chapter 1, an abstract data type is oblivious to the details of the data structures that meet its specification, and thus a quantum data structure can be used to meet the specifications of abstract data types that were conceived without quantum computing in mind. In this section, we show some standard classical algorithms in which the classical data structure can be replaced by a quantum equivalent. What is notable is that the quantum data structures we have seen so far come with a major caveat: to sidestep no-go theorems and achieve a space advantage, they

necessarily cannot support a universal set of operations of the analogous classical data structures. Therefore, in order to use these quantum data structures in standard classical algorithms, we must find algorithms that only use those operations that the quantum data structure is also able to support.

Note that the quantum data structures we have explored have some failure probability in their equality checking operations. As a result, using one of these quantum data structures may improve the space complexity of these algorithms over standard classical data structures, but at the cost of turning a deterministic algorithm to one with a one-sided error. However, the probability of error can be suppressed greatly without much overhead.

Moreover, in the cases we consider here, the quantum advantage is not over all possible classical algorithms, as under the one-sided error situation there are better classical algorithms relying on hashing techniques. Nonetheless, it is encouraging that even some very standard algorithms immediately connect with quantum data structures, and as the set of known quantum data structures grows, it seems likely that we will find algorithms where they can be used to achieve theoretical and practical advantages over classical methods.

6.1.1 String Processing

String processing algorithms are used ubiquitously in text processing, textual data analysis, databases, DNA analysis. Also, it arises frequently in real world applications that data arrives as a stream and needs to be processed in an online fashion under memory constraints, such as in networking devices, data produced from experimental setups (e.g. particle accelerators) or sensors (e.g. in space observation), and in the processing of digital signals. Here we consider how quantum fingerprints can be used as the data structure to encode the symbols in canonical string processing algorithms, especially where the string arrives in a stream.

The streaming setting is chosen because it is more practical in the near-term, as we already can move classical data onto a quantum computer piece by piece by the simple application of quantum logic gates chosen in response to the data. The less practical setting that we do not consider is of string processing algorithms that

access the string all at once in superposition by using a device such as a Quantum Random Access Memory.

There is an interesting reason for the choice of these string processing algorithms in this discussion. We will focus on the bracket matching algorithm, but other algorithms such as that for checking if one string is a prefix of another, also have this interesting structure. In these algorithms, specifically bracket matching and the algorithm to check if one string is the prefix of another, the classical data structure can be swapped out for the quantum data structure without any other changes to the algorithm. This is because these algorithms use the data types in a way that is naturally well suited to the quantum data structure. Equality checks via the SWAP test or QSE on quantum states destroy the state if the states are not equal. Therefore, quantum data structures shine in those algorithms where few equality comparisons are made, or at least only few equality comparisons return the result "NOT EQUAL". Note that not every algorithm is like this, and in fact most classical algorithms take equality comparisons for granted and use them ubiquitously, for example in the naive algorithm for searching for a string in a document [18]. So it is interesting that some algorithms do have this property that they have very few equality comparisons and thus quantum data structures can be used in them without modification of the algorithmic approach, and only by changing the data structure used.

We now demonstrate the use of quantum fingerprinting to get a space advantage in the problem of checking if a streamed input string is well bracketed. This is the same as checking if a string belongs to the language Dyck_k, defined as follows. Problems and algorithms relating to Dyck language recognition (i.e. the problem of validating that an input string is well bracketed) have been studied in quantum computation, both because it is a simple and fundamental computational problem, and because there are interesting connections to quantum computing. In the query algorithm setting, the query complexity of Dyck languages with bounded height has been studied for both single type of bracket [119, 120] and multiple types of brackets [121]. LaTeX environments and HTML tags are practical examples of strings which contain different kinds of brackets and need to be well-bracketed, but this

need arises in virtually all programming languages. Note that our algorithm is only asymptotically more efficient, but is not practical for programming language syntax analysis because of the large resource overheads of quantum computation.

In the streaming setting, there is an efficient reduction from Dyck_s to Dyck_2, and a space-efficient classical algorithm has been developed using hashing techniques [122, 123]. These techniques outperform our quantum algorithm.

Definition 1 (Dyck language with k kinds of brackets) *Let*

$$\Sigma_k \;=\; \big\{\, [_1,]_1, [_2,]_2, \ldots, [_k,]_k \,\big\}$$

*be an alphabet of k distinct pairs of matching parentheses. The **Dyck language** on Σ_k, denoted $\mathrm{Dyck}_k \subseteq \Sigma_k^*$, is the smallest set satisfying*

1. $\varepsilon \in \mathrm{Dyck}_k$ *(the empty word),*
2. *if* $u \in \mathrm{Dyck}_k$ *then* $[_i\, u\,]_i \in \mathrm{Dyck}_k$,
3. *if* $u, v \in \mathrm{Dyck}_k$, *then* $uv \in \mathrm{Dyck}_k$.

Equivalently, Dyck_k *is generated by the context-free grammar*

$$S \;\to\; \varepsilon \;\mid\; SS \;\mid\; [_1\, S\,]_1 \;\mid\; [_2\, S\,]_2 \;\mid\; \ldots \;\mid\; [_k\, S\,]_k .$$

The computational task of checking whether brackets match well in a string provided in a stream is then defined as follows.

Problem 4 (STREAMING-DYCK$_k$-MEMBERSHIP$_\varepsilon$) *A word $w = w_1 w_2 \ldots w_n \in \Sigma_k^*$ presented one symbol at a time, left to right.* ***Goal:*** *Decide whether $w \in \mathrm{Dyck}_k$ with one-sided error at most ε; that is, for every $w \in \Sigma_k^*$,*

$$\Pr\big[\mathcal{A}(w) = \textsc{accept}\big] \;\begin{cases} = 1, & \text{if } w \in \mathrm{Dyck}_k, \\ \leq \varepsilon, & \text{if } w \notin \mathrm{Dyck}_k . \end{cases}$$

The standard algorithm for this problem can be stated in terms of a Stack abstract data type (see Table 1.4). We will compare the space complexity of this algorithm in two cases, under a classical and quantum data structure to implement the symbols inside the Stack Abstract Data Type.

