Optimizing Quantum Compilers :

Efficient and Effective Algorithms

Thèse n. 9472 2023
présentée le 27 Avril 2023
à la Faculté des sciences et techniques de l'ingénieur
Laboratoire des Systèmes Intégrés – LSI
programme doctoral en Génie électrique
École polytechnique fédérale de Lausanne

pour l'obtention du grade de Docteur ès Sciences
par

Fereshte Mozafari

acceptée sur proposition du jury :

Prof. Andreas Peter Burg, président du jury
Prof. Giovanni De Micheli, directeur de thèse
Prof. Paolo Ienne, rapporteur
Prof. Robert Wille, rapporteur
Prof. Jason Cong, rapporteur

Lausanne, EPFL, 2023

To my beloved husband, parents, brother, and sisters.

Acknowledgements

First and foremost, I would like to express my heartfelt appreciation to Prof. Giovanni De Micheli, my supervisor, for his exceptional support, guidance, and encouragement during my Ph.D. journey. His mentorship and insightful feedback have been crucial in shaping my research and professional growth, particularly during the difficult times of the COVID-19 pandemic. His personality, passion and commitment to his students have been a constant source of inspiration, and I feel privileged to have had the opportunity to work under his supervision. I am truly grateful for his unwavering support and dedication.

I would like to thank my jury members Prof. Andreas Peter Burg, Prof. Paolo Ienne, Prof. Robert Wille, and Prof. Jason Cong for their time and their valuable feedbacks. I look forward to the opportunity to further discuss my research with them in the future.

I would like to thank Dr. Mathias Soeken for his invaluable guidance. His insights and expertise have been instrumental in shaping my research, and I am grateful for all the valuable ideas and discussions we have had together. I would like to extend my sincere thanks to Dr. Heinz Riener for his invaluable contributions to my research. His availability to answer my questions and his willingness to engage in valuable discussions have been very helpful in my work, and I am truly grateful for his guidance and support.

I would like to express my heartfelt appreciation to Dr. Yuxinag Yang, formerly a Postdoctoral Researcher at ETH and now a Professor at the University of Hong Kong, for all invaluable ideas and discussions. His knowledge, expertise, and unwavering dedication to our research have been instrumental in shaping my skills and expanding my knowledge of the field. I am deeply thankful for all collaborations and publications that we have had together.

I would like to express my gratitude to Dr. Harun Bayraktar, Dr. John Gunnel, Dr. Leo Fang, Dr. Yang Gao, and Dr. Andreas Hehn for the opportunity to work on exciting internship projects at NVIDIA. I am grateful for both research and software development collaborations, as well as the interactions I have had with my colleagues.

I would like to extend a special thanks to my current and former logic synthesis colleagues for engaging in fruitful discussions: Winston, Giulia, Kaitlin, Eleonora, Bruno, Ivan, Alessandro, Andrea, Dewmini, Sonia, Mingfei, and Rassul.

I want to express my sincere gratitude to Cristina, Carol, Chantal, and Valérie for their invaluable help in assisting me with various bureaucratic and non-bureaucratic issues.

I am grateful to my friends for their support, advice, and all fun times that we have had together during my academic journey.

Lastly, I thank my husband, my parents, and my family, to whom I dedicate this thesis, for unwavering support, love, and sacrifices throughout my academic journey. They inspire me to strive for greater goals in my life.

Lausanne, April 27, 2023 Fereshte Mozafari

Abstract

Quantum computing has made significant progress in recent years, with Google and IBM releasing quantum computers with 72 and 50 qubits, respectively. Google has also achieved *quantum supremacy* with its 54-qubit device, and IBM has announced the release of the Osprey quantum computer with 433 qubits. These developments suggest that quantum computers with even greater qubit counts may be available in the near future. This upcoming period is termed *Noisy Intermediate Scale Quantum* (NISQ) era, because of the noisy characteristics of near-term devices.

Computation on NISQ hardware is modeled using a library of supported quantum gates which can be directly implemented on it and a coupling graph that specifies available qubits interactions for multi-qubit quantum gates. While improvements to quantum hardware are continuously being made by experimentalists, quantum computing experts can contribute to the utility of quantum devices by developing software. This software would aim to adapt textbook quantum algorithms (e.g., for factoring or quantum simulation) to hardware constraints, that include: (1) limited number of qubits, (2) limited connectivity between qubits, (3) limited hardware-specific gate sets, and (4) limited circuit depth due to noise. Algorithms adapted to these constraints will likely look dramatically different from their textbook counterparts. Such software, which performs the task of translating quantum algorithms into quantum circuits according to hardware constraints, is called a quantum compiler. In conclusion, without a good compiler, most quantum algorithms are not applicable in practice.

Quantum compilers typically undertake three primary tasks: quantum state preparation, circuit synthesis, and qubit mapping. Quantum state preparation involves preparing the initial state of the quantum system before running the algorithm. The preparation of such a state itself requires a computation performed by a quantum circuit. This task can be challenging, as quantum algorithms assume some specific initial states in superposition before performing the desired application-specific computations. Circuit synthesis involves the process of constructing a quantum circuit that implements the desired quantum algorithm. Qubit mapping is another important task for quantum compilers, which involves mapping the logical qubits of the algorithm to the physical qubits of the quantum computer. In this thesis, I focus on the first two tasks of quantum compilers, quantum state preparation, and circuit synthesis. These tasks are essential for the successful execution of quantum algorithms, and developing efficient and effective algorithms for these tasks is crucial for the continued advancement of quantum computing.

Keywords: Logic Synthesis, Boolean Functions, Decision Diagrams, Quantum Computing, Quantum Compiler, Quantum State Preparation.

Zusammenfassung

Die Quanteninformatik hat in den letzten Jahren erhebliche Fortschritte gemacht. So haben Google und IBM Quantencomputer mit 72 bzw. 50 Qubits auf den Markt gebracht. Google hat mit seinem 54-Qubit-Gerät ebenfalls die Quantenüberlegenheit erreicht, und IBM hat die Veröffentlichung des Quantencomputers Osprey mit 433 Qubits angekündigt. Diese Entwicklungen deuten darauf hin, dass in naher Zukunft Quantencomputer mit einer noch größeren Anzahl von Qubits verfügbar sein könnten. Diese kommende Periode wird aufgrund der verrauschten Eigenschaften der in naher Zukunft verfügbaren Geräte als NISQ-Ära (Noisy Intermediate Scale Quantum) bezeichnet.

Die Berechnung auf NISQ-Hardware wird mit Hilfe einer Bibliothek von unterstützten Quantengattern, die direkt darauf implementiert werden können, und einem Kopplungsgraphen modelliert, der die verfügbaren Qubit-Wechselwirkungen für Multi-Qubit-Quantengatter angibt. Während Experimentalphysiker kontinuierlich Verbesserungen an der Quantenhardware vornehmen, können Quanten Experten für Quanteninformatik können durch die Entwicklung von Software zum Nutzen von Quantengeräten beitragen. Diese Software würde darauf abzielen, Lehrbuch-Quantenalgorithmen (z. B. für das Factoring oder die Quantensimulation) an die Hardwarebeschränkungen anzupassen, zu denen Folgendes gehört: (1) begrenzte Anzahl von Qubits, (2) begrenzte Konnektivität zwischen Qubits, (3) begrenzte hardwarespezifische Gattersätze und (4) begrenzte Schaltungstiefe aufgrund von Rauschen. Algorithmen, die an diese Beschränkungen angepasst sind, werden sich wahrscheinlich drastisch von ihren Gegenstücken aus dem Lehrbuch unterscheiden. Eine solche Software, die die Aufgabe hat, Quantenalgorithmen in Quantenschaltungen entsprechend den Hardwarebeschränkungen zu übersetzen, wird als Quantencompiler bezeichnet. Zusammenfassend lässt sich sagen, dass ohne einen guten Compiler die meisten Quantenalgorithmen in der Praxis nicht anwendbar sind.

Quantencompiler übernehmen typischerweise drei Hauptaufgaben: Quantenzustandsvorbereitung, Schaltungssynthese und Qubit-Mapping. Die Vorbereitung des Quantenzustands umfasst die Vorbereitung des Anfangszustands des Quantensystems, bevor der Algorithmus ausgeführt wird. Die Herstellung eines solchen Zustands selbst erfordert eine Berechnung, die von einem Quantenschaltkreis durchgeführt wird. Diese Aufgabe kann eine Herausforderung darstellen, da Quantenalgorithmen einige spezifische Anfangszustände in Überlagerung annehmen, bevor sie die gewünschten anwendungsspezifischen Berechnungen durchführen. Die Schaltungssynthese umfasst den Prozess des Aufbaus einer Quantenschaltung, die den gewünschten Quantenalgorithmus implementiert. Qubit-Mapping ist eine weitere wichtige Aufgabe für Quantencompiler, bei der die logischen Qubits des Algorithmus auf die physischen Qubits des Quantencomputers abgebildet werden. In dieser Arbeit konzentriere ich mich auf die ersten beiden Aufgaben von Quantenkompilierern, Schaltungssynthese und

Quantenzustandspräparation. Diese Aufgaben sind für die erfolgreiche Ausführung von Quantenalgorithmen unerlässlich, und die Entwicklung effizienter und effektiver Algorithmen für diese Aufgaben ist entscheidend für die kontinuierliche Weiterentwicklung des Quantencomputings.

Schlüsselwörter: Logische Synthese, Boolesche Funktionen, Entscheidungs diagramme, Quantencomputer, Quanten-Compiler, Quantenzustandsvorbereitung.

Résumé

L'informatique quantique a fait d'importants progrès ces dernières années, Google et IBM ayant mis sur le marché des ordinateurs quantiques dotés respectivement de 72 et 50 qubits. Google a également atteint la suprématie quantique avec son dispositif de 54 qubits, et IBM a annoncé la sortie de l'ordinateur quantique Osprey avec 433 qubits. Ces développements suggèrent que des ordinateurs quantiques avec un nombre de qubits encore plus élevé pourraient être disponibles dans un avenir proche. Cette période à venir est appelée l'ère *Noisy Intermediate Scale Quantum* (NISQ), en raison des caractéristiques bruyantes des dispositifs à court terme.

Le calcul sur le matériel NISQ est modélisé à l'aide d'une bibliothèque de portes quantiques prises en charge qui peuvent être directement mises en œuvre et d'un graphe de couplage qui spécifie les interactions de qubits disponibles pour les portes quantiques multi-qubits. Alors que les expérimentateurs ne cessent d'apporter des améliorations au matériel quantique, les experts en informatique quantique peuvent contribuer à l'utilité de ce matériel. Les experts en informatique quantique peuvent contribuer à l'utilité des dispositifs quantiques en développant des logiciels. Ce logiciel viserait à adapter les algorithmes quantiques classiques (par exemple, pour la factorisation ou la simulation quantique) aux contraintes matérielles, qui incluent : (1) le nombre limité de qubits, (2) une connectivité limitée entre les qubits, (3) des jeux de portes spécifiques au matériel limités, et (4) une profondeur de circuit limitée en raison du bruit. Les algorithmes adaptés à ces contraintes seront probablement très différents de leurs équivalents dans les manuels. Un tel logiciel, qui traduit les algorithmes quantiques en circuits quantiques en fonction des contraintes matérielles, est appelé compilateur quantique. En conclusion, sans un bon compilateur, la plupart des algorithmes quantiques ne sont pas applicables en pratique.

Les compilateurs quantiques entreprennent généralement trois tâches principales : la préparation de l'état quantique, la synthèse de circuits et la cartographie des qubits. La préparation de l'état quantique consiste à préparer l'état initial du système quantique avant d'exécuter l'algorithme. La préparation d'un tel état nécessite elle-même un calcul effectué par un circuit quantique. Cette tâche peut être difficile, car les algorithmes quantiques supposent certains états initiaux spécifiques en superposition avant d'effectuer les calculs spécifiques à l'application souhaités. La synthèse de circuits implique le processus de construction d'un circuit quantique qui implémente l'algorithme quantique souhaité. Le mappage des qubits est une autre tâche importante pour les compilateurs quantiques, qui consiste à mapper les qubits logiques de l'algorithme aux qubits physiques de l'ordinateur quantique. Dans cette thèse, je me concentre sur les deux premières tâches des compilateurs quantiques, la synthèse de circuits et la préparation d'états quantiques. Ces tâches sont essentielles pour l'exécution réussie des algorithmes quantiques, et le développement d'algorithmes efficaces et efficients pour ces tâches est crucial pour l'avancement continu de l'informatique quantique.

Mots-clés : Synthèse logique, fonctions booléennes, diagrammes de décision, calcul quantique, compilateur quantique, préparation d'états quantiques.

Contents

List of Figures

List of Tables

1 Introduction

The concept of quantum computing was first introduced by the physicist Richard Feynman in 1981 [5]. He proposed that quantum computers could be used to simulate quantum systems that are too complex for classical computers. In 1994, the mathematician Peter Shor developed a quantum algorithm for factoring large numbers, which can be exponentially faster than any known classical algorithm [6]. This algorithm demonstrated the potential of quantum computers to solve problems that are beyond the reach of classical computers. In 1995, the first experimental demonstration of quantum computing was performed by the physicist Isaac Chuang [7] and his colleagues at IBM, who implemented Shor's algorithm using a small-scale quantum computer based on nuclear magnetic resonance technology [8].

There are several types of quantum computer technologies developed by various companies and research institutions, such as:

- Superconducting qubits: This architecture uses superconducting circuits to create qubits [9, 10]. The qubit states are changed by applying microwave pulses. These qubits are easy to control as they can be easily integrated with classical electronic circuits. Superconducting qubits are highly scalable, but their creation requires extremely low temperatures near absolute zero. Despite their advantages, these qubits face challenges, including decoherence and error rates. Decoherence occurs when qubits lose their quantum information over time due to interactions with the environment. Additionally, quantum operations on superconducting qubits are subject to errors, necessitating the development of advanced error correction techniques to maintain computational accuracy. Companies such as IBM, Google, and Rigetti are working on developing superconducting qubit-based quantum computers. Google and IBM released quantum computers with 72 and 50 qubits, respectively [11, 12] in the beginning. While IBM has announced the release of the Osprey quantum computer with 433 qubits [13], recently.
- Ion trap qubits: This architecture uses ions suspended in an electromagnetic field to create qubits [14]. Multiple ions are trapped in a vacuum and the state of qubits is specified using the energy level of ions, which are manipulated using laser pulses. The main advantage of ion qubits is their long coherence time, which means they can keep quantum states safe for a long time. However, ion trap quantum computers face challenges such as scalability, readout error, physical footprint, speed, and gate fidelity.

Companies such as IonQ and Honeywell are developing ion trap qubit-based quantum computers. IonQ's quantum computer has 32 qubits.

- Photonic qubits: This architecture uses photons to create qubits [15, 16]; Information is encoded using polarization of photons and manipulated by various optical components. Photonic qubits are able to transmit over long distances using optical fibers, which makes them well-suited for quantum communication applications. The main challenge is the difficulty of generating and manipulating single photons with high efficiency and fidelity. Another challenge is the lack of a universal gate set. Companies such as PsiQuantum and Xanadu are developing photonic qubit-based quantum computers. PsiQuantum aims to build a million-qubit photonic quantum computer.

- Topological qubits: This architecture uses topological materials such as Majorana fermions or anyons to implement qubits [17]. The state of the qubit is protected from outside influences by the topology, or geometry, of the qubit itself. These materials exhibit exotic properties that allow quantum information to be stored in a highly robust manner. Hence, a topological qubit-based quantum computer is very useful for fault-tolerant computing. The main challenge is the need for extremely low temperatures and precise control over the qubits. Microsoft is one of the major player working on developing topological qubit-based quantum computers. Its topological quantum computer, called Station Q, is still in the research phase.

- Neutral atom qubits: This architecture uses arrays of single neutral atoms manipulated by light beams to encode and read out quantum states. Each qubit in this architecture is defined by one of two electronic states of an atom, and these single neutral atoms can be arranged in configurable arrays, similar to classical registers. One advantage of neutral atom quantum computing is that the neutral atoms used as qubits offer long coherence times. The main challenges include the limited two-qubit gate fidelities and gate operation speeds.

It is worth noting that the number of qubits in a quantum computer is constantly evolving as companies and research institutions continue to improve and scale their systems. The current period in quantum computing is often referred to as the *Noisy Intermediate Scale Quantum* (NISQ) era [18], due to the noisy and error-prone characteristics of near-term quantum devices. In this thesis, I focus on superconducting-based quantum computers, which has achieved impressive progress in terms of qubit counts and coherence times in recent years.

Quantum computing is an emerging field that has the potential to revolutionize the way of processing information. Traditional computers use classical bits, which can only be in one of two states (0 or 1) at any given time. However, quantum computers use quantum bits, or qubits, which can exist in multiple states simultaneously. This allows quantum computers to perform certain calculations exponentially faster than classical computers, making them well-suited for tackling complex problems in fields such as cryptography [19, 20], chemistry [21], machine learning [22], materials science [23], and algorithms for quantum linear equations [24].

Superposition and entanglement are both key features of quantum systems that make them fundamentally different from classical systems [25]. Superposition is the principle that a quantum system can exist in multiple states simultaneously. For example, an electron can exist in a superposition of two different energy levels at the same time. This is different from classical systems, where a system can only be in one state at a time. Superposition is critical for quantum computing because it allows qubits to represent many more states simultaneously than classical bits. Entanglement is the phenomenon where two or more quantum systems become correlated in such a way that their states are interdependent. This means that the state of one particle can instantaneously affect the state of the other particle, even if they are separated by large distances. Entanglement is important for quantum computing because it allows for the creation of quantum gates that can manipulate the state of multiple qubits at the same time.