Algorithm 9: STREAMINGDYCK(k) with an explicit stack

Input: Stream $w = w_1 w_2 \dots w_n$ over Σ_k

Output: ACCEPT or REJECT

$S : Stack \leftarrow$ `create()`

for $i \leftarrow 1$ **to** n **do**

  if w_i *is an opening bracket* $[_j$ **then** S.`push(`j`)`

  else if w_i *is a closing bracket* $]_j$ **then**

    if S.`size() == 0` **then**

      return REJECT

    $t \leftarrow S$.`pop()`

    if $t \neq j$ **then**

      return REJECT

return $(S = \varnothing)$? ACCEPT : REJECT

The quantum algorithm replaces the symbol data structure with quantum fingerprints. So instead of pushing a bitstring that identifies bracket type j onto the stack, a stack of quantum states is maintained instead, and the quantum fingerprint of the bitstring for j is pushed onto the stack. In an input string of size n, there are at most $\mathcal{O}(n)$ types of brackets. Thus $\mathcal{O}(\log n)$ bits are needed to store the classical bitstring for each bracket type, since each bracket type must have a unique encoding. Thus the classical algorithm uses $\mathcal{O}(n \log n)$ space. On the other hand, if we store quantum fingerprints for each bitstring instead, the $\mathcal{O}(\log n)$ bit strings can be stored in $\mathcal{O}(\log\log n + \log \varepsilon^{-1})$ space instead, for an ε error probability in distinguishing the fingerprints. Therefore the quantum algorithm uses $\mathcal{O}(n(\log\log n + \log \varepsilon^{-1}))$ space which is asymptotically better in n than the $\mathcal{O}(n \log n)$ scaling for the naive classical algorithm. However, note that this advantage is unlikely to be practical,

and moreover better designed classical algorithms achieving one sided error can use much less space [122] to solve this problem. As explained above, this problem is merely to illustrate that quantum data structures can sometimes straightforwardly replace the classical data structure in existing classical algorithms and achieve an advantage.

6.2 Message passing scenarios

As discussed in subsection 3.2.2, the Simultaneous Message Passing (SMP) model is an important computational setting in which we can define computational tasks that capture the difference in power between quantum and classical data structures. Quantum fingerprinting achieves an exponential communication complexity advantage in the SMP model when Alice and Bob do not share randomness, and the goal is to compute the equality function $f_{eq}(x,y) = [x == y]$.

One advance of quantum data structures in this thesis over quantum fingerprinting is that they have operations that modify the stored data. The data structures will thus perform well in a communication game that makes use of these additional modification operations. Consider the following scenario: as usual, Alice and Bob start with some data x and y respectively, but in addition, the Referee starts with a sequence of reversible data structure operations, g. The objective of the three parties is to compute the function $f_{eq(g)}(x,y) = [g(x) == y]$, which outputs a bit indicating whether $g(x)$ is equal to y. Naturally, the quantum data structures allows Alice and Bob to send the quantum state data structure encodings of x and y, after which the Referee can apply the operation g to the encoding of x and perform a SWAP test.

A variant of the SMP model is pictured in Figure 6.1 where A_1 first sends a message to A_2, and then A_2, B and R participate in a Simultaneous Message Passing scenario. We can also consider a generalisation to n parties $A_1, ..., A_n$ (see Figure 6.2). The use of different parties in the message passing scenario does not necessarily correspond to different computers or users, but could be the same user across different times. Consider the computational task where each party $A_2, \ldots, A_n$ and R is given

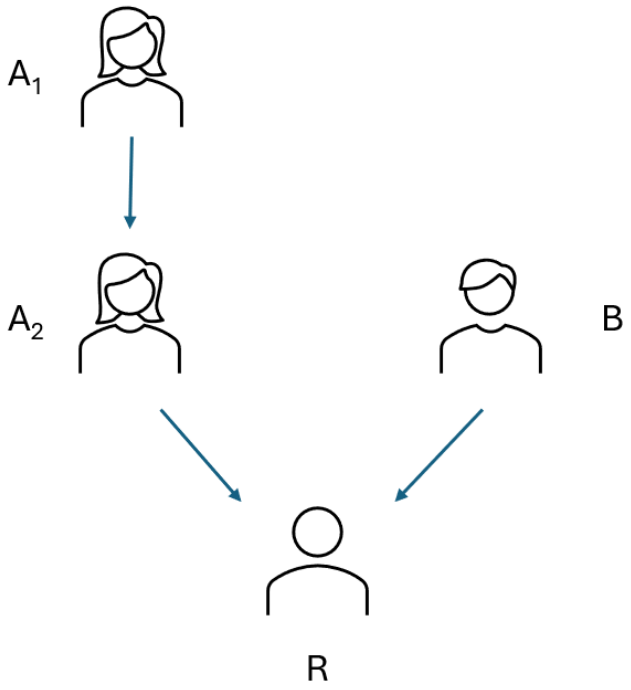


Figure 6.1: A modified Simultaneous Message Passing scenario where one branch has two contributors, A_1 first sends a message to A_2 and then the Referee receives a message from A_2 and Bob and has to compute something from both messages.

a data structure operation $g_2, \ldots, g_n, g_{n+1}$ respectively, and the objective of all the parties is that R should output $[g_{n+1} \circ \ldots \circ g_2(x) == y]$.

These scenarios are at least as hard as equality checking in the SMP model. In the first setting of computing $f_{eq(g)}$, since the identity operation is always a data structure operation and thus a possible choice for g, the communication complexity of computing f_{eq} is at most as much as that of computing $f_{eq(g)}$. In the latter settings of multiple A_i's, if A_n has additional perfect knowledge of the data held by all previous A_i, then this problem reduces to equality checking in SMP. Therefore the classical lower bounds for both these scenario are at least as much as that for equality checking in the SMP model, which is $\Omega(\sqrt{n})$ bits of communication for a bounded error probability [8, 61].

This means that as long as one of the aforementioned computational tasks calls for a quantum data structure that uses asymptotically fewer than $\sqrt{n}$ qubits, where n is the number of bits needed to give a unique encoding to each data value, there is a

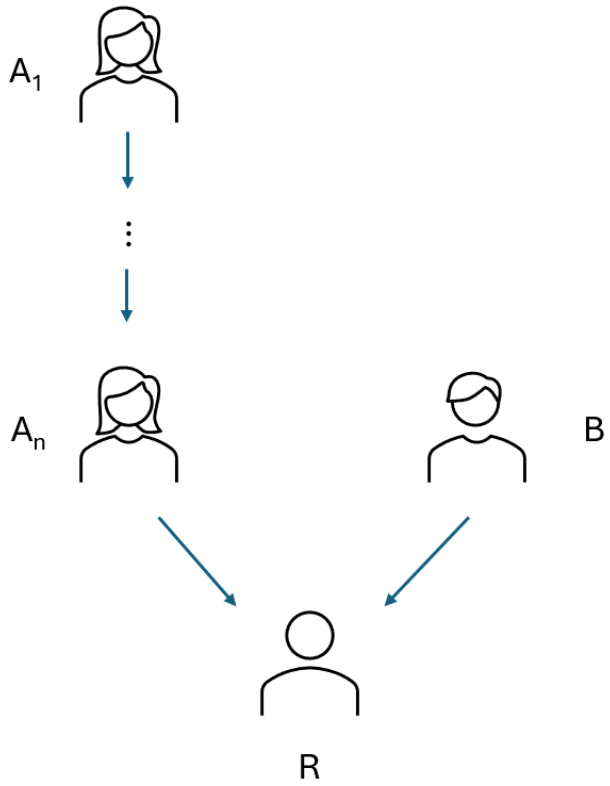


Figure 6.2: A modified Simultaneous Message Passing scenario where one branch has n contributors, where Referee receives a message from A_n and Bob and has to compute something from both messages. The various parties are only allowed to send messages according to the arrows starting at the top and proceeding downwards.

quantum space advantage on that task. For example, on the problem of determining whether $a_1 \oplus a_2 = b$, where a_1, a_2 and b refer to the bitstrings possessed by A_1, A_2 and B respectively, in a setting as in Figure 6.1, our quantum data structure for bitstrings with XOR (subsec. 4.3.1) achieves an exponential space advantage since it uses $\mathcal{O}(\log n)$ qubits to compute this property on n bit strings. In addition, all the examples of data structures given in Chapter 4 and the hypergraph data structure (Sec. 5.2) achieve asymptotic quantum space advantage in these games. However, this argument is not sufficient for proving the same advantage for the bipartite graph state list data structure, or the graph state data structure as they achieve only a quadratic advantage.

These theoretical communication scenarios also model how a data structure may be passed around between different parties or computational processes, each of which may make changes to the data. Ref. [62] describes a number of practical distributed computation scenarios in which such a need may arise. In addition, this setting is theoretically important to us as it helps highlight the difference in power between classical and quantum data structures.

6.3 Memory checking

In the computational task of memory checking, which is defined for a particular data type [124, 123], the input is a stream of operations in that data type, and the goal is to output in the manner that a correct implementation of the data type would output. Such a problem can be defined for any ADT and is canonical for that data structure because it captures its full power. Any other use of the data structure can be seen as a particular case of memory checking.

Our data structures achieve a space advantage over deterministic classical data structures because they use fewer qubits than the number of bits required to encode the data. Note that they do this at the cost of introducing a one-sided error. Thus the memory checking problems defined relative to them will only require outputting correctly with the same one-sided error probability. It is therefore possible that a randomised classical data structure may exist that can get the same space complexity on these memory checking problems. For example, as we saw in quantum fingerprinting, it is possible to sample some random bits from the codeword and use those for equality comparison. However, memory checking algorithms that use automorphisms (as introduced in Chapter 4) break this possibility as the modifications on the function state may leave the different data structures incomparable.

Consider a computational task where two data values u, and v are given, and afterwards two sequences of data structure operations f and g are given in a stream. The objective is to output whether $f(u) = g(v)$. Now consider how the classical random sampling strategy will fail on this computational task. Let E and F denote the error correcting codewords that encode u and v. That is, $|[u]\rangle = \sum_i |i\rangle|E_i\rangle$ and

$|[v]\rangle = \sum_i |i\rangle|F_i\rangle$. The classical randomised strategy would seek to dequantise this by storing the randomly sampled bits of these codewords $\{(i_1, E_{i_1}), ...(i_c, E_{i_c})\}$ and $\{(i_1, F_{i_1}), ...(i_c, F_{i_c})\}$. However, after the simulation of the data structure operations, the two sets may not contain any bits that are of the same coordinate, or only a few, and they may all be equal even if the two codewords are actually different.

We illustrate this with an example of graph states. Let

$$G_1 = (\{1,2,3\}, \{\{1,2\}, \{2,3\}\}),$$
$$G_2 = (\{1,2,3\}, \{\{2\}, \{1,2\}, \{2,3\}\}).$$

Suppose that we randomly sample the $011, 101, 100$ coordinates of the codewords. Thus, G_1 is encoded as $\{(011,1), (101,0), (100,0)\}$ and G_2 is encoded as $\{(011,0), (101,0), (100,0)\}$. If we apply `nlt(1)` on G_1, the correct result is G_2. On our random coordinate encoding, the stored value will become $\{(111,1), (001,0), (000,0)\}$. This has no indices in common with the encoding of G_2 and so cannot be compared. The algorithm should output that they are equal, but it has no basis from which to do this, as the encoding is incomparable to both graphs to which it is supposed to be equal and those it is not.

Therefore the classical random sampling strategy cannot dequantise all data structures that use coordinate permutations. There may be better classical data structures but it is somewhat tricky to construct them if they exist, as these will have to match the space and time complexities of quantum data structures on all the operations they support. Future work should examine this idea to see if the classical lower bound can be rigorously shown and quantum advantage proven.

6.4 Discussion

The examples in this chapter illustrate that quantum data structures can yield space advantages in specific algorithmic settings, though the practical scenarios where these advantages materialise remain limited at present. The bracket matching example shows that some classical algorithms are naturally compatible with quantum data structures, requiring only a swap of the underlying representation. We anticipate

that as the catalogue of quantum data structures expands, more such algorithms will be discovered, particularly in streaming settings, distributed verification, and problems like memory checking where the combination of modification operations and equality testing creates a computational task that resists classical randomised strategies. Future work will be needed to prove classical lower bounds and make these advantages rigorous. Another key challenge for future work is to move beyond asymptotic advantages and identify concrete applications where the space savings justify the engineering overhead of quantum computation.

Chapter 7

Discussion

7.1 Further questions

Further research should continue to design new quantum data structures for existing and new abstract data types. These data structures should then be applied to practical problems and existing classical algorithms, but also new computational tasks should be defined to better capture where these data structures are likely to shine. Some research should also explore whatever can be said in general about quantum data structure constructions, such as lower bounds on complexity and no-go theorems. Here we detail some other specific research directions that should be pursued.

7.1.1 Readout operations beyond equality testing

The main readout operation that we have focussed on in this thesis is equality testing. This was because to introduce equality testing in a quantum data structure just required an encoding with bounded inner products. Across this thesis we saw many ways to design encodings around inner products, namely: using the code distance of classical error correcting codes as in quantum fingerprinting and our extensions to it (sec. 3.2 and chap. 4), using stabiliser, graph and hypergraph states (sec. 3.3 and chap. 5), destructive interference of phases (sec. 3.4) and tuning the encoding to a promise in the problem (sec. 2.4). There may be still other techniques to construct quantum encodings with small pairwise inner products.

However, it would be interesting to consider other kinds of readout operation. For example, sampling is a natural readout operation for quantum states. The Pair

Sketch used in Ref. [125] gives one compelling example. One could consider a variant of Random Access Codes where the objective is not to store n bits and retrieve one, but rather to encode n Bernoulli distribution parameters, and sample from one of them later. Data structures like this would exploit quantum contextuality, achieving an advantage by a principle similar to Bell inequality violations. Another strength of quantum encodings is the possibility of using phenomena like unambiguous quantum state discrimination to get data structures and computational tasks that are natural in the quantum case, but more difficult in the classical case.