In recent years, there has been a surge of interest in quantum computing from both academia and industry. Major technology companies such as IBM, Google, Microsoft, and Rigetti have all invested significant resources in the development of quantum computing technologies. These companies are working to build quantum computers with ever-increasing numbers of qubits and to develop software tools and applications that can run on these machines.

One of the key challenges facing the quantum computing industry is the issue of scaling. While researchers have been able to build small-scale quantum computers with a handful of qubits, it remains a significant challenge to build large-scale, fault-tolerant machines that can outperform classical computers on a wide range of tasks. Additionally, there are also significant challenges in developing software tools and algorithms that can run efficiently on these devices while taking into account specific hardware constraints. These constraints include the coupling graph, which shows the connectivity between qubits, the available gate set library, and the number of qubits. Several studies, including those referenced in [26, 27, 28, 29, 30, 31, 32, 33, 34], have focused on incorporating these hardware constraints into the design of quantum algorithms and circuits synthesis. These studies aim to minimize the cost of quantum computation by reducing the number of gates and the circuit depth.

1.1 Contribution

Quantum computers are physical machines that consist of an array of qubits, which in contrast to classical bits, can be in a superposition state and can be entangled [25]. The state of an array of n qubits is described in terms of 2^n complex-valued amplitudes, which can be denoted as $|\varphi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle$. Each norm squared amplitude corresponds to the probability of the quantum state being in one of the 2^n possible Boolean states after measuring all qubits. A quantum state can be altered by means of a set of quantum operations, typically referred to as quantum gates.

In the last few years several physical implementations of quantum computers have been demonstrated by, e.g., Google [11], IBM [12], Rigetti [35], Intel [36], IonQ [37], and Alibaba [38]. The number of qubits is often considered the primary distinguishing characteristic of quantum computers; however, several other factors significantly impact the quality and performance of a quantum computer. These factors include qubit coherence times, coupling constraints that determine which qubits can interact with each other during quantum operations, and the supported quantum gate set that defines the available operations for manipulating qubits.

In addition to the development of the hardware for quantum computing, there has been a growing interest in the development of quantum compilers. These are tools that can take high-level quantum algorithms and automatically translate them into low-level instructions that can be executed on a specific quantum computer architecture.

One of the key challenges in quantum computing is that there are many different types of hardware architectures and constraints, each with its own strengths and weaknesses. This means that quantum algorithms must be tailored to the specific hardware they will be run on, which can be a time-consuming and challenging task. Quantum compilers aim to simplify this process by automating the translation of algorithms into executable instructions for specific hardware architectures. There are several different approaches to developing quantum compilers, each presenting its own advantages and disadvantages. Some compilers use a brute-force approach, generating a large number of possible circuit implementations and selecting the most efficient one. Other compilers use heuristic algorithms to guide the circuit synthesis process. In addition, some others use machine learning techniques to optimize the translation process.

One of the key challenges facing quantum compilers is the issue of optimization. As quantum computers continue to increase in scale and complexity, the need for automated tools for translating high-level algorithms into low-level hardware instructions becomes increasingly important. Quantum circuits can be very complex, and even small changes to the circuit can have a significant impact on its performance. Therefore, developing efficient and accurate optimization techniques is essential for the development of practical quantum compilers. Quantum compilers have the potential to make quantum computing more accessible and usable for a wider range of researchers and developers, which could help accelerate the development of practical quantum applications.

This thesis aims to present methods to automate and optimize quantum compilers for several important tasks. As the first contribution, I propose a compilation algorithm to permute a given set of 2^n elements which is very important in combinatorial optimization. Another task addressed in this thesis is preparing quantum states. Loading classical data into quantum registers is one of the most important primitives of quantum computing. Moreover, Some quantum algorithms require a specific quantum state at the beginning of the computation. Hence, in addition to the quantum circuit that performs the quantum algorithm, a specific quantum circuit is required that prepares the desired quantum state, and called *quantum state preparation*. Consequently, an efficient quantum state preparation is an important task for quantum compilers. By developing methods that automate and optimize these crucial tasks, this thesis aims to contribute to the progress and enhancement of quantum compiler design and performance.

I propose an automatic hardware-dependent compilation algorithm to translate quantum operations that realizes permutations. The algorithm takes as input a permutation over 2^n elements, the gate library, and coupling constraints of the targeted quantum computer. It returns a quantum circuit composed of gates from the gate library, which respects the coupling constraints. The proposed approach utilizes Young-subgroup based reversible logic synthesis [39], which for a given permutation over n qubits, finds a sequence of $2n-1$ single-target gates. I describe a general algorithm to translate a single-target gate into a quantum circuit composed of Clifford+Rz library gates. Finally, I employ an explicit rewiring technique in order to reduce the number of quantum gates.

The preparation of quantum states is performed by a quantum circuit consisting of Controlled-NOT (CNOT) and single-qubit quantum gates. Known algorithms to prepare arbitrary n-qubit quantum states create quantum circuits in $O(2^n)$ runtime and use $O(2^n)$ CNOTs, which are more expensive than single-qubit gates in NISQ architectures. Some approaches [40, 41, 42, 43, 44, 45] have been considered in the past to prepare arbitrary quantum states. Since these approaches can generate arbitrary quantum states, the input to such algorithms is 2^n complex-valued amplitudes, which limits their scalability drastically. Further, some of the algorithms require a rather abstract set of gates, and an additional compilation step in order to run on physical quantum computers. To reduce runtime and the number of CNOTs, I simplify the problem by considering important families of quantum states, which are *Uniform Quantum States* (UQSs), *cyclic quantum states*, and *sparse quantum states*.

A uniform quantum state is a superposition of a non-empty subset of basis states, where all basis states have equal probability amplitudes. In other words, all non-zero amplitudes in such a state have the same value. Uniform quantum states are important because they are considered as initial quantum state for algorithms such as Grover walk [46]. Moreover, many important quantum states are uniform, such as the Bell state, the W state [47], the GHZ state [48], and the uniform superposition of *all* basis states. The W and GHZ states are used as fundamental resources in distributed quantum information processing [49]. Such states can be characterized by a Boolean function where each minterm corresponds to a non-zero amplitude. As a result, the quantum state can be represented in a compact form, if the Boolean function permits a compact representation. Preparing UQSs using Boolean functions can help to take advantage of different representations of Boolean functions.

I propose a quantum state preparation algorithm that works directly on decision diagrams, which is a symbolic representation of Boolean functions. This enables a scalable quantum state preparation, since many Boolean functions of practical interest have small representations, e.g., in terms of binary decision diagrams (BDDs) [50]. The algorithm produces a sequence of multi-controlled gates. Afterwards, to run on a physical quantum computer, I use decomposition methods to generate a quantum circuit over CNOTs and single-qubit quantum gates. The detailed contributions are summarized as follows:

- Utilizing Boolean functions in order to provide a recursive algorithm.
- Proposing an algorithm that works on BDDs to enable a fast execution when the function representation is small (algorithm runs in polynomial time with respect to the number of BDD nodes).
- Reducing the number of elementary quantum gates by removing redundancies in the BDDs as well as applying a post-optimization technique for the GHZ state.

Experimental results show that the proposed approach can achieve a significant reduction in runtime compared to a state-of-the-art approach which relies on an explicit quantum state representation implemented in IBM's Qiskit quantum programming framework. Moreover, the results show that I can reduce the number of elementary quantum gates over the state of the art.

To further reduce the number of CNOTs, I utilize variable reordering and functional dependencies among the variables. My state preparation method requires an exponential number of CNOTs in the worst case but it reduces CNOTs significantly for practical benchmarks. Moreover, my method generates an exact representation of quantum states without using any ancillary qubits. To evaluate the efficacy of my approach, I compare the results generated by my algorithm to those obtained using Qiskit. The comparison indicates that my method can reduce the average number of CNOTs required by 75.31% for practical benchmarks. Moreover, the runtime is almost halved as a result of utilizing my technique.

Multipartite quantum states that remain invariant under permutations, such as Dicke states, possess unique and interesting properties. In my research, I focus on quantum states that remain invariant under cyclic permutations, which I refer to as cyclic states. I propose a quantum algorithm that deterministically prepares cyclic states, with a gate complexity of $O(n)$, without requiring any ancillary qubits. Through both theoretical analysis and experimentation, I show that my algorithm is more efficient than existing ones.

Preparing a generic quantum state can be an exponentially complex task with respect to the number of qubits involved. However, in many practical tasks the state to be prepared possesses a particular structure that allows for a sparse and compact representation, enabling faster preparation. I mainly consider sparse quantum states, where the number of non-zero amplitudes is relatively small. I propose an algorithm that utilizes the structure of decision diagrams to prepare the corresponding quantum states efficiently. The algorithm takes advantage of the sparsity of the quantum states and can be implemented with a circuit complexity that is linear in the number of paths in the decision diagram. Numerical experiments show that my algorithm reduces the circuit complexity by up to 31.85% compared to the state-of-the-art algorithm, when preparing generic n-qubit states with n^3 non-zero amplitudes. Additionally, for states with sparse decision diagrams, including the initial state of the quantum Byzantine agreement protocol, my algorithm reduces the number of CNOTs by 86.61% ~ 99.9%.

Finally, I present *angel*, a C++ library for quantum state preparation. The *angel* library is designed to synthesize optimized quantum circuits for specific quantum states. My proposed algorithms for preparing uniform quantum states and sparse quantum states are implemented in this library. The objective function of these algorithms is to minimize the depth of the quantum circuit and the number of elementary quantum gates required. The algorithms also aim to reduce the number of CNOTs since they tend to be relatively expensive compared to other elementary quantum gates in many experimental NISQ architectures. The *angel* library is open-source and it can be freely downloaded, used, and modified by anyone. It provides a useful tool for researchers in the field of quantum computing who are interested in optimizing the preparation of specific quantum states.

The above contributions are published in [51, 52, 53, 4, 54, 55, 56]. Moreover, other publications I contributed include [57, 58, 59].

1.2 Outline

This thesis is organized as follows:

- **Chapter 2:** introduces the field of logic synthesis. It defines Boolean functions as well as various data structures for representing them. Spectral techniques used in this field is introduced. Finally, it covers the ESOP synthesis method.
- **Chapter 3:** aims to provide readers with the necessary background knowledge of quantum computing to understand the remaining content of the thesis. It begins with an overview of basic quantum computing concepts, such as Dirac notation, qubits, quantum states, gates, Bloch sphere representation, and measurement. Moreover, the chapter provides a detailed description of quantum circuits and important quantum algorithms.
- **Chapter 4:** the topic of quantum circuit synthesis and my specific contribution in compiling single-target gates. I discuss how this technique can be applied to compile permutations and achieve more efficient circuit implementations. Additionally, I detail my approach to optimizing circuits through rewiring techniques, which can further reduce the number of gates required for a given circuit.
- **Chapter 5:** introduces the problem of quantum state preparation, which is a fundamental task in quantum computing.
- **Chapter 6:** provides an introduction to uniform quantum states and their importance. Afterwards, it presents my proposed methods to efficiently prepare them. I utilize Boolean functions and logic synthesis techniques, including decision diagram representations and functional dependency analysis, to outperform current state-of-the-art methods.
- **Chapter 7:** presents another family of quantum states called cyclic quantum states that exhibit interesting properties. This chapter shows the method that I propose to prepare these states. By carefully designing the circuit, I prepare cyclic quantum states in just linear circuit size. This is a significant improvement over previous methods, which typically require much larger circuits to prepare cyclic quantum states.
- **Chapter 8:** presents a novel contribution to the field of quantum computing, as it offers a new method for preparing sparse quantum states that can be efficiently represented by decision diagrams. The algorithm has a circuit complexity that is linear in the number of paths in the decision diagram. Furthermore, for states with sparse decision diagrams, the algorithm significantly reduces the number of CNOTs.
- **Chapter 9:** introduces an open-source C++ library dedicated to the quantum state preparation called *angel*. This library offers algorithms for preparing uniform quantum states using both *Binary Decision Diagrams* (BDDs) and truth tables. Additionally, the library provides functional dependency analysis to identify dependencies between qubits and improve the efficiency of the preparation process. Furthermore, the library also includes a method for preparing sparse quantum states using the *Algebraic Decision Diagrams* (ADDs). The *angel* library offers a modular and extensible framework, making it a useful tool for researchers in the field of quantum computing.

Background Part I

2 Logic Synthesis

Logic synthesis is the process by which a high-level description, typically at *Register Transfer Level* (RTL), is transformed into an optimized gate-level representation, using a computer program called a synthesis tool [60, 61]. A standard cell library in logic synthesis consists of simple logic gates, such as AND, OR, and NOR, or macrocells, such as ADDERs, MUXes, Memory, and Flip-Flops. A collection of standard cells is called a technology library.

Logic synthesis techniques are very helpful in many applications such as quantum compilation. In this chapter, I present useful logic data structures and techniques which I utilize in compiling quantum circuits.

2.1 Boolean functions

In this section, I define Boolean functions [62, 63] and their properties. Boolean functions are very helpful in the context of quantum state preparation, especially when the state to be prepared is a uniform superposition state. Boolean functions can also be used to define quantum oracles, which are black boxes that perform a specific computation. Quantum oracles can be used in various quantum algorithms, such as Grover's algorithm for quantum search and Shor's algorithm for integer factorization.

Let $\mathbb{B} = \{0, 1\}$, which represents a binary space.

Definition 2.1.1. *A Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}$ of n Boolean variables $x = x_1, \ldots, x_n$ is a mapping of n Boolean input values $\hat{x} = \hat{x}_1, \ldots, \hat{x}_n$ to a single Boolean output value $f(\hat{x})$. Each Boolean function is given as input 2^n input assignments 0, 1, …, $2^n - 1$.*

Definition 2.1.2. *A Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ is multi-output when $m > 1$.*

Definition 2.1.3. *A Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ is reversible if and only if f is a one-to-one function which $m = n$ and performs a permutation of the set of input assignments.*

Definition 2.1.4. *The positive cofactor of $f(x_1, x_2, \ldots, x_i, \ldots, x_n)$ with respect to variable x_i is $f_{x_i} = f(x_1, x_2, \ldots, 1, \ldots, x_n)$. Similarly, the negative cofactor of $f(x_1, x_2, \ldots, x_i, \ldots, x_n)$ with respect to variable $\bar{x}_i$ is $f_{\bar{x}_i} = f(x_1, x_2, \ldots, 0, \ldots, x_n)$.*

I use cofactors to compute the influence of each variable on the function's output. To compute the probability of x_i being 1 and 0 in f, I define

$$p_f(x_i) = \frac{|f_{x_i}|}{|f|} \quad \text{and} \quad p_f(\bar{x}_i) = \frac{|f_{\bar{x}_i}|}{|f|}, \tag{2.1}$$

respectively. In this notation, $|f|$ shows the number of input assignments in which their corresponding output values are 1.

Note that $p_f(x_i) + p_f(\bar{x}_i) = 1$. Moreover, I utilize *Shannon's decomposition* theorem, which states $f = x_i f_{x_i} + \bar{x}_i f_{\bar{x}_i}$ for variables x_i of f. The intuition is that Shannon's decomposition uses cofactors to partition a function into two halves.

2.2 Boolean function representations

Here, I present several forms to represent Boolean functions, which are frequently used by logic synthesis algorithms. Before going through representations, I will provide some definitions as follows.

In this thesis, for convenience I define a *literal* as a Boolean variable x_i^j, $j \in \{0,1,2\}$ in which

$$x_i^j = \begin{cases} \bar{x}_i, & \text{if } j = 0 \\ x_i, & \text{if} j = 1 \\ 1, & \text{if} j = 2 \end{cases} \tag{2.2}$$

A *product term* (or *cube*) $t(x)$ is a conjunction of literals as

$$t(x) = \prod_{i=1}^{n} x_i^j. \tag{2.3}$$

The size $|t|$ of a product term t is the number of literals appearing in t.

2.2.1 SOP form

The *sum-of-products* (SOP) is a form of simplifying the Boolean expressions of logic gates. In the SOP form, the variables are operated by AND (product) to form a product term and all these product terms are ORed (summed or added) together to get the final function. The sum-of-products form is also called *Disjunctive Normal Form* (DNF) [64] as the product terms are ORed together and the Disjunction operation is logical OR.

Example 2.2.1. *Let $f(x_1, x_2, x_3) = \langle x_1, x_2, x_3 \rangle$ be majority-of-three (majority-3) function. Its SOP is represented by*

$$f(x_1, x_2, x_3) = \langle x_1, x_2, x_3 \rangle = x_1 x_2 + x_1 x_3 + x_2 x_3. \tag{2.4}$$

2.2.2 POS form

In *product-of-sums* (POS) form, variables are ORed, i.e. written as sums to form sum terms, and all these sum terms are ANDed (producted) together to get the final form. This form is exactly the opposite of the SOP form which can be said it is a dual of SOP form. POS form is also called *Conjunctive Normal Form* (CNF).