Beyond equality and sampling, it is possible to consider data structures that can measure similarity between two data values. Graph states have this property, since the inner product of graph states corresponds well to a notion of similarity between graphs - specifically the number of vertices that have the same neighbourhood. However, other data structures that we have considered such as those based on quantum fingerprinting do not have this property. Some constructions of quantum counters have this property while others do not.

For sampling and similarity readout operations, it will be interesting to explore the design of data structures with such operations either axiomatically, or by studying interesting classes of states which may natively support such features.

7.1.2 Practicality of quantum data structures

Quantum data structure-based quantum algorithms have potential to yield quantum advantage in the near term as shown by some recent works on such data structures and the related topic of streaming algorithms [62, 126, 76]. In order to make more of these theoretical quantum data structure-based algorithms practical, complementary research and engineering work is needed.

Fault tolerance techniques need to be developed for quantum data structures. This could involve circuit level techniques tailored to the data structure, or else just finding the appropriate error correcting codes. It would be interesting if FTQC techniques are chosen so that the logical operations that are easier to implement in that quantum error correcting code or FTQC scheme are also the data structure operations that are used most often. For example, since graph state data structure

operations are all Clifford, they can be implemented fault tolerantly with low overheads in the surface code [127]. Techniques for fault-tolerance in quantum data structures will also need to consider the specific algorithmic needs of data structures, for example preserving the one-sided nature of quantum state equality testing, even in the presence of errors.

Once the FTQC scheme has been designed for the data structure, an investigation into the resource requirements and required performance metrics should be done. This will include choosing an architecture for the qubits, estimating the number of physical qubits and gates, estimating how fast the data structure would run operations in wall clock time, and estimating how much energy it would take to run. Then it would also help to know what sort of performance metrics a quantum data structure needs to hit for quantum advantage or practical utility. This will also set a goal for how much the quantum data structure and FTQC schemes need to be optimised or help decide which applications are ultimately impractical.

Another possibility is to co-design hardware architectures with quantum data structures in mind. This may lead to the prioritisation of certain operations, control protocols, interface speeds, etc, and it will help identify which qubit modalities are best for these algorithms.

Once quantum computing systems mature, it will be productive to formalise their computational model and define these data structures more precisely in terms of them. Another effort could involve coding up these data structures in a quantum programming language such as Q#, which has the high level constructs to support object oriented programming and dynamic memory allocation.

Finally more programming interfaces to quantum data structures should be developed which allow algorithm and software developers to write high level programs that use quantum data structures, even if they do not know much about quantum computing.

7.1.3 Hybrid data structures

Another possibility is to use a universal classical data structure together with a quantum data structure. In this approach, the quantum data structure can be used

to respond quickly to those queries that it can support, and if the quantum state is consumed in a read out operation, we can fall back to the classical data structure to respond to the query. While the computer is less busy, it can use the classical data structure to prepare another copy of the quantum data structure for future use.

This approach opens the design space greatly, as it means the role of quantum data structures is only to add more powers to the data structure. Moreover, since the quantum data structures use asymptotically less space than the classical data structures, from an asymptotic perspective, this also does not change the space complexity of the original classical data structure. To make this case compelling, we should find application areas where there are busy periods where data structure queries need to be responded to quickly, and quiet periods when the quantum data structures that were consumed by readout operations can be restored.

7.1.4 Arbitrary group data structures

The quantum data structures for Counter ADT are evocative of finite group representation theory. Since the increment unitary satisfies $D^n = I$, the group homomorphism $\rho : C_n = (\mathbb{Z}_n, +(\mathrm{mod}\, n)) \to \mathrm{SU}(d)$, with $\rho(1) = D$, is a complex linear representation of cyclic group. This motivates generalising this investigation to arbitrary groups.

Once again a key design choice for quantum state data structure will be the approximate orthogonality of the different values that the data structure can take. Let $|[g]\rangle$ denote the encoding of group element g. We want each pair of states to have a small inner product.

We also want to be able to operate on the data structure with elements of the group. For every $g \in G$, there exists a unitary U_g such that for every $h \in G$, $U_g|[h]\rangle = |[gh]\rangle$.

In summary we make the following assumptions:

- Each value of the data type ($d \in \mathcal{D}$) is assigned a quantum state ($|\psi_d\rangle \in X$). That is we have a bijective encoding function $|\psi_\square\rangle : \mathcal{D} \mapsto X$.
- There exists a constant ε such that for all $d_1 \neq d_2 \in \mathcal{D}$, $\left\langle \psi_{d_1} \middle| \psi_{d_2} \right\rangle \leq \varepsilon$

- Our data structure supports reversible operations ($\{U_k\}_k$) and equality comparisons implemented as a SWAP test, or by QSE (see 2.2).

The key question is, given a group G and ε, the maximum inner product between states, find a faithful representation of this group.

Notice how each reversible operation U_k can be seen as a group action on X, at the logical level the data structure operation can be seen as a group action on $\mathcal{D}$. Ref. [128] investigates this question for finite Abelian groups. They use a similar approach as in [67] of randomly sampling representations and putting them in a diagonal matrix to produce the matrices that represent the group elements. This also uses the fact that finite Abelian groups have roots of unity as their one-dimensional representations, just like quantum counters. It will be interesting to explore if similar randomised approaches will work for finite groups that do not have one-dimensional representations.

It is also possible to consider a data structure where the operations are the representation of a group but the data values do not necessarily correspond to group elements. In this setting, $\mathcal{D} = \{d_i\}_i$ would be a set of data values, stored as quantum states $\{|[d_i]\rangle\} = X$ and $\{O_i\}_i$ would be a set of reversible operations acting on the set $\mathcal{D}$, which would be implemented as unitary operations on X.

7.1.5 Exploring generative AI as a tool for data structure and algorithm design

The research questions in this PhD thesis were conceived before the LLM boom. The hope was that quantum data structures would help human scientists and engineers design quantum algorithms. It has now become commonplace to do research with AI-assistance. As a result, a new standard against which to judge the results in this work is to assess whether they help AI models to design new quantum algorithms. As quantum algorithm design has been a difficult challenge, and one hope is that AI tools will help make it easier [13, 129].

It is a well-established maxim of research that breakthroughs often come when an idea from a different research community is brought into a field. Often patterns of

thinking within a field can become stale, and fresh new techniques and perspectives can shed new light on the topic. Such "serendipity" will be familiar to anyone who has done research across more than one topic, or changed departments during their career.

One strength of using AI agents for scientific research (in September 2026) is that they are able to identify such productive connections between research topics automatically. Consider the following model of the situation. Take a set of all scientific ideas $\mathcal{I} = \{I_1, I_2, ...\}$, defined in one of many possible ways such as techniques, tricks, papers, algorithms, data structures or Wikipedia pages. Next suppose that a process, $\mathcal{P}$, of considering the connections between two scientific ideas exists so that if this process is applied to two ideas, there is a small probability that it will yield a new research result. At present, we apply $\mathcal{P}$ to pairs of ideas in a clique-like fashion. All members of a research community follow broadly the same set of research journals and conferences and are mostly familiar with all the best ideas within their field, but a small minority of the best ideas outside their fields. Moreover, to properly perform $\mathcal{P}(\mathcal{I}_m, \mathcal{I}_n)$, it may require a human researcher to study two academic papers over at least a few weeks.

However, we are now able to automate the process $\mathcal{P}$ with LLM agents, and by applying this on some corpus of research papers, generating various new candidate ideas, and asking AI agents to score the best ones, we could enable discoveries that require combining those ideas for which there is no human researcher who has had the opportunity or interest to study both. An example of this was with the recent AI-generated counterexample to the Erdős unit distance conjecture [130].

The perspective of quantum data structures is helpful here because we can reduce the combinatorial explosion of considering pairs of all possible research ideas, and we can instead design a process to quickly assess if a particular data structure will be applicable to a particular algorithm, and then carefully check the most promising candidates. The final results will be verifiable, as it will be a matter of checking whether the program for the algorithm only uses the operations available

in the data structure, and the correctness of the data structure, algorithms and the argument for the complexity of the algorithm.

Another direction of investigation is to apply frontier LLM agents to the new mathematical problems posed by quantum data structures, such as the design of error correcting codes over large alphabets with many automorphisms. Frameworks are thus more useful research outputs today because of the availability of intelligent agents to fill in the frameworks automatically.

7.2 Conclusion

This thesis has advanced the research in quantum data structures on many fronts. We have motivated quantum data structures as a field of study that can help design new quantum algorithms, which is a pressing need in quantum computing. In particular, we drew attention to how non-universal data structures are capable of quantum space advantages and are worth exploring despite no-go theorems that may initially seem to rule out the possibility of such quantum advantages.

We have shown how data structure concepts help to better articulate known quantum algorithms such as Deutsch-Jozsa (sec. 2.4), Grover (sec. 2.5), Bernstein-Vazirani algorithm and its generalisations, and Quantum Phase Estimation (sec. 2.6.2). Such explanations are conceptual and so require less algebraic derivation. As a result, they are easier to remember and apply to novel situations. In fact, even the postulates of quantum information can be stated as the claim that nature admits the construction of a particular data structure (sec. 2.1).

We presented an overview of existing literature on the theme of quantum data structures, most notably covering quantum fingerprinting [8] (sec. 3.2) and graph states as a data structure [64] (sec. 3.3).

We then extended these data structures in a number of ways in Chapter 4. Quantum fingerprinting was extended by adding modification operations, so that the stored data could be altered by unitary operations while staying encoded as a quantum fingerprint. This exploration led to a framework for data structures like quantum fingerprints, each of which can be designed by choosing error correcting

codes and using their code automorphisms as data structure operations. We examined the requirements on the codes that lead to good data structures and presented some data structures in this framework. We also used graph states to define a list-like data structure, and showed how hypergraph states can be used to define a data structure for hypergraphs, and a data structure for multivariable polynomials.

Finally we discussed the computational settings in which all these quantum data structures are more efficient than their classical analogues. In the years to come quantum computers will have small numbers of logical qubits, and so we suggest that the perspective taken in this thesis is timely and should be explored further as memory advantages on streaming algorithms are a viable approach for useful quantum computations in the early fault-tolerant quantum computer era. We also stand at the brink of an explosion in AI-powered research. We hope that the questions and provocations in this thesis will be the seed of not only productive quantum algorithms, but also those which will remain accessible and beautiful to humans.

Appendix A

Construction of Finite fields for Quantum Fingerprinting

A *finite field* is a finite set of elements (which one should think of being like numbers) together with addition and multiplication operations that satisfy the familiar arithmetic properties: commutativity, associativity, distributivity, and the existence of additive and multiplicative inverses (except for zero, which has no multiplicative inverse). In simpler terms, it is a number system with finitely many elements where addition, subtraction, multiplication, and division behaves similarly to how it does with real numbers. Finite fields are very useful in coding theory, because they allow us to apply useful properties of polynomials in the design of codes, without running into issues like the limited precision with which real numbers are stored in computers. For there to exist a finite field of order q, it must be that for some prime p and positive integer k, $q = p^k$. Thus we may have to use a field larger than our desired size.

The construction of finite fields of prime order (i.e. when $k = 1$) is a lot simpler than other finite fields ($k > 1$). In the case of prime q, all finite fields of order q are isomorphic to $(\mathbb{Z}_q, +, \times)$. Such fields are easy to construct and implement as the operations are just integer arithmetic modulo q, where we may choose an appropriately large prime q for our desired success probabilities and universe size. The construction of finite fields of order $q = p^k$ for $k > 1$ is more involved requiring a computationally expensive search for irreducible polynomials of high degree and implementation of arithmetic over polynomials rather than integers (for more details

see this blog post). We choose to avoid this complexity, opting to construct our finite fields as $(\mathbb{Z}_p, +, \times)$ for primes p. Note that this restriction does not greatly limit us as primes are abundant. There is always a prime number between n and $2n$ for any positive integer n, thus the size of our finite field will be at most two times larger than our desired size.

In order to avoid searching for primes when the data structure is initialised, a list of primes of various magnitudes can be maintained, and selected from based on the parameters of the data structure. Note that a particularly delightful set of primes are the Mersenne primes. Mersenne primes are primes that can be expressed as $2^C - 1$ for some natural number C. This is an elegant choice as it corresponds directly to integer arithmetic in a C-bit ones' complement system.

Consider the Mersenne prime $p = 2^3 - 1 = 7$. The field $\mathbb{Z}_7 = \{0,1,2,3,4,5,6\}$ can be implemented using 3-bit ones' complement arithmetic. For example: $5 + 2 = 7 \equiv 0 \pmod 7$, which is done in ones' complement as: $101_2 + 010_2 = 111_2$. In ones' complement, the bit pattern 111_2 represents -0 (which is equivalent to 0). If we later add 3 to this, $111_2 + 011_2 = 1010_2$. We only have 3 bits, so the overflow bit wraps around and gets added back (end-around carry): $010_2 + 1 = 011_2 = 3$. This allows standard and perfected arithmetic circuits to be used, rather than designing new ones for other primes or prime powers. For larger Mersenne primes such as $2^{31} - 1$ or $2^{61} - 1$, the same principle applies.