Example 2.2.2. *Consider majority-3 function $f(x_1, x_2, x_3)$. A POS form is*

$$f(x_1, x_2, x_3) = (x_1 + x_2)(x_1 + x_3)(x_2 + x_3). \tag{2.5}$$

2.2.3 Minterm canonical form

A product term in which all Boolean variables $x_1, \dots, x_n$ appear exactly once is a *minterm*. Each Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ can be uniquely represented in its *minterm canonical form*

$$\begin{aligned} f(x) = \quad & f(0,0,\dots,0) \cdot \bar{x}_1 \bar{x}_2 \cdots \bar{x}_n + \\ & + f(0,0,\dots,1) \cdot \bar{x}_1 \bar{x}_2 \cdots x_n + \\ & \vdots \\ & + f(1,1,\dots,1) \cdot x_1 x_2 \cdots x_n, \end{aligned} \tag{2.6}$$

where $f(\hat{x}_1, \dots, \hat{x}_n)$ are Boolean values, called *discriminants*, and $x_1^{\hat{x}_1} x_2^{\hat{x}_2} \dots x_n^{\hat{x}_n}$ are product terms. Consequently, the set

$$\mathsf{Minterms}(f) = \{x_1^{\hat{x}_1} \dots x_n^{\hat{x}_n} \mid f(\hat{x}_1, \dots, \hat{x}_n) = 1\} \tag{2.7}$$

of all minterms of a Boolean function f characterizes the function uniquely. The size $|f|$ of f is defined as the number of minterms of f.

Example 2.2.3. *Consider majority-3 function $f(x_1, x_2, x_3)$. A minterm canonical form is*

$$f(x_1, x_2, x_3) = \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 x_3 + x_1 x_2 \bar{x}_3 + x_1 x_2 x_3. \tag{2.8}$$

2.2.4 Truth table form

A truth table is an explicit representation where all input assignments in the Boolean input space ($\mathbb{B}^n$) and their corresponding output values are listed. The input part can be omitted if a specific ordering of input assignments is assumed. In general, a truth table for a Boolean function $f(x_1, ..., x_n)$ corresponds to a column vector $f = (f_{2^{n-1}}, \dots, f_0)^T$ containing 2^n values. In other words, it is a bit-string $b_{2^n-1} b_{2^n-2} \dots b_1 b_0$ containing 2^n bits, where $f(x) = b_x$ such that $x = (x_1 \dots x_n)_2$ is the integer representation of the input assignment. Consequently, I may also consider a truth table as a number in the half-open interval $[0, 2^{2^n})$, for which the truth table representation is the binary expansion of that number.

Example 2.2.4. *Consider a majority-3 function $f(x_1, x_2, x_3)$. A truth table representing the function is shown in Table 2.1. It can also be expressed as a bit-string of $2^3 = 8$ bits, 11101000. To prevent the rapid growth of binary notation, it is common to use hexadecimal notation,*

Table 2.1: Truth table representation of the majority-3 function.

$x_1x_2x_3$	f
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

where each block of 4 bits corresponds to a single hexadecimal digit. The majority-3 function is represented by the truth table #e8 *in hexadecimal. A hash prefix is used to indicate a hexadecimal number.*

Clearly, truth tables cannot provide a scalable function representation. Nevertheless, for small functions, they can be beneficial as they enable very fast implementations. For example, a truth table for a six-variable function requires $2^6 = 64$ bits and therefore fits into a single unsigned integer of a 64-bit computer architecture. Many operations, e.g., computing the AND of two functions can be performed using bitwise AND, which accounts for a single processor instruction. Such an approach works reasonably well in practice up to 16-variable functions, which require $2^{10} = 1024$, 64-bit unsigned integers, and therefore 8 KB of memory. A truth table is a canonical (i.e., unique) representation of a function. Consequently, for small functions, the truth tables can be used for a simple equivalence check of two functions, if a truth table can be efficiently derived from them.

2.2.5 ESOP form

Each Boolean function can also be represented in *Exclusive-or Sum-Of-Products* (ESOP) forms. ESOPs are a classical two-level logic representation consisting of one level of AND gates, followed by one level of XOR gates. Hence, an ESOP form in Boolean variables $x = x_1, \ldots, x_n$ is a Boolean function

$$f(x) = t_1(x) \oplus \cdots \oplus t_m(x), \tag{2.9}$$

where m is a positive integer called the *degree* of the ESOP form, and $t_1, \ldots, t_m$ are product terms.

ESOPs provide a compact logic representation of Boolean functions, and are, for some classes of functions, exponentially more compact when compared to the SOP representation. The ESOP representation of a Boolean function is not unique, i.e., the same Boolean function can be expressed as multiple structurally different, but semantically equivalent ESOP forms. In practice, it is important to find a small representation of an ESOP form to reduce the overall costs for realizing it in hardware or implementing it in software.

The problem of synthesizing an ESOP form for a given Boolean function is to identify a set of product terms over the Boolean variables of the function such that each minterm in the

OFF-set (set of minterms for which the Boolean function evaluates to 0) of the function is covered by the product terms an even number of times and each minterm in the ON-set (set of minterms for which the Boolean function evaluates to 1) of the Boolean function is covered an odd number of times. Exact algorithms, such as [65, 66], and heuristics, such as [67], have been proposed for finding and minimizing the degree of an ESOP form of a Boolean function f.

ESOP forms play an important role in logic synthesis due to their improved compactness for arithmetic and communication circuits with respect to other two-level representations [68] and their excellent testability properties [69]. The inherent reversibility of the XOR operation, moreover, makes ESOP forms particularly suitable in applications such as security [70]. Moreover, ESOP forms have applications in quantum compilation flows since they allow us to decompose multi-controlled single-target gates into a sequence of generalized Toffoli gates [57, 71, 72, 73, 74, 75].

Example 2.2.5. *Given majority-3 function $f(x_1, x_2, x_3)$, one possible ESOP representation for f is*

$$f(x_1, x_2, x_3) = x_1x_2 \oplus x_1x_3 \oplus x_2x_3. \tag{2.10}$$

2.2.6 Decision diagram form

The *Decision Diagram* (DD) form of a Boolean function is a symbolic representation that shows a compact representation for it. Here I give a brief introduction to DDs.

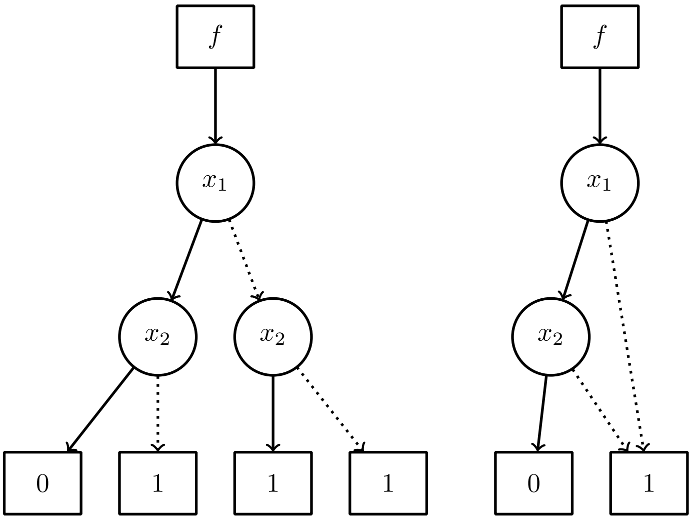
Binary decision diagram. A *Binary Decision Diagram* (BDD) [76, 77, 78] is a rooted, directed acyclic graph ($f \cup \{0,1\} \cup V$) representing a Boolean function f where

- f represents the root node with out-degree one.
- $\{0,1\}$ are terminal nodes with out-degree zero.
- V is the set of nodes corresponding to a set of variables $x = x_1, \ldots, x_n$ of out-degree two. The two outgoing edges of each x_i are given by two functions $low(x_i)$ and $high(x_i)$ which represent that x_i evaluates to 0 and 1, respectively, and I call them zero-child and one-child. Note that in the symbolic representation, low and $high$ functions are represented by dotted and solid arrows.

A BDD is *Ordered* (OBDD) if on all paths through the graph, the variables respect a given linear order $x_1 < x_2 < \cdots < x_n$.

An OBDD is *Reduced* (ROBDD) if the rules below are applied:

1. Two nodes are merged and their incoming edges are redirected to the merged node, if i) they are both terminal and have the same value, or ii) they are both internal and have the same sub-graphs.
2. An internal node is eliminated, if its two edges point to the same child. After elimination, its incoming edges are redirected to the child.



(a) Ordered BDD.

(b) Reduced Ordered BDD.

Figure 2.1: The OBDD and ROBDD for $f = x_1\bar{x}_2 + \bar{x}_1 x_2 + \bar{x}_1\bar{x}_2$.

ROBDDs have some interesting properties. They provide compact representations of Boolean expressions, and there are efficient algorithms for performing all kinds of logic operations on ROBDDs. They are all based on the crucial fact that for any function $f : \mathbb{B}^n \to \mathbb{B}$ there is exactly one ROBDD representing it. This means, in particular, that there is exactly one ROBDD for the constant true (and constant false) function on $\mathbb{B}^n$: the terminal node 1 (and 0 in case of false). Hence, it is possible to test in constant time whether an ROBDD is constantly true or false. (Recall that for Boolean expressions this problem is NP-complete.)

Example 2.2.6. *Fig. 2.1 shows the OBDD and ROBDD of f* $= x_1\bar{x}_2 + \bar{x}_1 x_2 + \bar{x}_1\bar{x}_2$. *The ROBDD shown in 2.1.b is obtained from the OBDD in Fig. 2.1.a by applying reduction rules. First, three terminal nodes with value 1 merge into one. Next, node b in the right-side of the tree eliminates as both children are terminal node 1.*

It is worth mentioning that ROBDD commonly referred to as BDD for simplicity. Therefore, I use the term BDD to refer to ROBDD in this thesis.

Algebraic decision diagram. An *Algebraic Decision Diagram* (ADD) is similar to a BDD, except that its terminal nodes can have any values [79] different than the set $\{0,1\}$. In other words, BDDs are ADDs whose terminal nodes have binary values. I can still apply reduction rules and get a ROADD, called ADD for short.

An ADD is a tuple (x, S, π, G) where

- x is a set of Boolean variables $\{x_1, x_2, \ldots, x_n\}$,
- S is an arbitrary set of values.
- $\pi : X \to \mathbb{Z}^+$ shows diagram variable order.

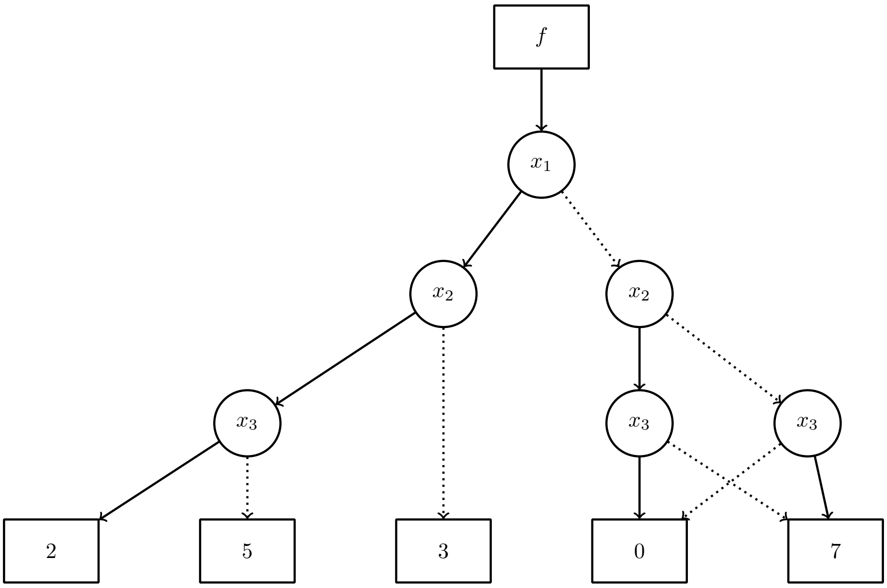


Figure 2.2: The ADD representation for the function f in example 2.2.7.

- G is a rooted, directed acyclic graph satisfying the following three properties. First, every terminal node of G is labeled with an element of S. Second, every non-terminal node of G is labeled with an element of X and has two outgoing edges labeled 0 and 1. Finally, for every path in G, the labels of the visited non-terminal nodes must occur in increasing order under π.

An ADD (x, S, π, G) is a compact representation of a function $f : \mathbb{B}^n \to S$. Although there are many ADDs representing each such function f, for each injection $\pi : X \to \mathbb{Z}^+$, there is a unique minimal ADD that represents f with π as the diagram variable order, called the canonical ADD. ADDs can be minimized in polynomial time, so it is typical to only work with canonical ADDs.

ADDs were originally designed for matrix multiplication and shortest path algorithms [79]. ADDs have also been used for stochastic model checking [80, 81, 82].

Example 2.2.7. *Fig. 2.2 shows the ADD representation for function*

$$f = 2x_1x_2x_3 + 5x_1x_2\bar{x}_3 + 3x_1\bar{x}_2 + 7\bar{x}_1x_2\bar{x}_3 + 7\bar{x}_1\bar{x}_2x_3, \tag{2.11}$$

with $S = \{0, 2, 3, 5, 7\}$.

2.3 Spectral technique

The Boolean function $f(x_1, \ldots, x_n)$ can be represented in terms of a *truth table* form as a column vector $f = (f_{2^n-1}, \ldots, f_0)^T$.

The $\{-1, 1\}$ encoding of a truth table is a column vector $\hat{f} = (\hat{f}_{2^n-1}, \ldots, \hat{f}_0)^T$ where $\hat{f}_i = 1 - 2f_i$ for $0 \le i < 2^n$. In other words, $\hat{f}_i = -1$, if $f_i = 1$, and $\hat{f}_i = 1$, if $f_i = 0$.

Example 2.3.1. *For majority-3 function $f(x_1, x_2, x_3) = (1,1,1,0,1,0,0,0)^T$, the $\{-1,1\}$ encoding equals to*

$$\hat{f} = (-1,-1,-1,1,-1,1,1,1)^T. \tag{2.12}$$

The recursive Hadamard matrix is

$$H_n = \begin{pmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{pmatrix}, \quad \text{and } H_0 = 1. \tag{2.13}$$

Then for an n-variable Boolean function f, the vector $s = H_n \hat{f}$ is called the Rademacher-Walsh *spectrum* of f. I will refer to s as spectrum.

Example 2.3.2. *For majority-3 function $f(x_1, x_2, x_3)$, the spectrum is*

$$s = (0,4,4,0,4,0,0,-4)^T. \tag{2.14}$$

2.4 SAT-based exact ESOP synthesis

The Boolean Satisfiability problem (SAT) is defined as follows:

Definition 2.4.1. *Suppose f is a Boolean function expressed in POS form, also known as CNF representation. The SAT problem involves determining whether there exists a truth assignment to the variables of f that makes f evaluate to true. If such an assignment exists, f is satisfiable. Otherwise, if no such assignment exists, f is unsatisfiable.*

SAT is the central NP-complete problem. Today SAT is not only used in theorem proving, but in many application domains like automatic test pattern generation, logic synthesis, and verification. In the last ten years significant improvements have been made in the area of SAT solvers. Several powerful tools have been developed that make use of Boolean constraint propagation and efficient learning techniques to speed up the proof process.

Authors in [66] present an exact synthesis approach for computing ESOP forms with a minimum number of product terms. Their approach starts with an incomplete Boolean function as a specification, and iteratively constructs a constraint satisfaction problem that can only be satisfied if and only if there is an ESOP form with k product terms (at first $k = 1$). An SAT-solver is used to solve the problem and, if satisfiable, an ESOP form is returned with k product terms. Alternatively, if k fails to satisfy, it is increased and the synthesis process is restarted. In the synthesis process, the number of Boolean variables is hardly important, and it is particularly fast when the Boolean function can be expressed using only a few product terms. ESOP-based synthesis is very useful in functional dependency analysis, which involves identifying relationships between variables in a Boolean function. During this thesis, I use functional dependency analysis to identify relationships between qubits, which can be used to optimize quantum circuit implementations.

2.5 Summary

Classical logic synthesis is a process used in *Electronic Design Automation* (EDA) tools to optimize logic circuits for factors such as performance, power consumption, and area utilization. The process involves converting a high-level *Hardware Description Language* (HDL) specification into a low-level representation of the circuit using logic gates. The output of the logic synthesis process is a gate-level netlist that can be used for further design verification and implementation. In this chapter, I introduced logic synthesis techniques that are useful during my thesis. I described Boolean functions, their various representations, i.e., SOP form, minterm canonical form, truth table form, ESOP form, and decision diagram form. Finally, SAT-based ESOP synthesis is presented.

3 Quantum Computing

3.1 Dirac notation

To distinguish a vector from a scalar, in mathematics and physics textbooks, an arrow over the identifying symbol, e.g. $\overrightarrow{a}$ is used. Quantum mechanics, use the *Dirac* notation, which was invented by Paul Dirac [83]. In Dirac notation, a vector is identified by a symbol inside a "*ket*" looking as $|a\rangle$ [84]. The dual vector for a is defined with a "*bra*", which is written as $\langle a|$. Afterwards, the inner product of two vectors a and b will be shown by "*bra-ket*" $\langle a|b\rangle$. In quantum mechanics, finite-dimensional vector spaces over the complex numbers are considered. Such vector spaces are a class of vector spaces called *Hilbert* spaces. In the Hilbert space, a basis is selected. In this basis which called *computational basis*, vectors can be represented by finite column vectors. The Dirac notation of 2^n basis states of binary strings of length n are

$$|00...00\rangle,\ |00...01\rangle,\ ...,\ |11...10\rangle,\ |11...11\rangle, \tag{3.1}$$

and their corresponding column vectors consisting of 2^n elements in Hilbert space are

$$\begin{pmatrix}1\\0\\\vdots\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\\vdots\\0\\0\end{pmatrix}, \ldots, \begin{pmatrix}0\\0\\\vdots\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\\vdots\\0\\1\end{pmatrix}. \tag{3.2}$$

In other definition, all elements are zero in the column vector except one that shows the index of the binary string in the Dirac notation.

Example 3.1.1. *Given the vector in Dirac notation as*

$$\sqrt{\frac{5}{4}}|00\rangle + \frac{i}{\sqrt{4}}|10\rangle, \tag{3.3}$$

it can be alternatively written as the column vector

$$\begin{pmatrix} \sqrt{\frac{5}{4}} \\ 0 \\ \frac{i}{\sqrt{4}} \\ 0 \end{pmatrix}. \tag{3.4}$$

3.2 Quantum bits & quantum states

Definition 3.2.1. *A quantum bit or a qubit is the elementary unit of information in quantum computation. A qubit describes a two dimensional quantum system.*

A qubit can be any superposition of the two basis states $|0\rangle = \binom{1}{0}$ and $|1\rangle = \binom{0}{1}$, which can be denoted as

$$|\varphi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}, \tag{3.5}$$

where $\alpha_0, \alpha_1 \in \mathbb{C}$, often called *amplitudes* of the basis states, and $|\alpha_0|^2 + |\alpha_1|^2 = 1$. The squared norm of amplitudes $|\alpha_0|^2$ and $|\alpha_1|^2$ indicate the probability that the quantum state will collapse to the classical state 0 or 1 after the qubit is measured. Whenever a qubit is measured, it automatically converts to a bit. Moreover, this definition can be extended to n-qubit quantum states.

Definition 3.2.2. *A quantum state over n qubits is a column vector of* 2^n *complex values defined as*

$$|\varphi\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{2^n-2} \\ \alpha_{2^n-1} \end{pmatrix}. \tag{3.6}$$

A quantum state over n qubits also can be represented by

$$|\varphi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle, \tag{3.7}$$

which is any combination of 2^n complex values α_i such that $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$. Each squared norm of amplitude $|\alpha_i|^2$ indicates the probability that after measurement the n qubits are in the classical state i.

Quantum states can be combined by applying the *Kronecker product* to produce larger ones, e.g., $\binom{1}{0} \otimes \frac{1}{\sqrt{2}}\binom{1}{1} = \frac{1}{\sqrt{2}}(1,1,0,0)^T$, which represents a 2-qubit state that is in the perfect **superposition** between the classical states 00 and 01 [85]. Moreover, a quantum state, such as $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ can not be represented as superposition state because it can not be written as two separate qubits. This state is an **entangled** state which measuring one qubit tells the state of the other qubits and collapse its superposition.

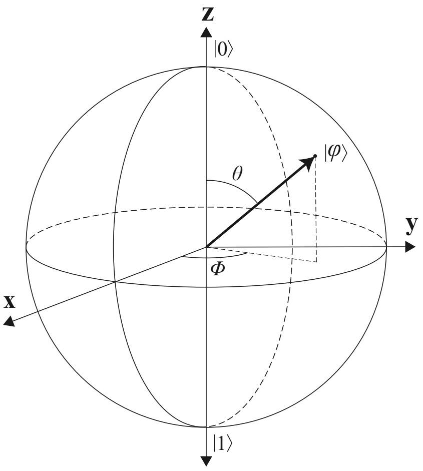


Figure 3.1: The representation of a quantum state $|\varphi\rangle$ on Bloch sphere.

3.3 Bloch sphere representation

As mentioned, the general state of a qubit is the same as Equation 3.5, where α_0 and α_1 are complex numbers. Alternatives to using complex numbers α_0 and α_1 include using real numbers as well as a term that indicates the relative phase between the two numbers as

$$\left|\varphi\right\rangle = \alpha_0|0\rangle + e^{i\phi}\alpha_1|1\rangle, \tag{3.8}$$

where $\alpha_0, \alpha_1, \phi \in \mathbb{R}$. As $|\alpha_0|^2 + |\alpha_1|^2 = 1$, one can use the trigonometric identity $\left|cos(\frac{\theta}{2})\right|^2 + \left|sin(\frac{\theta}{2})\right|^2 = 1$, which corresponds α_0 and α_1 to $cos(\frac{\theta}{2})$ and $sin(\frac{\theta}{2})$, meaning that α_0 and α_1 are described in terms of one variable θ. As a result, two parameters θ and ϕ can be used to describe the quantum state

$$\left|\varphi\right\rangle = cos(\frac{\theta}{2})|0\rangle + e^{i\phi}sin(\frac{\theta}{2})|1\rangle, \tag{3.9}$$

where θ and ϕ are real numbers. The numbers θ and ϕ specify a point on a three-dimensional sphere, called *Bloch sphere*. Fig. 3.1 shows the state $|\varphi\rangle$ on the Bloch sphere [86, 87].

3.4 Measurement

Measurement is an essential element in quantum circuits, which are typically performed at the end of the computation to extract the result of the computation. When a measurement is performed on a quantum state, the state "collapses" into one of its basis states. The measurement outcomes are classical bits that can be stored, transmitted, and processed in classical computers. It is important to note that measurements in quantum mechanics are inherently probabilistic and non-deterministic. Therefore, even if the same quantum state is prepared

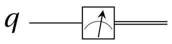


Figure 3.2: The symbolic representation of the measurement operator in quantum circuits.

and measured multiple times under identical conditions, the measurement outcomes may still vary due to the probabilistic nature of quantum mechanics. Fig. 3.2 shows the quantum circuit symbol for the measurement operator.

To determine the probability of measuring the state $|\varphi\rangle$ in the state $|i\rangle$, the squared modulus of the inner product between the two states is calculated. Mathematically, this is given by

$$p(|i\rangle) = |\langle i|\varphi\rangle|^2, \tag{3.10}$$

where $\langle i|\varphi\rangle$ is the inner product of the two states $|i\rangle$ and $|\varphi\rangle$.

For example, to compute the probability of being zero for a state, it is needed to compute $|\langle 0|\varphi\rangle|^2$.

3.5 Quantum gates

A *quantum gate* is an operation applied to one or more qubits to change their quantum state. A quantum gate that acts on n qubits can be specified by a $2^n \times 2^n$ unitary matrix [88, 25]. A matrix U is *unitary* if

$$U^\dagger U = UU^\dagger = I, \qquad U^\dagger = U^{-1}, \tag{3.11}$$

where the dagger '†' denotes the *conjugate transpose*. It is also named the *Hermitian adjoint*. Any gate with unitary matrix U that is its own unitary inverses is called *Hermitian* or *self-adjoint* operator which means

$$U^\dagger = U^{-1} = U. \tag{3.12}$$

Some quantum gates (which I will introduce in the subsection 3.5.1) such as the Hadamard (H), Identity (I), and the Pauli gates (X, Y, Z) are Hermitian operators, while others like the phase shift gates (S, T, P) are not.

The *matrix product* $A \cdot B$, the *direct sum* $A \oplus B$, and the *direct product* (also called *Kronecker product*) $A \otimes B$ of two matrices A and B, respectively, are defined as usual [25].

Depending on the number of qubits that quantum gates acting on, I classify them into 3 groups, single-qubit gates, two-qubit gates, and multi-qubit gates.

3.5.1 Single-qubit gates

A single-qubit quantum gate acts on a single qubit, and transforms its state into another state. The single-qubit gates are represented by 2 × 2 unitary matrices [88, 25]. These gates include *rotation* gates, *Pauli* gates, Identity, Hadamard, and Phase shift gates.

Since single-qubit states correspond to points on the Bloch sphere [25], quantum gates on a single-qubit correspond to rotations. There are three types of rotation gates R_x, R_y, and R_z regarding the three axes $\hat{x}$, $\hat{y}$, and $\hat{z}$. Each rotation gate is parametrized with a continuous angle $\theta \in [0, 2\pi]$.

These gates are defined by unitary matrices

$$R_x(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\mathrm{i}\sin\frac{\theta}{2} \\ -\mathrm{i}\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \tag{3.13}$$

$$R_y(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \tag{3.14}$$

and

$$R_z(\theta) = \begin{pmatrix} e^{-\mathrm{i}\frac{\theta}{2}} & 0 \\ 0 & e^{\mathrm{i}\frac{\theta}{2}} \end{pmatrix}. \tag{3.15}$$

Other important family of single-qubit gates are three Pauli gates X, Y, Z gates with Pauli matrices $\sigma_x, \sigma_y, \sigma_z$. They correspond to a rotation around the $\hat{x}$, $\hat{y}$, and $\hat{z}$ axes of the Bloch sphere by π radians. Note that the Pauli-X gate is the quantum equivalent of the NOT gate which transfers $|0\rangle$ to $|1\rangle$ and conversely $|1\rangle$ to $|0\rangle$. The Pauli-Y maps $|0\rangle$ to $i|1\rangle$ and $|1\rangle$ to $-i|0\rangle$. While the Pauli-Z leaves the basis state $|0\rangle$ unchanged and maps $|1\rangle$ to $-|1\rangle$. Note that the Pauli-Z may call a phase-flip gate. These gates are represented by

$$X = \sigma_x = NOT = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tag{3.16}$$

$$Y = \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \tag{3.17}$$

and

$$Z = \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{3.18}$$

The Identity gate does not affect the qubit's state, and it is represented by

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{3.19}$$

The Hadamard gate maps the basis state $|0\rangle$ to $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$, and $|1\rangle$ to $|-\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$. This gate specifies a rotation around $\frac{\hat{x}+\hat{z}}{\sqrt{2}}$ axis by π radian. It is defined by the following unitary matrix

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{3.20}$$

The phase shift gate leaves the basis state $|0\rangle$ unchanged and maps the basis state $|1\rangle$ to $e^{i\phi}|0\rangle$. This gate does not change the probability of measuring $|0\rangle$ and $|1\rangle$ basis states, while it changes the phase of quantum state. In other definition, it is equivalent to rotating in a horizontal circle on the Bloch sphere by ϕ radians. This gate and its dagger are represented by

$$P(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{\mathrm{i}\phi} \end{pmatrix}, \qquad P^{\dagger}(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{-\mathrm{i}\phi} \end{pmatrix}. \tag{3.21}$$

Some particular well-known phase shift gates are T gate and S gate. These gates and their daggers are defined as

$$T = P(\frac{\pi}{4}) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}, \qquad T^{\dagger} = \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\frac{\pi}{4}} \end{pmatrix}, \tag{3.22}$$

and

$$S = P(\frac{\pi}{2}) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \qquad S^{\dagger} = \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\frac{\pi}{2}} \end{pmatrix}. \tag{3.23}$$

In general, an arbitrary single-qubit gate with unitary matrix $U_{2\times 2}$ can be constructed by a combination of aforementioned single-qubit gates. For example, consider a single-qubit gate with the unitary matrix

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}, \tag{3.24}$$

I can construct it by a combination of NOT and H gates as

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \times \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{3.25}$$

Table. 3.1 shows the quantum circuit symbols for the single-qubit gates.

3.5.2 Two-qubit gates

These gates act on two qubits. In this thesis, I divide them into two groups, controlled gates and SWAP gate. For controlled gates, one qubit corresponds to a control-qubit and another to a target-qubit. When the control-qubit is $|1\rangle$, the target-qubit performs the operation, and when the control-qubit is $|0\rangle$ nothing happens. Two well-known controlled gates are CNOT and CZ with unitary matrices

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{3.26}$$

and

Table 3.1: The symbolic representations of the single-qubit quantum gates.

Gate	Symbol
NOT	$-\oplus-$
Y	$-\boxed{Y}-$
Z	$-\boxed{Z}-$
Identity (I)	$-\boxed{I}-$
Hadamard (H)	$-\boxed{H}-$
T	$-\boxed{T}-$
S	$-\boxed{S}-$
$R_x(\theta)$	$-\boxed{R_x(\theta)}-$
$R_y(\theta)$	$-\boxed{R_y(\theta)}-$
$R_z(\theta)$	$-\boxed{R_z(\theta)}-$
Phase ($P(\phi)$)	$-\boxed{P(\phi)}-$

$$\mathrm{CZ} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}. \tag{3.27}$$

Moreover, in general, a controlled-U gate (CU) is defined by unitary matrix

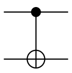
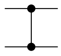
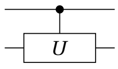
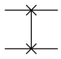
$$\mathrm{CU} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{pmatrix}, \qquad U = \begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix}. \tag{3.28}$$

The SWAP gate swaps the state of two qubits. In other words, it maps $|01\rangle \to |10\rangle$, $|10\rangle \to |01\rangle$, and leaves $|00\rangle$ and $|11\rangle$ unchanged. This gate is represented by unitary matrix

$$\mathrm{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.29}$$

The quantum circuit symbol representations of these gates are depicted in Table 3.2.

Table 3.2: The symbolic representations of the two-qubit quantum gates.

Gate	Symbol
CNOT	
CZ	
CU	
SWAP	

3.5.3 Multi-qubit gates

These gates act on n qubits (more than two), and are represented in terms of $2^n \times 2^n$ unitary matrices. Note that it is needed to use decomposition methods to convert these gate into a sequence of elementary quantum gates that are runnable on real quantum computers. I introduce some of multi-qubit gates that I consider in my work as follows.

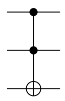
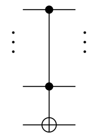
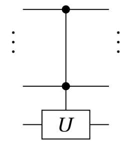
Toffoli gates. One of the popular three-qubit gates is Toffoli gate that corresponds to a two-controlled NOT gate or CCNOT. The Toffoli gate performs a NOT operation on the third qubit (the target qubit) if and only if the first two qubits (the control qubits) are both in the state $|1\rangle$. Otherwise, the third qubit is left unchanged. The Toffoli gate can be represented by a 8×8 unitary matrix as

$$\text{Toffoli} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3.30}$$

Multi-controlled NOT gates. These gates are a generalization of Toffoli gates that allow for multiple control qubits. The number of control-qubits is more than two and a NOT gate applies to the target-qubit. The target-qubit toggles if and only if all control-qubits are in the state $|1\rangle$. I make use of the method proposed in [89] to decompose multi-controlled NOT gates to a sequence of Toffoli gates. A multi-controlled NOT gate on n qubits can be represented by a $2^n \times 2^n$ unitary matrix.

Multi-controlled U gates. In general, I consider multi-controlled U gates and U can be any single-qubit gate. When all control-qubits are in the state $|1\rangle$, the U gate acts, otherwise nothing happens. A multi-controlled U gate on n qubits can be demonstrated by a unitary matrix with the size of $2^n \times 2^n$.

Table 3.3: The symbolic representations of the multi-qubit gates.

Gate	Symbol
Toffoli	
MCNOT	
MCU	

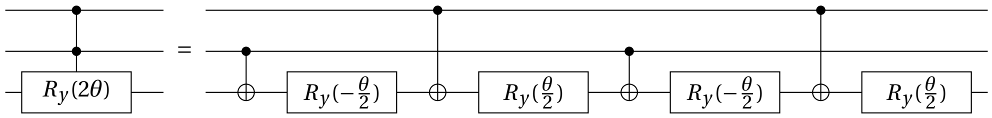


Figure 3.3: Decomposing a 2-controlled $R_y(2\theta)$ gate into elementary quantum gates.

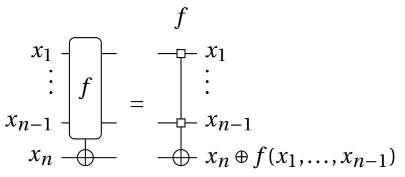
The quantum circuit symbol representations of Toffoli, multi-controlled NOT (MCNOT), and multi-controlled U (MCU) gates are depicted in Table 3.3.

Example 3.5.1. *In this thesis, I make use of 1-controlled R_y gates and 2-controlled R_y gates. A 1-controlled $R_y(2\theta)$ is easily seen to be created by 2 $R_y(\pm\theta)$ and 2 CNOTs. It is also easy to see that a 2-controlled $R_y(2\theta)$ is created by 4 $R_y(\pm\frac{\theta}{2})$ and 4 CNOTs that is depicted in Fig. 3.3.*

Multi-controlled single-target gates. A single-target gate with control function $f(x_1,\ldots,x_{n-1})$ and target qubit x_n is an abstract quantum operation acting on n qubits that maps

$$U_f : |x_1 \ldots x_{n-1}\rangle|x_n\rangle \mapsto |x_1 \ldots x_{n-1}\rangle|x_n \oplus f(x_1,\ldots,x_{n-1}\rangle. \tag{3.31}$$

In other words, it inverts the value of the target qubit x_n, if and only if the control function evaluates to true for the values of the control qubits $x_1,\ldots,x_{n-1}$. Well-known instances of single-target gates are the X gate for which $n = 1$ and $f = 1$, the CNOT gate for which $n = 2$ and $f = x_1$, or the Toffoli gate for which $n = 3$ and $f = x_1x_2$. Several pictorial representations for a multi-controlled single-target gate (single-target gate) are used in the literature. In the remainder of the thesis, I use one of the following two:



(3.32)

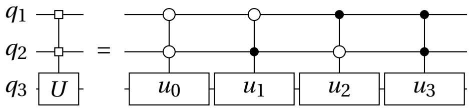


Figure 3.4: A uniformly-controlled single-qubit gate with two controls.