Bibliography

[1] Shashvat Shukla. A good first impression for quantum algorithms. In *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 3, pages 129–133, 2025.

[2] Qasim Bedford and Shashvat Shukla. Quantum Time Capsules. In *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 3, pages 87–95. IEEE, 2025.

[3] Shashvat Shukla, Dan E. Browne, and Shin Nishio. Anticipating decoder side-channel attacks in fault-tolerant quantum computers. *arXiv preprint arXiv:2607.12174*, 2026.

[4] Emily Grumbling and Mark Horowitz, editors. *Quantum Computing: Progress and Prospects*. National Academies Press, 2019.

[5] Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2001.

[6] Lov 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.

[7] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE, 1994.

[8] Harry Buhrman, Richard Cleve, John Watrous, and Ronald de Wolf. Quantum fingerprinting. *Physical Review Letters*, 87(16):167902, 2001.

[9] Charles H Bennett and Gilles Brassard. An update on quantum cryptography. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 475–480. Springer, 1984.

[10] Ashley Montanaro. Quantum algorithms: an overview. *npj Quantum Information*, 2(1):1–8, 2016.

[11] Ryan Babbush, Jarrod R McClean, Michael Newman, Craig Gidney, Sergio Boixo, and Hartmut Neven. Focus beyond quadratic speedups for error-corrected quantum advantage. *PRX Quantum*, 2(1):010103, 2021.

[12] Alexander M Dalzell, Sam McArdle, Mario Berta, Przemyslaw Bienias, Chi-Fang Chen, András Gilyén, Connor T Hann, Michael J Kastoryano, Emil T Khabiboulline, Aleksander Kubica, et al. Quantum algorithms: a survey of applications and end-to-end complexities. *arXiv preprint arXiv:2310.03011*, 2023.

[13] Ryan Babbush, Robbie King, Sergio Boixo, William Huggins, Tanuj Khattar, Guang Hao Low, Jarrod R. McClean, Thomas O'Brien, and Nicholas C. Rubin. The grand challenge of quantum applications. *arXiv preprint arXiv:2511.09124*, 2025.

[14] Peter W Shor. Introduction to quantum algorithms. In *Proceedings of Symposia in Applied Mathematics*, volume 58, pages 143–160, 2002.

[15] Carl B Boyer and Uta C Merzbach. *A history of mathematics*. John Wiley & Sons, 2011.

[16] Christopher Havens and Amy Shell-Gellasch. Fibonacci meets the pharaohs: The decomposition of multiplication. *Math Horizons*, 30(1):24–27, 2022.

[17] John E Hopcroft, Jeffrey D Ullman, and Alfred Vaino Aho. *Data structures and algorithms*, volume 175. Addison-Wesley Boston, MA, USA:, 1983.

[18] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2022.

[19] Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.

[20] Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014.

[21] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5):052328, 2004.

[22] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Architectures for a quantum random access memory. *Physical Review A*, 78(5):052310, 2008.

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

[24] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, 1992.

[25] Daniel R Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997.

[26] Tomasz Arodz. Generalized quantum Deutsch-Jozsa algorithm. In *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part VI 20*, pages 465–472. Springer, 2020.

[27] EPSRC Centre for Doctoral Training in Quantum Computation and Quantum Communications (QC2). Quantum doctoral programmes. `https://www.ucl.ac.uk/quantum/study-here/quantum-doctoral-programmes`, 2025. Accessed 28 March 2025.

[28] Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society of London. Series*

A: Mathematical, Physical and Engineering Sciences, 454(1969):339–354, 1998.

[29] David Deutsch. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[30] Ronald de Wolf. Quantum computing: lecture notes. *arXiv preprint arXiv:1907.09415*, 2019.

[31] Thomas G Wong. *Introduction to classical and quantum computing*. Rooted Grove Omaha, NE, USA, 2022.

[32] Peter W. Shor. Lecture notes for 8.370/18.435 quantum computation. `https://math.mit.edu/~{}shor/435-LN/`, 2022. Lecture notes, version accessed 7 July 2025.

[33] Lynn West. An introduction to various multiplication strategies. *Washington. USA. Merrill Prentice Hall*, 2011.

[34] Allen Klinger. The Vandermonde matrix. *The American Mathematical Monthly*, 74(5):571–574, 1967.

[35] Jon Louis Bentley, Dorothea Haken, and James B Saxe. A general method for solving divide-and-conquer recurrences. *ACM SIGACT News*, 12(3):36–44, 1980.

[36] Gloria Paradisi and Hugues Randriam. A presentation of the quantum Fourier Transform from a recursive viewpoint. *arXiv preprint quant-ph/0411069*, 2004.

[37] Lloyd N Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 2022.

[38] Alán Aspuru-Guzik, Anthony D Dutoi, Peter J Love, and Martin Head-Gordon. Simulated quantum computation of molecular energies. *Science*, 309(5741):1704–1707, 2005.

[39] Carl W Helstrom. Quantum detection and estimation theory. *Journal of Statistical Physics*, 1:231–252, 1969.

[40] Alexander Semenovich Holevo. Bounds for the quantity of information transmitted by a quantum communication channel. *Problemy Peredachi Informatsii*, 9(3):3–11, 1973.

[41] Alexander S Holevo. Statistical problems in quantum physics. In *Proceedings of the Second Japan-USSR Symposium on Probability Theory*, pages 104–119. Springer, 2006.

[42] Lev B Levitin. On quantum measure of information. In *Proceedings of the Fourth All-Union Conference on Information Transmission and Coding Theory, Tashkent*, 1969.

[43] Stephen Wiesner. Conjugate coding. *SIGACT News*, 15(1):78–88, January 1983.

[44] Andris Ambainis, Ashwin Nayak, Ammon Ta-Shma, and Umesh Vazirani. Dense quantum coding and a lower bound for 1-way quantum automata. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, STOC '99, pages 376–383, New York, NY, USA, 1999. Association for Computing Machinery.

[45] Ashwin Nayak. Optimal lower bounds for quantum automata and random access codes. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 369–376, 1999.

[46] Avraham Ben-Aroya, Oded Regev, and Ronald de Wolf. A hypercontractive inequality for matrix-valued functions with applications to quantum computing and LDCs. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, October 2008.

[47] João F. Doriguello and Ashley Montanaro. Quantum random access codes for Boolean functions. *Quantum*, 5:402, March 2021.

[48] Dmitry Gavinsky, Julia Kempe, Oded Regev, and Ronald de Wolf. Bounded-error quantum state identification and exponential separations in communication complexity. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, pages 594–603, 2006.

[49] Hartmut Klauck. On quantum and probabilistic communication: Las Vegas and one-way protocols. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 644–651, 2000.

[50] Harry Buhrman and Ronald de Wolf. Communication complexity lower bounds by polynomials. In *Proceedings 16th Annual IEEE Conference on Computational Complexity*, pages 120–130. IEEE, 2001.