Single-target gates describe complex operations and cannot generally be implemented natively on quantum computer. However, they provide a convenient intermediate representations when mapping complex functionality, such as permutations, into quantum gates.

Uniformly-controlled single-qubit gates. In this thesis, I make use of a family of unitary matrices called *uniformly-controlled single-qubit gates* [40, 42]. These unitary matrices are $2^n \times 2^n$ block diagonal matrices of the form

$$U_i = u_0 \oplus \cdots \oplus u_{k-1} = \begin{pmatrix} u_0 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & u_{k-1} \end{pmatrix}, \tag{3.33}$$

with one target qubit q_n and $n-1$ control qubits $q_1, ..., q_{n-1}$. The matrices can be decomposed into the direct sum of $k = 2^{n-1}$ unitary 2×2 matrices $u_0, \ldots, u_{k-1}$.

An example of a uniformly-controlled single-qubit gate on $n = 3$ qubits with two controls is shown in Fig. 3.4. The figure also shows the visual representation of the uniformly-controlled single-qubit gate on the left-hand side.

Decomposing an n-qubit uniformly-controlled rotation gate requires an exponential number of elementary gates ($2^n - 1$ rotation gates and $2^n - 2$ CNOTs) [40].

3.6 Universal quantum gates sets

The ability to implement any possible unitary, means achieving universality in the sense of standard digital computers. A universal quantum gate set is a set of quantum gates that can be used to approximate any quantum gate to an arbitrary degree of accuracy. There are several different universal gate sets. For example, single-qubit gates and CNOT gate together are universal. Toffoli gates together with single-qubit gates can also form a universal gate set. Another important universal gate set library is Clifford+T gate library which is considered typically for fault-tolerant quantum computing. This library consists of the CNOT gate, the Hadamard (H) gate, as well as the T gate, and its inverse $\mathrm{T}^\dagger$. Any unitary operation can be approximated to arbitrary accuracy using these gates which for convenient in this thesis, I call them *elementary quantum gates*. Hence, when there is a complex gate, decomposition methods can be used to convert it to elementary quantum gates that are available physically in real quantum computers.

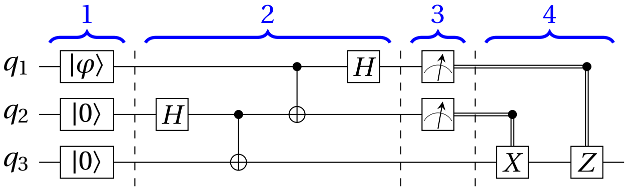


Figure 3.5: The quantum circuit parts related to the quantum program in the Exp. 3.7.1.

3.7 Quantum circuits

A *quantum circuit* is a structural description of a quantum program. It is an ordered sequence of quantum gates on quantum data, such as qubits, and concurrent real-time classical computation. Quantum circuits follow a sequence from left to right that correspond to the evolution of time.

Example 3.7.1. *Fig. 3.5 shows a quantum circuit that implements the quantum teleportation algorithm. As it is shown, this circuit is consisted of 4 parts as follows.*

1. *Initialization. The quantum circuit starts with a well-defined quantum state. Here q_2 and q_3 are initialized to $|0\rangle$ state, q_1 is initialized by a quantum state $|\varphi\rangle$ which can be any combination of single-qubit gates. The initialization part is an important part of a quantum circuit, and any quantum algorithm requires its corresponding initial quantum state in the beginning. Note that, this part can be provided as a quantum circuit to bring qubits in the desired initial quantum state.*

2. *Quantum gates. In this stage of the circuit, a sequence of quantum gates are applied to manipulate qubits as required by the teleportation algorithm.*

3. *Measurements. Here, two qubits q_1 and q_2 are measured which are useful for classical computer interpreters.*

4. *Classically conditioned quantum gates. In this stage, the results of the measurement in the previous stage are used, concurrently in real-time, to control two quantum gates X and Z that act on q_3.*

3.8 Quantum algorithms

3.8.1 Quantum oracles

A quantum oracle function is a black box quantum circuit that implements a classical function $f(x)$ in such a way that it can be queried by a quantum algorithm in superposition. Quantum oracle functions are an important component of many quantum algorithms, including Shor's algorithm for factoring large numbers and Grover's algorithm for searching unstructured databases.

The mathematical definition of a quantum oracle can be given as follows:

Definition 3.8.1. *Let f be a classical function that maps an n-bit binary input and produces an m-bit binary output, i.e.,*

$$f : \{0,1\}^n \to \{0,1\}^m. \tag{3.34}$$

Then, the quantum oracle O associated with f is a unitary operator that acts on $n+m$ qubits and is defined as:

$$O(|x\rangle \otimes |y\rangle) = |x\rangle \otimes |y \oplus f(x)\rangle. \tag{3.35}$$

where $|x\rangle$ is an n-qubit input register, $\left|y\right\rangle$ is an m-qubit output register, $\oplus$ denotes bitwise addition modulo 2, and $f(x)$ is the value of the classical function f on input $x = (x_1, \ldots, x_n)$. The input qubit states can be labeled as $|x\rangle = |x_1\rangle \otimes \ldots \otimes |x_n\rangle$.

In other words, the quantum oracle O maps the input state $|x\rangle$ to the output state $|x\rangle \left|y \oplus f(x)\right\rangle$, where the output register is modified based on the value of $f(x)$.

Phase oracles. These oracles encode $f(x)$ into an oracle O by applying a phase depending on the input x. It is defined as

$$O|x\rangle = (-1)^{f(x)}|x\rangle. \tag{3.36}$$

If a phase oracle acts on a register initially in a computational basis state $|x\rangle$, then the phase is a global phase and is not directly observable. However, if the phase oracle is applied to a superposition state or used as a controlled operation, it can become a powerful resource in quantum algorithms such as quantum search and quantum simulation.

3.8.2 Quantum Fourier transform

The Fourier transform is a widely used mathematical tool in classical computing, applied in areas such as signal processing, data compression, and complexity theory. The quantum Fourier transform (QFT) is the quantum implementation of the discrete Fourier transform that applies to the amplitudes of a quantum wave function. It is a fundamental component of many quantum algorithms, including Shor's factoring algorithm and quantum phase estimation.

The QFT acts on a quantum state $|X\rangle = \sum_{j=0}^{N-1} x_j |j\rangle$, $N = 2^n$, and maps it to the state $|Y\rangle = \sum_{k=0}^{N-1} y_k |k\rangle$, where the amplitudes y_k are given by:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j w_N^{jk}, \tag{3.37}$$

where $w_N^{jk} = e^{2\pi i \frac{jk}{N}}$. Note that only the amplitudes of the quantum state are affected by this transformation.

The QFT is a transformation that acts on a quantum state in the computational (Z) basis and maps it to a state in the Fourier basis. The H gate is the single-qubit QFT, and it transforms between the Z-basis states $|0\rangle$ and $|1\rangle$ to the X-basis states $|+\rangle$ and $|-\rangle$. Similarly, all multi-qubit states in the computational basis have corresponding states in the Fourier basis. The QFT is simply the function that transforms between these bases.

3.8.3 Deutsch-Joza algorithm

The Deutsch-Jozsa algorithm [90] is the first example of a quantum algorithm that outperforms the best classical algorithm.

The problem is to determine whether a given Boolean function $f(x_1, x_2, \ldots, x_n)$, which takes as input a binary string of length n and returns either 0 or 1, is constant or balanced. A constant function returns the same value for all inputs, while a balanced function returns 0 for half of the inputs and 1 for the other half.

The Deutsch-Jozsa algorithm solves this problem using a quantum circuit that can determine the nature of the function using just one query to the function. The Deutsch-Jozsa algorithm is significant because it provides an exponential speedup over the best classical algorithm for this problem, which requires $2^{n-1}+1$ queries to the function in the worst case.

Assume the function f is implemented as a quantum oracle, which maps the state $|x\rangle|y\rangle$ to $|x\rangle|y \oplus f(x)\rangle$. The generic circuit for the Deutsch-Jozsa algorithm is illustrated on Fig. 3.6. This algorithm acts as follows.

1. Initialize two quantum registers $|x\rangle$ and $|y\rangle$ to $|0\rangle^{\otimes n}$, and $|1\rangle$, respectively:

$$|\varphi_1\rangle = |0\rangle^{\otimes n}|1\rangle \tag{3.38}$$

2. Apply a H gate to each qubit:

$$|\varphi_2\rangle = \frac{1}{\sqrt{2^{n+1}}}\sum_{x=1}^{2^n} |x\rangle(|0\rangle - |1\rangle) \tag{3.39}$$

3. Apply the quantum oracle O_f, which maps $|x\rangle|y\rangle$ to $|x\rangle|y \oplus f(x)\rangle$:

$$\begin{aligned} |\varphi_3\rangle &= \frac{1}{\sqrt{2^{n+1}}}\sum_{x=1}^{2^n} |x\rangle(|f(x)\rangle - |1 \oplus f(x)\rangle) \\ &= \frac{1}{\sqrt{2^{n+1}}}\sum_{x=1}^{2^n} (-1)^{f(x)}|x\rangle(|0\rangle - |1\rangle) \end{aligned} \tag{3.40}$$

4. At this point the second single-qubit register may be ignored. Apply a Hadamard gate to each qubit in the first register:

$$\begin{aligned} |\varphi_4\rangle &= \frac{1}{2^n}\sum_{x=1}^{2^n} (-1)^{f(x)} \left[\sum_{y=1}^{2^n} (-1)^{x.y}|y\rangle\right] \\ &= \frac{1}{2^n}\sum_{y=1}^{2^n} \left[\sum_{x=1}^{2^n} (-1)^{f(x)}(-1)^{x.y}\right]|y\rangle, \end{aligned} \tag{3.41}$$

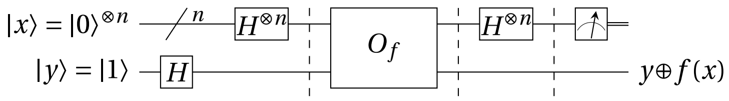


Figure 3.6: The quantum circuit of the Deutsch-Jozsa algorithm.

where $x.y = x_1y_1 \oplus ... \oplus x_ny_n$ is the sum of the bitwise product.

5. Measure the first register. Notice that the probability of measuring evaluates to 1 if $f(x)$ is constant, and 0 if $f(x)$ is balanced.

3.8.4 Grover's algorithm

Grover's algorithm is a quantum algorithm that can be used to solve unstructured search problems, such as searching a large database to find a specific item with a unique property. The algorithm can speed up the search process quadratically, which is much faster than classical algorithms. The algorithm uses a technique called amplitude amplification to amplify the amplitude of the solution(s) in the quantum state.

Given a large list of N items. Among these items, there is one item with a unique property and I wish to locate, which I call it the winner w. To find the w using classical computation, one would have to check on average $\frac{N}{2}$ of items, and in the worst case, all N of them. On a quantum computer, however, the marked item can be find in roughly $\sqrt{N}$ steps with Grover's algorithm. Considering the winner w, the corresponding Boolean function f to mark w can be defined as

$$f(x) = \begin{cases} 1, & \text{if } x \neq w \\ -1, & \text{if } x = w \end{cases} \tag{3.42}$$

The algorithm works by first encoding the items in the database into quantum states. Each item is represented as a basis state in a quantum register. The winner item w is marked with a phase shift of -1, while the other items are marked with a phase shift of 1. The quantum oracle will be a diagonal matrix, where the entry that correspond to the marked item will have a negative phase. This is achieved by applying the oracle gate that flips the phase of the winner item and can be described as

$$O_f|x\rangle = (-1)^{f(x)}|x\rangle, \tag{3.43}$$

and the oracle's matrix will be a diagonal matrix of the form

$$O_f = \begin{bmatrix} (-1)^{f(0)} & 0 & \dots & 0 \\ 0 & (-1)^{f(1)} & \dots & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \dots & (-1)^{f(2^n-1)} \end{bmatrix}. \tag{3.44}$$

One can extend the problem for multiple solutions (winners). Suppose there are $N = 2^n$ eligible items for the search task and they are indexed by assigning each item an integer from 0 to $N-1$. Further, suppose that there are M different solutions, meaning that there are M

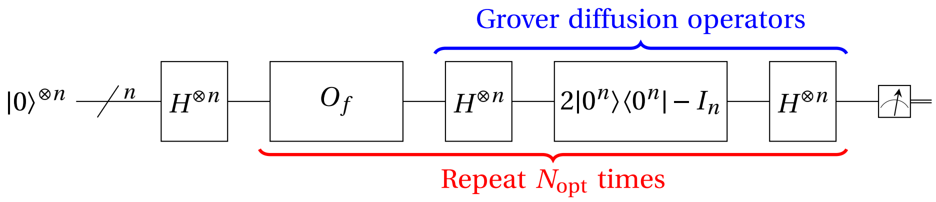

Grover diffusion operators
$|0\rangle^{\otimes n}$
n
$H^{\otimes n}$
O_f
$H^{\otimes n}$
$2|0^n\rangle\langle 0^n| - I_n$
$H^{\otimes n}$
Repeat N_{opt} times

Figure 3.7: The quantum circuit of Grover's algorithm.

inputs for which $f(x) = 1$. The generic circuit for the Grover's algorithm is illustrated on Fig. 3.7, and the steps of the algorithm are as follows:

1. Start with a register of n-qubit initialized in the state $|0\rangle$.
2. Prepare the register into a uniform superposition by applying H gates to each qubit of the register, which transform the state as
$$|\varphi_1\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle. \tag{3.45}$$
3. **Oracle.** Apply the oracle O_f that marks the solutions in the database.
4. **Diffusion.** The Grover diffusion operator is used to amplify the amplitude of the solution states and decrease the amplitude of the non-solution states. This part is constructed as follows.
   - Apply H gates to all qubits.
   - Apply a conditional phase shift of -1 to every computational basis state except $|0\rangle$. This can be represented by the unitary operation
   $$2|0^n\rangle\langle 0^n| - I_n. \tag{3.46}$$
   - Apply H gates again to all qubits.
5. Repeat two steps 3 and 4 by N_{opt} times. N_{opt} is the optimal number of iterations that maximizes the likelihood of obtaining the correct item by measuring the register which is computed by
$$N_{\text{opt}} = \lfloor \frac{\pi}{4}\sqrt{\frac{N}{M}} - \frac{1}{2} \rfloor. \tag{3.47}$$
6. Measure qubits to obtain the index of an item that is a solution with very high probability.

3.8.5 Shor's algorithm

Shor's algorithm is a quantum algorithm for factoring integers efficiently in polynomial time. The problem of factoring large integers is important in cryptography, as many encryption schemes rely on the difficulty of factoring large numbers. Since the best-known classical algorithm requires super-polynomial time to factor the product of two primes, the widely used cryptosystem, such as RSA, being impossible for large enough integers.

The key insight behind Shor's algorithm is that factoring an integer N can be reduced to the problem of finding the period of a certain function, namely $f(x)$ as

$$f(x) = a^x \bmod N, \tag{3.48}$$

where a and N are positive integers, a is less than N, and they have no common factors. The period, or order r, is the smallest (non-zero) integer such that

$$a^r \bmod N = 1. \tag{3.49}$$

Shor's algorithm uses the quantum phase estimation algorithm to find the eigenvalues and eigenvectors of the unitary operator U, defined as

$$U|y\rangle \equiv |ay \bmod N\rangle. \tag{3.50}$$

The eigenstates of U are superpositions of the states in the cycle of length r, where $a^r \bmod N = 1$.

One such eigenstate is the superposition of the states $|a^k \bmod N\rangle$ for $k = 0, 1, \ldots, r-1$, which can written as

$$|u_0\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |a^k \bmod N\rangle. \tag{3.51}$$

This eigenstate has an eigenvalue of 1, which is not very useful. A more interesting eigenstate can be constructed by assigning a phase that is proportional to the index of each computational basis state. Specifically, consider the case where the phase of the k-th state is proportional to k

$$|u_1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi ik}{r}} |a^k \bmod N\rangle. \tag{3.52}$$

Applying the operator U to $|u_1\rangle$ yields an eigenvalue of $e^{\frac{2\pi i}{r}}$, which is particularly interesting because it contains r. In fact, r must be included to ensure that the phase differences between the r computational basis states are equal. However, this is not the only eigenstate with this property. To generalize further, an integer s can be multiplied to the phase difference, resulting in the following eigenstate and eigenvalue as

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi isk}{r}} |a^k \bmod N\rangle, \tag{3.53}$$

$$U|u_s\rangle = e^{\frac{2\pi is}{r}} |u_s\rangle. \tag{3.54}$$

Thus, there is a unique eigenstate for each integer value of s between 0 and $r-1$.

Remarkably, if all these eigenstates are summed, the different phases cancel out all computational basis states except $|1\rangle$

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle. \tag{3.55}$$

Since $|1\rangle$ is a superposition of these eigenstates, performing quantum phase estimation on U using $|1\rangle$ will result in a random phase

$$\varphi = \frac{s}{r}, \tag{3.56}$$

where s is a random integer between 0 and $r-1$. By applying the continued fractions algorithm to φ, the value of r can be determined, finally.

3.9 Summary

This chapter introduces the basic concepts of quantum computing, including Dirac notation, qubits, superposition, and entanglement. The Bloch sphere representation is described as a way to visualize the state of a qubit. Three different classes of quantum gates are presented: single-qubit gates, two-qubit gates, and multi-qubit gates. A universal quantum library consisting of Clifford+T gates is described, and its importance in fault-tolerant quantum computing is emphasized. Several important quantum algorithms are also presented in detail, including Grover's algorithm for unstructured search, Shor's algorithm for factoring large numbers, the quantum Fourier transform, and the Deutsch-Joza algorithm for determining whether a function is constant or balanced.

Overall, this chapter provides the necessary background knowledge for understanding quantum computation and the algorithms needed in the remainder of this thesis.

Quantum Circuit Synthesis Part II

4 Compiling Permutations

In this chapter, I consider the compilation of quantum state permutations into quantum gates for physical quantum computers. Quantum compilation is the task of translating a high-level quantum algorithm into a low-level quantum circuit, which is technology-dependent, i.e., it is described in terms of supported quantum operations and respects all architectural constraints. It is not uncommon that several phases of quantum compilation take place when translating a high-level quantum algorithm into a low-level quantum circuit [91].

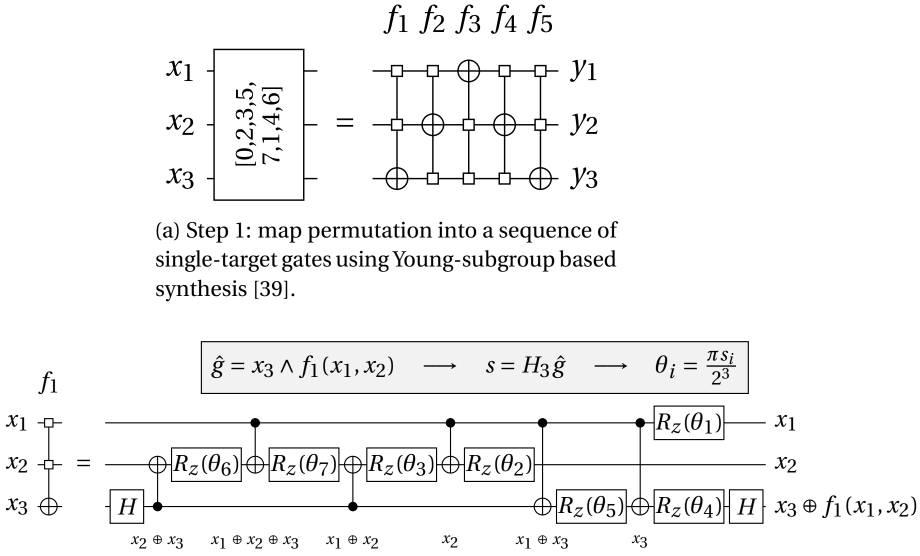
I propose an automatic technology-dependent compilation technique to translate quantum operations that permute the amplitudes in quantum states. Many quantum algorithms make use of such permutations, in particular as a way to implement combinatorial operations [92]. My compilation algorithm targets quantum architectures whose gate set supports rotation gates with arbitrary angles, such as the 8-qubit and 19-qubit superconducting transmon computers from Rigetti.

4.1 Proposed compilation algorithm

Fig. 4.1 provides a birds-eye overview of my proposed algorithm. The algorithm takes as input a permutation over 2^n elements, the gate library, and coupling constraints of the target quantum computer. It returns a quantum circuit composed of gates from the gate library, which respects the coupling constraints. The algorithm essentially contains the following three steps:

1. Map the input permutation into a reversible circuit composed of $2n-1$ single-target gates.
2. Map each single-target gate into a circuit over the gate set $\{\text{CNOT}, R_z(\theta), H\}$.
3. Map the resulting quantum circuit into a circuit composed of gate library gates, which respects the coupling constraints.

For the first step, I employ the decomposition-based reversible synthesis algorithm using



(a) Step 1: map permutation into a sequence of single-target gates using Young-subgroup based synthesis [39].

(b) Step 2: each single-target gate is translated into a regular circuit structure composed of H gates, CNOT gates, and R_z gates. The angles for the R_z gates can be obtained from the control function of the single-target gate.

Figure 4.1: Birds-eye overview of the proposed compilation algorithm.

Young-subgroups presented in [39]. Fig. 4.1.a illustrates this step for the permutation

$$\text{Perm} = [0, 2, 3, 5, 7, 1, 4, 6], \tag{4.1}$$

which can be realized using a reversible circuit consisting of five single-target gates with control functions $f_1, \ldots, f_5$.

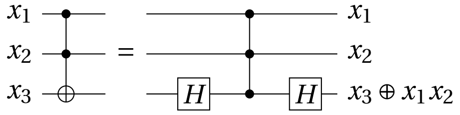
The second step encompasses the main contribution of this chapter. Each single-target gate is decomposed into a circuit structure composed of H gates, CNOT gates, and R_z gates. The step is illustrated for the first single-target gate with the control function f_1 in Fig. 4.1.b. Hadamard gates are inserted at the beginning and at the end of the circuit on the target qubit of the single-target gate. CNOT gates are used to create all linear combinations of the inputs. In Fig. 4.1.b, these linear combinations are written under each CNOT gate. To each of these linear combinations one R_z gate is applied, whose rotation angle corresponds to the spectral coefficients of the function $x_3 \wedge f_1(x_1, x_2)$. Details on computing spectral coefficients are presented in Section 2.3. For example, the first CNOT gate creates the linear combination $x_2 \oplus x_3$, which corresponds to the spectral coefficient s_6. The rotation angle is $\theta_6 = \frac{\pi s_6}{2^3}$. Note that $R_z(0)$ gates may be omitted which in turn can cause further reduction of CNOT gates. I make use of the GRAYSYNTH algorithm [93] to find a network with possibly few CNOT gates for generating required linear combinations.

By performing the second step for each single-target gate, one obtains a quantum circuit that can be passed as input to a quantum compiler. However, the compiler of the quantum computer still needs to apply changes to accommodate for the supported gate library and coupling constraints. For example, in case of the Rigetti 8-qubit computer, the H and CNOT gates need to be translated into R_x, R_z and CZ gates; also, there are no three qubits that permit pairwise interaction and therefore some SWAP gates need to be inserted. I extended the existing GRAYSYNTH algorithm by heuristics to further minimize the number of CNOTs, thereby reducing the possibility of violating coupling constraints. For example, special strategies can be used when the spectrum has no zero coefficients and therefore CNOTs for all linear combinations need to be generated.

For the third step, I make use of the quantum gate compilers shipped with the software development kits from Rigetti and IBM.

4.2 Compiling single-target gates

In this section, I describe in detail the second step of the algorithm that was presented in the previous section. Before delving into the details on how to compile single-target gates, I briefly review a well-known identity for Toffoli gates, i.e., a single target gate acting on three qubits with control function $f = x_1x_2$. For compiling Toffoli gates into quantum gates, it is often first mapped into a two-controlled Z gate using Hadamard gates:



(4.2)

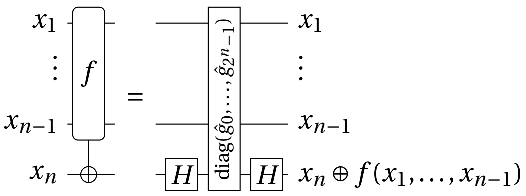
The unitary operation of the two-controlled Z gate is $\mathrm{diag}(1,1,1,1,1,1,1,-1)$.

The following theorem generalizes this decomposition for single-target gates of arbitrary size.

Theorem 4.2.1. *Let U_f be the unitary matrix realized by a single-target gate acting on x_n with control function $f(x_1,\dots,x_{n-1})$. Then*

$$U_f = (I_{2^{n-1}} \otimes H)\cdot \mathrm{diag}(\hat{g}_0,\dots,\hat{g}_{2^n-1})\cdot (I_{2^{n-1}} \otimes H), \tag{4.3}$$

where $\hat{g}_i$s are the truth table entries of the function $g = x_n \wedge f$ in $\{-1,1\}$ coding. The following circuit illustrates the decomposition:


x_1
$\vdots$
x_{n-1}
x_n
f
=
H
$\mathrm{diag}(\hat{g}_0,\dots,\hat{g}_{2^n-1})$
H
x_1
$\vdots$
x_{n-1}
$x_n \oplus f(x_1,\dots,x_{n-1})$

(4.4)

Proof. I evaluate the effect of applying the right-hand side of (4.3) to the quantum state $|x\rangle|x_n\rangle$, where $|x\rangle = |x_1 \dots x_{n-1}\rangle$. Applying $(I_{2^{n-1}} \otimes H)$ to $|x\rangle|x_n\rangle$ yields

$$|\varphi_1\rangle = \frac{1}{\sqrt{2}}|x\rangle(|0\rangle + (-1)^{x_n}|1\rangle).$$

Now note that $\mathrm{diag}(g_0,\dots,g_{2^n-1})$ maps a state $|x\rangle|x_n\rangle$ to $(-1)^{x_n\wedge f(x)}|x\rangle|x_n\rangle$. When making a case distinction on x_n, one can see that the operation maps $|x\rangle|0\rangle$ to $|x\rangle|0\rangle$, and $|x\rangle|1\rangle$ to $(-1)^{f(x)}|x\rangle|1\rangle$. Therefore, applying the diagonal matrix to $|\varphi_1\rangle$ yields

$$\begin{aligned} |\varphi_2\rangle &= \frac{1}{\sqrt{2}}|x\rangle(|0\rangle + (-1)^{f(x)}(-1)^{x_n}|1\rangle) \\ &= \frac{1}{\sqrt{2}}|x\rangle(|0\rangle + (-1)^{f(x)\oplus x_n}|1\rangle). \end{aligned}$$

Then, finally

$$(I_{2^{n-1}} \otimes H)|\varphi_2\rangle = |x\rangle|x_n \oplus f(x)\rangle = U_f|x\rangle|x_n\rangle.$$

□

Welch et al. have shown in [94] that the matrix $\operatorname{diag}(\hat{g}_0, \ldots, \hat{g}_{2^n-1})$ is equivalent to the unitary operation that maps $|x\rangle$ to

$$e^{\mathrm{i}\pi s(x)/2^n}|x\rangle, \tag{4.5}$$

where $s(x) = \sum_{y\in\mathbb{B}^n} s_y|y\rangle\langle x|$.

Schuch and Siewert have shown in [95] that a unitary mapping as in (4.5) can be implemented by a quantum circuit on n qubits that uses only CNOT and R_z gates. The CNOT gates are used to generate the linear combinations $|y\rangle\langle x|$ on some qubit to which then the phase gate $R_z\left(\frac{\pi s_y}{2^n}\right)$ is applied. Note that only those linear combinations need to be generated for which $s_y \neq 0$. In [93], Amy et al. presented an algorithm called GRAYSYNTH that finds a circuit which minimizes the number of CNOT gates.

4.3 Rewiring optimizations

The algorithm to compile single-target gates as described in the previous section does not take into account the coupling constraints of the quantum computer. In this section, I describe three modifications to the algorithm that take the coupling constraints into account.

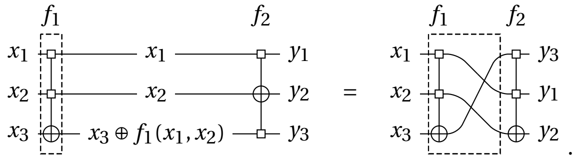
First, in order to find a good CNOT network to create linear combinations for the non-zero spectral coefficients, GRAYSYNTH uses a heuristic to minimize the number of CNOT gates. Often the algorithm has multiple choices and I use the coupling constraints as a tie breaker in such cases.

Second, in the case of spectra with no zero coefficients, all linear combinations of qubits are required. In this case, I employ dedicated algorithms to generate CNOT sequences for all linear combinations instead of using GRAYSYNTH. Since all spectral coefficients are not zero, the sequence of CNOT gates depends only on the number of variables in the function. As a consequence, I precompute circuit structures that minimize the number of CNOTs as well as violations of coupling constraints of the quantum computer.

Third, I allow rewirings after each single-target gate. Formally, instead of implementing U_f as in (3.31), I find a circuit for $U_\pi U_f$, where

$$U_\pi : |x_1 \ldots x_n\rangle \mapsto |x_{\pi(1)} \ldots x_{\pi(n)}\rangle \tag{4.6}$$

for some permutations $\pi \in S_n$ (i.e., over n elements). The unitary transformation U_π can be realized without any gates, simply by rewiring the qubits. The following example illustrates this:



(4.7)

The highlighted gate on the left-hand side implements U_{f_1} while the highlighted gate on the right-hand side implements $U_{[3,1,2]}U_{f_1}$. The implementation of the latter may require fewer gates.

4.4 Experimental results

I implemented the proposed algorithm and the state-of-the-art algorithm for comparison in C++.

4.4.1 Benchmarks

I use the benchmark families `TOF`(n), `PRIME`(n), and `HWB`(n) to evaluate the proposed approach, where n indicates that the benchmark describes a permutation over 2^n elements. The benchmark `TOF`(n) describes a multi-controlled Toffoli gate with $n-1$ control qubits acting on the least-significant qubit; it represents the single transposition $(2^{n-2}, 2^{n-1})$. Such gates play, e.g., an important role in the Grover search quantum algorithm [96]. The benchmark `PRIME`(n) represents the permutation that maps 0 to 0, and then maps the successive numbers first to all primes and then to all non-primes smaller than 2^n in increasing order. For example, `PRIME(3)` $= [0, 2, 3, 5, 7, 1, 4, 6]$, the permutation used in the example in Fig. 4.1. Finally, the benchmark `HWB`(n) maps x to $x \lll_r \nu x$, i.e., x is left-shift-rotated by the number of 1s in x. For example, `HWB(3)` $= [0, 2, 4, 5, 1, 6, 3, 7]$.

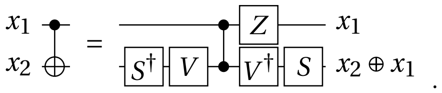
4.4.2 Quantum gate libraries and quantum architectures

Rigetti quantum computers

The 8-qubit QPU (called Agave) and 19-qubit QPU (called Acorn) from Rigetti natively supports four X-rotations $\{R_x(\frac{k\pi}{2}) \mid k \in \mathbb{Z}\} = \{I, V, X, V^\dagger\}$, arbitrary Z-rotations $R_z(\theta)$ for any $\theta \in \mathbb{R}$ and CZ gates. Other gates can be represented in terms of this library. The Hadamard gate H can be expressed in terms of three rotations:

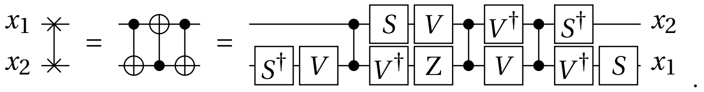
$$-\boxed{H}- \;=\; -\boxed{V}-\boxed{S}-\boxed{V}- \;=\; -\boxed{S}-\boxed{V}-\boxed{S}- \;=\; -\boxed{V^\dagger}-\boxed{S^\dagger}-\boxed{V^\dagger}- \;=\; -\boxed{S^\dagger}-\boxed{V^\dagger}-\boxed{S^\dagger}- \,. \tag{4.8}$$

The CNOT gate can be expressed in terms of a CZ gate and rotations, using 6 gates and a depth of 5:



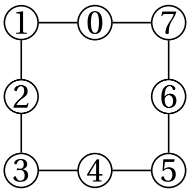
. (4.9)

Finally, the SWAP gate can be expressed in terms of three CZ gates and rotations, using 14 gates and a depth of 10:



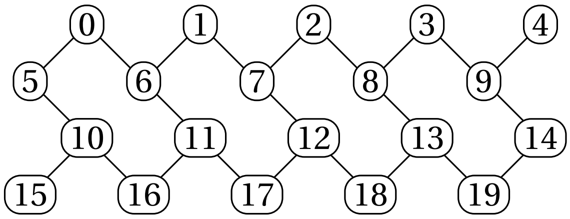
. (4.10)

The topology of the 8-qubit quantum computer is the undirected 8-cycle



, (4.11)

while the topology of the 19-qubit computer is according to the following undirected graph:



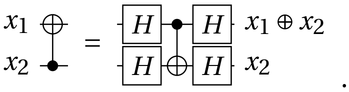
. (4.12)

IBM quantum computers

The IBM quantum computers natively support the U gate, $U(\theta,\phi,\lambda) = R_z(\phi)R_y(\theta)R_z(\lambda)$, which is parametrized over 3 continuous variables, and the CNOT gate. As the algorithm described in this chapter, maps permutations to an intermediate representation containing of CNOT, Hadamard, and arbitrary Z rotations, it is helpful to point out the following identities:

$$H = U(\tfrac{\pi}{2},0,\pi) \quad \text{and} \quad R_z(\theta) = U(\theta,0,0). \tag{4.13}$$

The coupling constraints in an IBM architecture also imply a direction of a CNOT gate, i.e., for two adjacent qubits the target of a CNOT gate can only be applied to one of the qubits. The following circuit identity can be used to turn the direction of a CNOT gate:



. (4.14)

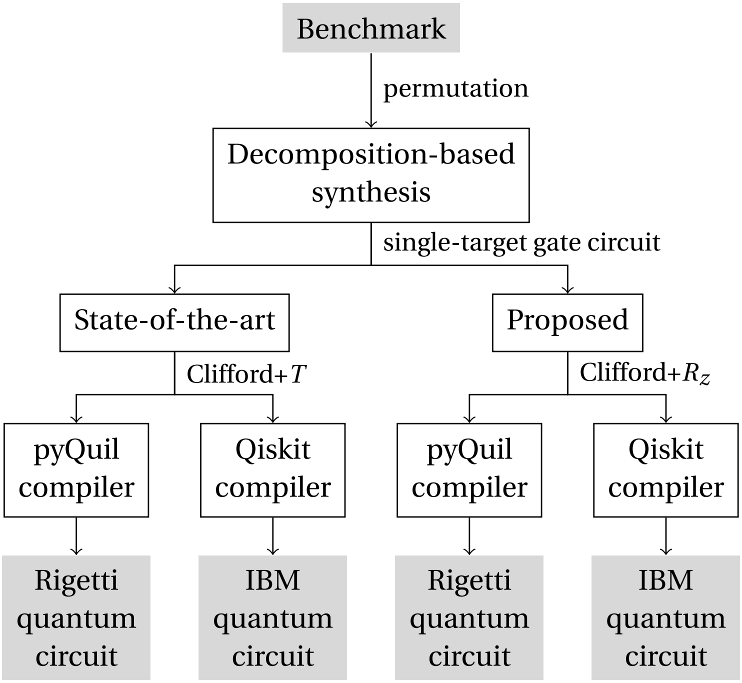
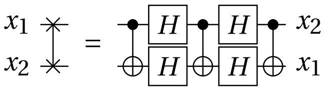


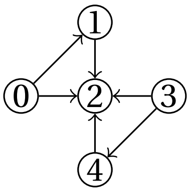
Figure 4.2: Compilation flows for experimental results.