[51] Iordanis Kerenidis and Ronald de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *Journal of Computer and System Sciences*, 69(3):395–420, 2004. Special Issue on STOC 2003.

[52] Iordanis Kerenidis. *Quantum encodings and applications to locally decodable codes and communication complexity*. PhD thesis, 2004.

[53] Stephanie Wehner and Ronald de Wolf. Improved lower bounds for locally decodable codes and private information retrieval. In *International Colloquium on Automata, Languages, and Programming*, pages 1424–1436. Springer, 2005.

[54] Masahito Hayashi, Kazuo Iwama, Harumichi Nishimura, Rudy Raymond, and Shigeru Yamashita. Quantum network coding. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 610–621. Springer, 2007.

[55] Masahito Hayashi, Kazuo Iwama, Harumichi Nishimura, Rudy Raymond, and Shigeru Yamashita. (4, 1)-quantum random access coding does not exist—one qubit is not enough to recover one of four bits. *New Journal of Physics*, 8(8):129, 2006.

[56] Scott Aaronson. The learnability of quantum states. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 463(2088):3089–3114, 2007.

[57] Marcin Pawłowski and Nicolas Brunner. Semi-device-independent security of one-way quantum key distribution. *Physical Review A*, 84:010302, Jul 2011.

[58] Hong-Wei Li, Zhen-Qiang Yin, Yu-Chun Wu, Xu-Bo Zou, Shuang Wang, Wei Chen, Guang-Can Guo, and Zheng-Fu Han. Semi-device-independent random-number expansion without entanglement. *Physical Review A*, 84:034301, Sep 2011.

[59] Ronald de Wolf. *Quantum computing and communication complexity*. PhD thesis, 2001.

[60] Peter Shor. The story of Shor's algorithm, straight from the source. YouTube, Jul 2021. Video lecture at the 40th Anniversary of the Physics of Computation Conference, MIT.

[61] Andrew Chi-Chih Yao. Some complexity questions, 1979.

[62] P. Gokhale, E. R. Anschuetz, C. Campbell, F. T. Chong, E. D. Dahl, P. Frederick, E. B. Jones, B. Hall, S. Issa, P. Goiporia, S. Lee, P. Noell, V. Omole, D. Owusu-Antwi, M. A. Perlin, R. Rines, M. Saffman, K. N. Smith, and T. Tomesh. SupercheQ: Quantum advantage for distributed databases. *arXiv preprint arXiv:2212.03850*, 2022.

[63] Radu Ionicioiu and Tim P. Spiller. Encoding graphs into quantum states: An axiomatic approach. *Physical Review A*, 85(6), June 2012.

[64] Liming Zhao, Carlos A. Pérez-Delgado, and Joseph F. Fitzsimons. Fast graph operations in quantum computation. *Physical Review A*, 93(3), March 2016.

[65] M. Hein, J. Eisert, and H. J. Briegel. Multiparty entanglement in graph states. *Physical Review A*, 69(6), June 2004.

[66] Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. Graphical description of the action of local Clifford transformations on graph states. *Physical Review A*, 69(2), February 2004.

[67] Andris Ambainis and Nikolajs Nahimovs. Improved constructions of quantum automata. *arXiv preprint arXiv:0805.1686*, 2008.

[68] John Harrison Watrous. *Space-bounded quantum computation*. The University of Wisconsin-Madison, 1998.

[69] John Watrous. On the complexity of simulating space-bounded quantum computations. *Computational Complexity*, 12:48–84, 2003.

[70] A. Borodin, S. Cook, and N. Pippenger. Parallel computation for well-endowed rings and space-bounded probabilistic machines. *Information and Control*, 58(1):113–136, 1983.

[71] François Le Gall. Exponential separation of quantum and classical online space complexity. *Theory of Computing Systems*, 45(2):188–202, December 2007.

[72] Andris Ambainis and Rusins Freivalds. 1-way quantum finite automata: strengths, weaknesses and generalizations. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pages 332–341. IEEE, 1998.

[73] Dmitry Gavinsky, Julia Kempe, Iordanis Kerenidis, Ran Raz, and Ronald de Wolf. Exponential separations for one-way quantum communication complexity, with applications to cryptography. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, pages 516–525, 2007.

[74] John Kallaugher, Ojas Parekh, and Nadezhda Voronova. Exponential quantum space advantage for approximating maximum directed cut in the streaming model. *arXiv preprint arXiv:2311.14123*, 2023.

[75] John Kallaugher and Ojas Parekh. The quantum and classical streaming complexity of quantum and classical Max-Cut. *arXiv preprint arXiv:2206.00213*, 2022.

[76] Haimeng Zhao, Alexander Zlokapa, Hartmut Neven, Ryan Babbush, John Preskill, Jarrod R McClean, and Hsin-Yuan Huang. Exponential quantum advantage in processing massive classical data. *arXiv preprint arXiv:2604.07639*, 2026.

[77] John Kallaugher. A quantum advantage for a natural streaming problem. *arXiv preprint arXiv:2106.04633*, 2021.

[78] Ashley Montanaro. The quantum complexity of approximating the frequency moments. *arXiv preprint arXiv:1505.00113*, 2015.

[79] Yassine Hamoudi and Frédéric Magniez. Quantum Chebyshev's inequality and applications. *arXiv preprint arXiv:1807.06456*, 2018.

[80] Ashwin Nayak and Dave Touchette. Augmented index and quantum streaming algorithms for Dyck(2). *arXiv preprint arXiv:1610.04937*, 2016.

[81] Amandeep Singh Bhatia and Ajay Kumar. Quantum finite automata: survey, status and research directions. *arXiv preprint arXiv:1901.07992*, 2019.

[82] Aida Gainutdinova and Abuzer Yakaryilmaz. Unary probabilistic and quantum automata on promise problems. *arXiv preprint arXiv:1502.01462*, 2015.

[83] Andris Ambainis and Abuzer Yakaryılmaz. Superiority of exact quantum automata for promise problems. *Information Processing Letters*, 112(7):289–291, 2012.

[84] Jibran Rashid and Abuzer Yakaryilmaz. Implications of quantum automata for contextuality. *arXiv preprint arXiv:1404.2761*, 2014.

[85] Jozef Gruska, Daowen Qiu, and Shenggen Zheng. Potential of quantum finite automata with exact acceptance. *International Journal of Foundations of Computer Science*, 26(03):381–398, 2015.

[86] Maria Paola Bianchi, Carlo Mereghetti, and Beatrice Palano. Complexity of promise problems on classical and quantum automata. In *Computing with New Resources: Essays Dedicated to Jozef Gruska on the Occasion of His 80th Birthday*, pages 161–175. Springer, 2014.