The same identity can be used to implement a SWAP operation:

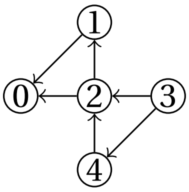


. (4.15)

In the experiments, I evaluate my algorithm on the three available cloud-based IBM quantum computers ibmqx2 (also called IBM 5Q Yorktown), ibmqx4 (also called IBM 5Q Tenerife), each of which have 5 qubits, and ibmqx5 (also called IBM 16Q Rueschlikon), which has 16 qubits. The coupling constraints for ibmqx2 and ibmqx4 are defined to be the two directed graphs

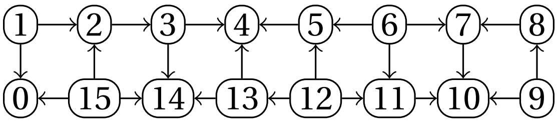


and



, (4.16)

respectively. For ibmqx5, the coupling constraints are as follows:



. (4.17)

4.4.3 Methodology

Fig. 4.2 shows the flow for obtaining the results. Input is a benchmark generated from the benchmark families described in the Section 4.4.1. The permutation is decomposed into a sequence of single-target gates using the algorithm described in [39]. Then each single-target gate is mapped into a Clifford+T circuit using a state-of-the-art compilation flow or into a Clifford+R_z circuit using my proposed flow. In the state-of-the-art synthesis flow, I first use ESOP-based synthesis to map each single-target gate into a sequence of multi-controlled mixed-polarity Toffoli gates, which are single-target gates in which the control function is a product term (conjunction of literals). Multi-controlled Toffoli gates with more than four controls are decomposed into Toffoli gates with at most four controls using the decomposition described in [97]; this may introduce one additional helper qubit called *ancilla*. Finally, each remaining multi-controlled Toffoli gate is decomposed into a Clifford+T circuit as described in [98].

The circuits generated by the state-of-the-art and the proposed compilation approach are then compiled into architecture aware circuits for both Rigetti and IBM quantum computers using the compilers provided in their software development kits. For Rigetti, I use architecture configurations for Agave 8Q and Acorn 19Q, and for IBM I use architecture configurations for both 5-qubit and the 16-qubit quantum computers.

Tables 4.1, 4.2, 4.3, and 4.4 show the experimental results after compilation for the Rigetti and IBM quantum computers, respectively. The table shows the input permutation and the number of variables or qubits. For each quantum computer, it shows the number of gates and the gate depth after compilation using the state-of-the-art and the proposed approach. Note that gate count and volume for the Rigetti computers (R gates and R depth) and the IBM computers (I gates and I depth) are not directly comparable due to differences in the underlying gate sets and device technologies. In the columns for the proposed approach also the improvement in percentage is listed. The runtime in all cases is a few seconds and negligible. For the `TOF(`n`)` benchmark, the circuit includes only one multi-controlled Toffoli. As SOTA is optimized in compiling Toffoli gates, the results for SOTA are better. However, for other benchmarks that include several multi-controlled gates, my method works better. As can be seen, my proposed approach is particularly powerful when the number of variables is larger than 3, because only then the proposed algorithm can exploit the smaller rotation angles in the R_z gates. Moreover, for a small number of variables, SOTA creates a circuit with fewer number of multi-controlled Toffoli gates that can efficiently compile them. However, as the number of variables increases, the number of required multi-controlled Toffoli gates also grows, resulting in less efficient compilation. Also, the proposed algorithm does not need any ancilla. As a result, benchmarks such as `TOF(5)`, `PRIME(5)`, and `HWB(5)` can be compiled for a 5-qubit quantum computer using the proposed approach, while the state-of-the-art approach cannot generate compatible circuits (see cells marked N/A). Further, for the experiments on the IBM quantum computer, it can be seen that more improvement is possible for the smaller quantum computers, indicating that my proposed algorithm better addresses the coupling constraints.

Table 4.1: Experimental results after compilation for Rigetti computer Agave 8Q.

		Rigetti Agave 8Q					
Permutation	#qubits	SOTA		Proposed			
		gates	depth	gates	Improve	depth	Improve
TOF(3)	3	52	29	58	-11.54%	40	-37.93%
TOF(4)	4	232	68	196	15.52%	76	-11.76%
TOF(5)	5	444	135	496	-11.71%	164	-21.48%
TOF(6)	6	1121	232	1002	10.62%	305	-31.47%
PRIME(3)	3	171	110	215	-25.73%	131	-19.09%
PRIME(4)	4	1063	391	745	29.92%	310	20.72%
PRIME(5)	5	5408	1450	2971	45.06%	1082	25.38%
HWB(4)	4	1252	483	1140	8.95%	443	8.28%
HWB(5)	5	7953	1359	3724	53.17%	1254	7.73%

Table 4.2: Experimental results after compilation for Rigetti computer Acorn 19Q.

		Rigetti Acorn 19Q					
Permutation	#qubits	SOTA		Proposed			
		gates	depth	gates	Improve	depth	Improve
TOF(3)	3	52	29	58	-11.54%	40	-37.93%
TOF(4)	4	225	63	196	12.89%	76	-20.63%
TOF(5)	5	429	119	496	-15.62%	164	-37.82%
TOF(6)	6	1142	291	1002	12.26%	305	-4.81%
PRIME(3)	3	181	115	215	-18.78%	131	-13.91%
PRIME(4)	4	1079	404	745	30.95%	310	23.27%
PRIME(5)	5	5590	1715	2971	46.85%	1082	36.91%
HWB(4)	4	1323	485	1140	13.83%	445	8.25%
HWB(5)	5	7482	2284	3724	50.23%	1254	45.10%

Table 4.3: Experimental results after compilation for IBM Yorktown and Tenerife 5Q.

		IBM 5Q Yorktown						IBM 5Q Tenerife					
Permutation	#qubits	SOTA		Proposed				SOTA		Proposed			
		gates	depth	gates	Improve	depth	Improve	gates	depth	gates	Improve	depth	Improve
TOF(3)	3	17	13	16	5.88%	12	7.69%	16	13	15	6.25%	12	7.69%
TOF(4)	4	162	87	68	58.02%	50	42.53%	81	52	82	-1.23%	54	-3.85%
TOF(5)	5	*N/A*		130		96		*N/A*		186		119	
TOF(6)	6	*N/A*		*N/A*				*N/A*		*N/A*			
PRIME(3)	3	47	37	51	-8.51%	39	-5.41%	45	36	52	-15.56%	39	-8.33%
PRIME(4)	4	616	364	338	45.13%	212	41.76%	496	319	281	43.35%	191	40.13%
PRIME(5)	5	*N/A*		896		596		*N/A*		1038		637	
PRIME(6)	6	*N/A*		*N/A*				*N/A*		*N/A*			
HWB(4)	4	622	384	413	33.60%	277	27.86%	740	436	434	41.35%	286	34.40%
HWB(5)	5	*N/A*		1196		770		*N/A*		1259		824	

Table 4.4: Experimental results after compilation for IBM computer Rueschlikon 16Q.

		IBM 16Q Rueschlikon					
Permutation	#qubits	SOTA		Proposed			
		gates	depth	gates	Improve	depth	Improve
TOF(3)	3	21	17	40	-90.48%	28	-64.71%
TOF(4)	4	112	67	92	17.86%	63	5.97%
TOF(5)	5	142	85	202	-42.25%	129	-51.76%
TOF(6)	6	407	251	473	-16.22%	265	-5.58%
PRIME(3)	3	115	81	120	-4.35%	83	-2.47%
PRIME(4)	4	517	338	339	34.43%	231	31.66%
PRIME(5)	5	2174	1339	1341	38.32%	838	37.42%
PRIME(6)	6	8290	4836	4099	50.55%	2434	49.67%
HWB(4)	4	598	357	447	25.25%	293	17.93%
HWB(5)	5	2891	1810	1665	42.41%	1013	44.03%

4.5 Summary

I presented a compilation algorithm to realize permutations in terms of quantum circuits for Rigetti's and IBM's superconducting computers. The proposed approach utilizes Young-subgroup based reversible logic synthesis [99, 39, 100], which for a given permutation for a n-qubit state, finds a sequence of $2n-1$ so-called single-target gates, which describe quantum operations to alter the quantum state w.r.t. a Boolean function. I described a general algorithm to translate a single-target gate into a quantum circuit composed of Clifford+R_z gates. Finally, I employed an explicit rewiring technique in order to reduce the number of quantum gates. The experimental evaluation shown that the proposed approach leads to quantum circuits with lower quantum gates and lower depth compared to state-of-the-art generic compilation techniques. For Rigetti QPUs, gate count and gate depth are reduced up to 53% and 45%, respectively, and for IBM QPUs, gate count and gate depth are reduced up to 58% and 49%, respectively.

Quantum State Preparation Part III

5 Problem Definition

Preparing quantum states is the process of bringing a qubit or a quantum register to a desired quantum state. The ability to prepare a qubit in a specific quantum state is a crucial component of many quantum algorithms and protocols, as well as quantum error correction and fault-tolerance schemes.

Preparing quantum states is important because many quantum algorithms rely on starting in a particular state. For example, the quantum Fourier transform (QFT) algorithm requires the input state to be a superposition of all possible input values. Similarly, the Grover's algorithm requires the input to be encoded in the amplitudes of the basis states. In addition, preparing quantum states is necessary in order to perform certain quantum operations, such as quantum error correction, where it is necessary to initialize the qubits in a known state to detect and correct errors. Furthermore, it plays a critical role in the development of quantum simulation algorithms, where the goal is to simulate the behaviour of complex quantum systems. In order to do so, it is necessary to prepare the quantum system in a particular state that corresponds to the physical system being simulated. Overall, preparing quantum states is essential for the implementation of quantum algorithms and quantum operations, and it is a fundamental building block for the development of new quantum technologies.

In general, preparing quantum states involves applying a sequence of quantum gates to a set of qubits that are in a known state, such as the ground state. The sequence of gates must be carefully chosen to generate the desired quantum state with high fidelity and minimal error. This process is called *Quantum State Preparation* (QSP).

A quantum state over n qubits is any superposition of all basis states $|0\rangle, |1\rangle, \ldots, |2^n - 1\rangle$ characterized by

$$|\varphi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle, \tag{5.1}$$

a column vector of 2^n *amplitudes* $\alpha_i \in \mathbb{C}$ where

$$|\alpha_0|^2 + \cdots + |\alpha_{2^n-1}|^2 = 1. \tag{5.2}$$

Each squared norm amplitude $|\alpha_i|^2$ indicates the probability that after measurement the n

qubits are in the classical state i.

Multiple algorithms [42, 43, 41, 44, 101, 102, 103, 104, 105, 106, 2] have been proposed for preparing *arbitrary quantum states*, which require an exponential number of CNOTs and runtime with respect to the number of qubits [107]. Consequently, preparing the state can become a bottleneck of an otherwise efficient quantum algorithm. Finding QSP methods that work efficiently in practice is thus an important research challenge.

Researchers address this problem with two possible approaches: (1) adding ancilla qubits, (2) approximate state preparation algorithms that focus on preparing a quantum state to a high precision with a fixed bound on the error, and (3) input state restrictions that allow researchers to characterize a family of states for which quantum state preparation can be performed precisely and efficiently.

Adding ancilla qubits to a quantum circuit can help to reduce the number of gates and circuit depth, but it also adds additional overhead in terms of the number of qubits required. Approximate state preparation algorithms allow one to prepare states with some inaccuracy, which is valid because quantum computers themselves experience gate and measurement errors, such that all computations are only correct as long as a certain error threshold is not exceeded. However, developing robust methods for fault-tolerant quantum computing with strong error correction is in general complicated and an obstacle that has yet not been resolved for quantum computers. Preparing quantum states approximately adds to this burden, leading even to more overhead. In my research, I follow the second approach, trying to identify families of quantum states that can be prepared efficiently while being absolutely precise.

As families of quantum states I consider 3 families, uniform quantum states, cyclic quantum states, and sparse quantum states. Uniform quantum states are quantum states that are evenly distributed over a set of basis states. In other words, they are quantum states that have equal probabilities of being in any of the basis states. Many quantum states are uniform such as Bell state, GHZ state, W state. Cyclic quantum states are a family of quantum states that are defined as uniform superposition of all the possible states of n qubits with a fixed Hamming weight (i.e., number of ones) and where the ones are adjacent and arranged cyclically. Cyclic states are important in quantum error correction because they have the property that applying a cyclic shift to the state corresponds to a phase shift in the Fourier domain. This property allows for the detection and correction of errors in a quantum computer or communication channel. Sparse quantum states are quantum states that have a small number of non-zero amplitudes. Sparse quantum states are of interest in quantum computing because they can be efficiently manipulated using quantum algorithms and techniques. This is because the quantum gates and operations used in quantum algorithms typically act on only a few qubits at a time, and sparse quantum states allow for efficient representation and manipulation of these qubits.

6 Uniform Quantum State Preparation

6.1 Introduction

I restrict my search to a special family of quantum states, aiming for an efficient and exact implementation without using ancilla qubits. As an important family of quantum states, I consider *Uniform Quantum States* (UQSs) in this chapter. These states are superpositions of basis states, where all amplitudes are either zero or have the same value.

UQSs form an important class of quantum states and have recently attracted attention from researchers. For example, they have been used in effective quantum machine learning as the quantum version of a uniform random sample [108]. Many well-known quantum states are uniform, such as the uniform superposition of all basis states, the Bell state, the GHZ state [48], and the W state [47]. They also appear frequently as initial states of important quantum algorithms, such as the quantum random-search algorithm *Grover Walk* [46], *Quantum Byzantine Agreement* [109], and *Secret Sharing* [110], which have large applications in quantum cryptography. Hence, having an efficient algorithm for preparing UQSs alongside arbitrary quantum states is important.

In this chapter, I propose a new algorithm for *UQS Preparation* (UQSP). The central idea of my work is a characterization of UQSs with Boolean functions. This characterization simplifies the problem because one simple Boolean function can describe an exponential number of amplitudes. Moreover, this enables one to apply well-understood and optimized techniques from logic synthesis [60]. I develop the theory and propose a functional decomposition method based on co-factoring for synthesis.

My decomposition method can be applied to any representation of Boolean functions. I use truth-table-based, and decision-diagram-based representations for smaller, and larger numbers of qubits, respectively. I show how I use decision diagrams to prepare UQSs. Decision diagrams enable a scalable quantum state preparation since many Boolean functions of practical interest have small representations, e.g., in terms of *Binary Decision Diagrams* (BDDs) [50].

Moreover, I present another orthogonal improvement that identifies dependencies among variables of the Boolean functions and determines an order in which the qubits should be prepared. I show that, if a functional dependency can be identified, the decomposition algorithm can be avoided in favor of more efficient quantum circuit constructions. This idea allows reducing the overall number of CNOT gates.

I develop the underlying theory of UQSP using Boolean methods and demonstrate empirically that an implementation of my approach outperforms a state-of-the-art algorithm for preparing arbitrary quantum states [101] that is implemented in an industrial framework, Qiskit [111]. My algorithm achieves better quality (smaller circuits) in significantly less runtime.

The contributions of this chapter can be summarized as follows:

1. I propose a Boolean algorithm to prepare UQSs which simplifies the problem and provides the ability to apply well-understood techniques from logic synthesis.
2. I apply my decomposition algorithm to the BDD representation of Boolean functions to enable a fast execution when the function representation is small (the algorithm runs in polynomial time with respect to the number of BDD nodes). Experimental results show that by increasing the number of qubits to at most 30, my method prepares states in milliseconds whereas Qiskit cannot prepare states for more than 18 qubits.
3. I investigate the problems of identifying functional dependencies among the qubits and determining the best order for preparing the qubits of a UQS. I present several heuristics for solving these two problems and show that one can construct quantum circuits for UQSP with different trade-offs between runtime and the number of CNOTs. The comparison with Qiskit shows that the proposed method achieves an average reduction on the number of CNOTs by 75.31% for practical benchmarks and the runtime is improved by almost a factor of 2. Hence, my algorithm for preparing UQSs can be integrated into Qiskit to improve the quality-of-results for this important family of quantum states.