[87] Harry Buhrman, Richard Cleve, Serge Massar, and Ronald de Wolf. Nonlocality and communication complexity. *Reviews of Modern Physics*, 82:665–698, 2010.

[88] Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 103–111, 1995.

[89] Thomas G Draper. Addition on a quantum computer. *arXiv preprint quant-ph/0008033*, 2000.

[90] Lidia Ruiz-Perez and Juan Carlos Garcia-Escartin. Quantum arithmetic with the quantum Fourier transform. *Quantum Information Processing*, 16:1–14, 2017.

[91] Connor T Hann. *Practicality of quantum random access memory*. PhD thesis, Yale University, 2021.

[92] Samuel Jaques and Arthur G Rattew. QRAM: a survey and critique. *Quantum*, 9:1922, 2025.

[93] Iordanis Kerenidis and Anupam Prakash. Quantum recommendation systems. *arXiv preprint arXiv:1603.08675*, 2016.

[94] Ewin Tang. A quantum-inspired classical algorithm for recommendation systems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 217–228, 2019.

[95] Charles Yuan and Michael Carbin. Tower: data structures in quantum superposition. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA2):259–288, 2022.

[96] Miklos Santha. Quantum walk based search algorithms. In *International Conference on Theory and Applications of Models of Computation*, pages 31–46. Springer, 2008.

[97] Stacey Jeffery, Robin Kothari, and Frédéric Magniez. Nested quantum walks with quantum data structures. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1474–1485. SIAM, 2013.

[98] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977.

[99] Madhu Sudan. Probabilistically checkable proofs: A primer, 2006.

[100] Stephen P Jordan, Noah Shutty, Mary Wootters, Adam Zalcman, Alexander Schmidhuber, Robbie King, Sergei V Isakov, Tanuj Khattar, and Ryan Babbush. Optimization by decoded quantum interferometry. *arXiv preprint arXiv:2408.08292*, 2024.

[101] Richard W Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[102] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[103] D. E. Muller. Application of Boolean algebra to switching circuit design and to error detection. *Transactions of the I.R.E. Professional Group on Electronic Computers*, EC-3(3):6–12, 1954.

[104] W Cary Huffman and Vera Pless. *Fundamentals of error-correcting codes*. Cambridge University Press, 2010.

[105] Marvin Geiselhart, Ahmed Elkelesh, Moustafa Ebada, Sebastian Cammerer, and Stephan ten Brink. Automorphism ensemble decoding of Reed–Muller codes. *IEEE Transactions on Communications*, 69(10):6424–6438, 2021.

[106] Connor T Hann, Gideon Lee, SM Girvin, and Liang Jiang. Resilience of quantum random access memory to generic noise. *PRX Quantum*, 2(2):020311, 2021.

[107] Jørn Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18(5):652–656, 2003.

[108] Donald E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, MA, 3rd edition, 1997.

[109] Mark Webster. *The XP Stabilizer Formalism*. PhD thesis, University of Sydney, 2023.

[110] Davide Poderini, Dagmar Bruß, and Chiara Macchiavello. Quantum hypergraph states: a review. *Reports on Progress in Physics*, 89(6):066001, 2026.

[111] Matteo Rossi, Marcus Huber, Dagmar Bruß, and Chiara Macchiavello. Quantum hypergraph states. *New Journal of Physics*, 15(11):113022, 2013.

[112] Yue Gao, Shuyi Ji, Xiangmin Han, and Qionghai Dai. Hypergraph computation. *Engineering*, 40:188–201, 2024.

[113] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 11–20, 1993.

[114] Ashley Montanaro. The quantum query complexity of learning multilinear polynomials. *Information Processing Letters*, 112(11):438–442, 2012.

[115] Ryan O'Donnell. Analysis of Boolean functions. *arXiv preprint arXiv:2105.10386*, 2021.

[116] Claude Carlet. *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press, 2021.

[117] Ruben Gonzalez. Kyber – how does it work? Approachable Cryptography (blog), September 2021. Accessed: 2026-08-31.

[118] Xiaotong Ni, Oliver Buerschaper, and Maarten Van den Nest. A non-commuting stabilizer formalism. *Journal of Mathematical Physics*, 56(5), 2015.

[119] Kamil Khadiev and Yixin Shen. Quantum query complexity of Dyck languages with bounded height. *arXiv preprint arXiv:1912.02176*, 2020.

[120] Andris Ambainis, Kaspars Balodis, Jānis Iraids, Krišjānis Prūsis, and Juris Smotrovs. Quantum lower bounds for 2d-grid and Dyck language. *arXiv preprint arXiv:1911.12638*, 2019.

[121] Kamil Khadiev and Dmitry Kravchenko. Quantum algorithm for Dyck language with multiple types of brackets. *arXiv preprint arXiv:2106.09374*, 2021.

[122] F. Magniez, C. Mathieu, and A. Nayak. Recognizing well-parenthesized expressions in the streaming model. *arXiv preprint arXiv:0911.3291*, 2009.

[123] Amit Chakrabarti, Graham Cormode, Ranganath Kondapally, and Andrew McGregor. Information cost tradeoffs for augmented index and streaming language recognition. *SIAM Journal on Computing*, 42(1):61–83, 2013.

[124] Manuel Blum, Will Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. *Algorithmica*, 12(2):225–244, 1994.

[125] John Kallaugher, Ojas Parekh, and Nadezhda Voronova. How to design a quantum streaming algorithm without knowing anything about quantum computing. In *2025 Symposium on Simplicity in Algorithms (SOSA)*, pages 9–45. SIAM, 2025.

[126] Pradeep Niroula, Shouvanik Chakrabarti, Steven Kordonowy, Niraj Kumar, Sivaprasad Omanakuttan, Michael A Perlin, MS Allman, JP Campora III, Alex Chernoguzov, Samuel F Cooper, et al. Realization of a quantum streaming algorithm on long-lived trapped-ion qubits. *arXiv preprint arXiv:2511.03689*, 2025.

[127] Austin G Fowler and Craig Gidney. Low overhead quantum computation using lattice surgery. *arXiv preprint arXiv:1808.06709*, 2018.

[128] Alexander Vasiliev. Quantum hashing for finite Abelian groups. *Lobachevskii Journal of Mathematics*, 37(6):753–757, 2016.

[129] Dave Bacon. Augmenting our way to quantum advantage. YouTube, 2021. Invited talk at the Oxford University Quantum Information Society.

[130] Noga Alon, Thomas F. Bloom, W. T. Gowers, Daniel Litt, Will Sawin, Arul Shankar, Jacob Tsimerman, Victor Wang, and Melanie Matchett Wood. Remarks on the disproof of the unit distance conjecture. *arXiv preprint arXiv:2605.20695*, 2026.