6.2 Related works

The general problem of preparing arbitrary quantum states requires quantum circuits with an exponential number of CNOTs, and exponential depth and runtime in the worst case. Multiple algorithms have been proposed for preparing them.

The algorithm presented in [104] prepares quantum states with a divide-and-conquer strategy. Although it creates quantum circuits with poly-logarithmic depth, it uses additional n ancilla qubits and increases the number of elementary quantum gates. In [112], the authors present a way of preparing arbitrary quantum states with l unique amplitudes for the purpose of reducing T gates. To prepare a state, they start with a uniform superposition. If l is not a binary power, they prepare the initial superposition using amplitude amplification. Afterwards, by loading data from *Quantum Read-Only Memory* (QROM), they decide whether to keep the state or to alter it. Moreover, they prepare states with the help of ancilla qubits. The authors in [113] propose an algorithm to prepare only efficiently-integrable probability distributions. Their algorithm requires overheads by using additional ancilla qubits.

The authors in [102, 103] propose algorithms without using ancilla qubits, but they prepare quantum states approximately. Approximate state preparation algorithms introduce some inaccuracy, such that all computations are only correct as long as a certain error threshold is not exceeded.

In this chapter, I am interested in algorithms that prepare the quantum state exactly and without using ancilla qubits, which reduce the implementation overhead. These algorithms [42, 43, 41, 44] to prepare arbitrary quantum states, however, require an exponential number of CNOTs and runtime with respect to the number of qubits [107]. To reduce runtime and the number of CNOTs, I propose a Boolean algorithm and compare my results with [101] from this category that is a method for preparing arbitrary quantum states. The idea in [101] is based on dividing qubits in two parts and applying an iterative circuit. When n is even, each part is associated with $\frac{n}{2}$ qubits. For an odd number of qubits, the qubits are divided into $\frac{n-1}{2}$ and $\frac{n+1}{2}$ qubits parts. Then, a state on the first part is prepared, defined by the generalized Schmidt coefficients in the computational basis. Next, a set of CNOT gates between the qubits in the first and second parts are applied. Finally, a unitary operation on the first part of the qubits, followed by another unitary operation on the second part of the qubits are performed. After decomposing unitaries, finally, the number of CNOTs is bounded by $\frac{23}{24}2^n$ and $\frac{115}{96}2^n$ for n even and odd, respectively.

6.3 UQSP motivation

6.3.1 UQSP problem

I consider n-qubit quantum states that are uniform superpositions over a non-empty subset of the basis states $|0\rangle, |1\rangle, \dots, |2^n - 1\rangle$. In such quantum states all amplitudes of the state vector are either 0 or have the same value $\alpha = \frac{1}{\sqrt{s}}$, where s is the size of the subset of basis states. I exploit the fact that such states can be characterized by a Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ such that $f(x) = 1$, if and only if $|x\rangle$ is in the subset of the considered basis states, and therefore its corresponding amplitude is non-zero.

Example 6.3.1. *The uniform quantum state*

$$|\varphi\rangle = \frac{1}{\sqrt{3}}(0,0,0,1,0,1,1,0))^T, \tag{6.1}$$

which is the W state over 3 qubits, corresponds to the Boolean function f_W that has the truth table

$$f_W(x_1, x_2, x_3) = (0,0,0,1,0,1,1,0)^T. \tag{6.2}$$

Proposition 1. *There is a one-to-one correspondence between uniform quantum states and Boolean functions. A uniform quantum state $|\varphi_f\rangle$ corresponds to the normalized truth table of f*

$$|\varphi_f\rangle = \frac{f}{\sqrt{|f|}} = \frac{1}{\sqrt{|f|}} \sum_{x \in \mathrm{Minterms}(f)} |x\rangle.$$

Now, the UQSP problem refers to the problem of finding a quantum circuit that prepares such

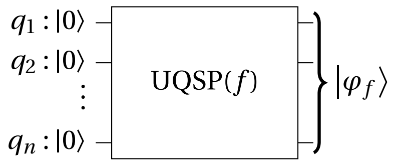


Figure 6.1: The problem of preparing the UQS corresponding to the given Boolean function $f(x_1, x_2, ..., x_n)$ as input.

a state, given as input a Boolean function f in some representations. In the beginning, all qubits are assumed to be in their zero states. Then, I am searching for a construction that transforms a given unitary matrix UQSP(f) into a circuit, where

$$\mathrm{UQSP}(f)|0\rangle^{\otimes n} = |\varphi_f\rangle. \tag{6.3}$$

Fig. 6.1 describes my problem formulation.

6.3.2 Motivational examples

Uniform quantum states can be found in various quantum algorithms, e.g., the uniform superposition of all basis states, for which $f = 1$ (tautology), the Bell state, for which $f = \bar{x}_1 \oplus x_2$, the generalized GHZ state, for which $f = \bar{x}_1\bar{x}_2 \dots \bar{x}_n \oplus x_1x_2 \dots x_n$, and the generalized W state, for which $f = [x_1 + x_2 + \cdots + x_n = 1]$ that means only one of the variables is one and the others are equal to zero.

UQSs are used in several protocols in quantum communication and cryptography, for example, in the *Quantum Byzantine Agreement* [114, 109]. Byzantine agreement protocols are important algorithms that are robust to failures in distributed computing. A group of n players must agree on a bit despite the faulty behaviour of some of the players. Quantum Byzantine agreement represents the quantum version of Byzantine agreement which works in constant time [109]. It is shown that in this protocol, for n players, it is required to prepare two quantum states that are uniform, namely

$$\big|\varphi_1\big\rangle = \frac{1}{\sqrt{2}}(|0,0,\dots,0\rangle + |1,1,\dots,1\rangle) \tag{6.4}$$

on n qubits, and

$$\big|\varphi_2\big\rangle = \frac{1}{\sqrt{n^3}} \sum_{a=1}^{n^3} |a\rangle \tag{6.5}$$

on n qubits.

Although the quantum state in (6.4) represents the well-known GHZ state and one can apply a template to prepare it, the second state in (6.5) is more general and one require an automated algorithm to prepare it.

As another example, the initial quantum state used in the Quantum Coupon Collector [108] problem is also uniform. It is given as

$$|\varphi\rangle = \frac{1}{\sqrt{|S|}} \sum_{i \in S} |i\rangle \tag{6.6}$$

over the elements of an unknown k-element set $S \subseteq \{1, \ldots, n\}$, where $k = |S| < n$.

Moreover, UQSs are helpful when one want to extend a problem. In this domain, I already have the results for some parts of the problem and I only need to solve the problem for a new part. Instead of preparing all possible input assignments, I only need to prepare the input assignments for the new part that are UQSs. As an example, consider the Zed city problem introduced in[58], which is an instance of vertex coloring. As some of the nodes are colored already, it requires a UQS in the beginning to create the desired input assignments for uncolored nodes.

Hence, all these applications show that having an automated algorithm to prepare UQSs efficiently is very important.

6.4 Using functional decomposition for UQSP

To prepare an n-qubit uniform quantum state $|\varphi_f\rangle$ associated with the Boolean function f, I define a correspondence between the qubits in the quantum circuit and the variables in f. Hence, the uniform quantum state $|\varphi_f\rangle$ and the qubit q_i correspond to the Boolean function f and the variable x_i, respectively. In the remainder of this chapter, I will use these symbols interchangeably.

Representing uniform quantum states as Boolean functions give me the opportunity to apply *Shannon's decomposition* to solve the state preparation problem recursively. To construct the desired quantum circuit corresponding to the UQSP(f) block shown in the Fig. 6.1, I iterate over the variables of the Boolean function in an order, and prepare them one by one by applying Shannon's decomposition. For each qubit q_i (x_i), I decompose UQSP blocks as follows.

1. A gate $G(P)$ is applied, which is a unitary transformation and satisfies

   $$G(P)|0\rangle = \sqrt{P}|0\rangle + \sqrt{1-P}|1\rangle. \tag{6.7}$$

   The parameter P represents the probability of $q_i(x_i)$ being zero in the current decomposed function ($\hat{f}$), which equals to $p_{\hat{f}}(\bar{x}_i)$.

2. q_i is either zero or one. Hence, I construct new blocks for the corresponding cofactors by applying negative-control and positive-control, respectively.

Fig. 6.2 shows the construction of UQSP(f) using my functional decomposition for one iteration where $i = 1$. I formulate the general idea of my state preparation algorithm as

$$\mathrm{UQSP}(f)|0\rangle^{\otimes n} = (\mathrm{UQSP}(f_{\bar{x}_i}) \oplus \mathrm{UQSP}(f_{x_i}))(G(p_f(\bar{x}_i)) \otimes I_{2^n-1})|0\rangle^{\otimes n}. \tag{6.8}$$

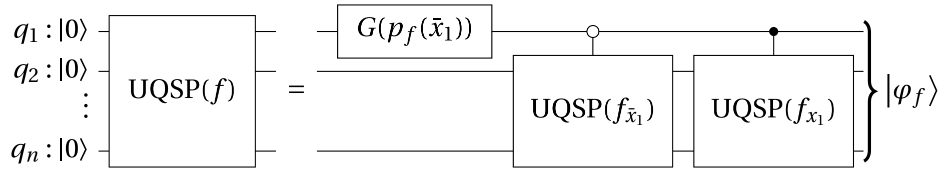


Figure 6.2: The preparation of the quantum state associated by Boolean function f using functional decomposition.

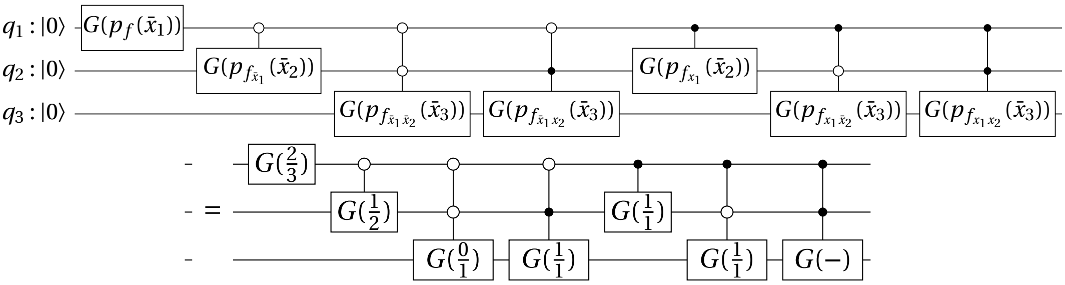

$q_1 : |0\rangle$
$q_2 : |0\rangle$
$q_3 : |0\rangle$
$G(p_f(\bar{x}_1))$
$G(p_{f_{\bar{x}_1}}(\bar{x}_2))$
$G(p_{f_{\bar{x}_1\bar{x}_2}}(\bar{x}_3))$
$G(p_{f_{\bar{x}_1x_2}}(\bar{x}_3))$
$G(p_{f_{x_1}}(\bar{x}_2))$
$G(p_{f_{x_1\bar{x}_2}}(\bar{x}_3))$
$G(p_{f_{x_1x_2}}(\bar{x}_3))$
=
$G(\frac{2}{3})$
$G(\frac{1}{2})$
$G(\frac{0}{1})$
$G(\frac{1}{1})$
$G(\frac{1}{1})$
$G(\frac{1}{1})$
$G(-)$

Figure 6.3: Multi-controlled single-target gates of UQSP(f_W).

By applying this procedure recursively for all variables, I obtain the desired quantum circuit.

Example 6.4.1. *Applying the proposed functional decomposition algorithm for the Boolean function of Example 6.3.1 (f_W) in the order x_1, x_2, x_3 results in Fig. 6.3. First, I decompose f over the variable x_1. I apply $G(p_f(\bar{x}_1))$ and divide the circuit into two parts corresponding to cofactors $f_{\bar{x}_1}$ and f_{x_1} by adding negative and positive controls, respectively. Next, I decompose with x_2, leading to four cofactors $f_{\bar{x}_1\bar{x}_2}$, $f_{\bar{x}_1x_2}$, $f_{x_1\bar{x}_2}$ and $f_{x_1x_2}$. Finally, the four cofactors are decomposed with x_3. The detailed structure is shown on the left-hand side of '=' in the figure. It is visible that preparing each qubit requires 2^k MC-gates with k controls with different polarities corresponding to different cofactors.*

By applying the definition in (2.1)*, the probability values for G gates are computed as the right-hand side of '=' in the figure. The target-qubit represented by* $G(-)$ *specifies a division by zero corresponding to the zero-probability* $G(\frac{0}{0})$*. This case never happens and I do not insert any gate for that. Moreover,* $G(1)$ *shows that the qubit is always equal to the zero state and can be safely ignored, too.*

The resulted quantum circuit consists of a sequence of multi-controlled single-target gates with $G(P)$ as target. From the definition of $R_y(\theta)$ and (6.7) one can readily derive that

$$G(P) = R_y\left(2\cos^{-1}(\sqrt{P})\right). \tag{6.9}$$

Consequently, by replacing all gates on the target-qubit by R_y gates, I obtain a circuit consisting only of *multi-controlled R_y* (MC-R_y) gates.

Example 6.4.2. *Fig. 6.4 shows the quantum gate realization of G gates for* UQSP(f_W) *in Fig. 6.3 using* (6.9).

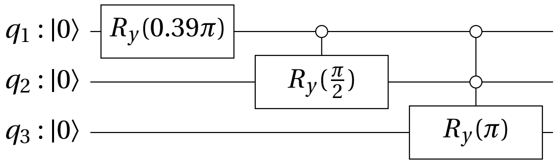


Figure 6.4: A sequence of MC-R_y gates for UQSP(f_W).

6.5 UQSP using binary decision diagrams

In practice, it is infeasible to store Boolean functions using truth tables for a large number of variables (typically more than 15 variables). As an alternative, the proposed approach can extract the quantum circuit for UQSP(f) when f is represented as a *Reduced Ordered BDD* (ROBDD, or BDD for short) [76]. This is due to the fact that counting all minterms and computing cofactors can be efficiently performed using BDDs. The compact representation not only enables a scalable quantum state preparation, if the BDD representation for f is small, but also reduces the number of multi-controlled single-target gates and maybe CNOTs.

In particular, in this section I propose an algorithm that works directly on decision diagrams. This enables a scalable quantum state preparation, since many Boolean functions of practical interest have small representations, e.g., in terms of binary decision diagrams (BDDs) [50].

I propose an automatic quantum state preparation algorithm, which takes as input a Boolean function and produces a sequence of multi-controlled gates. Afterwards, to run on a physical quantum computer, I use decomposition methods to generate a quantum circuit over CNOTs and single-qubit quantum gates. The detailed contributions are summarized as follows:

- Proposing an algorithm that works on BDDs to enable a fast execution when the function representation is small (algorithm runs in polynomial time with respect to the number of BDD nodes).
- Reducing the number of elementary quantum gates by removing redundancies in the BDDs as well as applying a post-optimization technique for the GHZ state.

Experimental results show that the proposed approach can achieve a significant reduction in run-time compared to a state-of-the-art approach which relies on an explicit quantum state representation implemented in IBM's Qiskit framework. Moreover, the results show that I reduce the number of elementary quantum gates over the state of the art.

6.5.1 Proposed algorithm

Algorithm 6.1 shows the UQSP using decision diagrams. First in the procedure *ComputeZeroProbabilities*, I traverse the BDD in top-down post-order to compute zero probabilities by dividing the number of ones in the zero-child by the number of ones in the current node. The number of ones of the current node is equal to the summation of the number of ones of the zero-child and the one-child.

Algorithm 6.1: UQSP using decision diagrams.

```
Input: The root $r$ of the BDD of the Boolean function $f$
Output: Quantum circuit $QC$
Proc $UQSP_{DD}(r)$:
    $p = \{\}$
    $ones = \{\}$
    ComputeZeroProbabilities($r, p, ones$)
    $QC = \{\}$
    ComputeMCgates($r, p, QC$)
    return $QC[r]$

Proc ComputeZeroProbabilities($v, p, ones$):
    if IsTerminal(v) or IsVisited(v) then
        return
    ComputeZeroProbabilities($low(v), p, ones$)
    ComputeZeroProbabilities($high(v), p, ones$)
    $low\_ones$ = ComputeOnesFromChild($v, low(v), ones$)
    $high\_ones$ = ComputeOnesFromChild($v, high(v), ones$)
    $p[v] = \frac{low\_ones}{low\_ones + high\_ones}$
    $ones[v] = low\_ones + high\_ones$
    return

Proc ComputeOnesFromChild($v, c, ones$):
    if IsZeroTerminal(c) then
        return 0
    R_nodes = NumberOfReducedNodes($v, c$)
    if IsOneTerminal(c) then
        return $2^{R\_nodes}$
    return $ones[c] \times 2^{R\_nodes}$

Proc ComputeMCgates($v, p, QC$):
    if IsTerminal(v) or IsVisited(v) then
        return
    ComputeMCgates($low(v), p, QC$)
    ComputeMCgates($high(v), p, QC$)
    InsertGgate($p[v]$, QC[$v$])
    ApplyControlToChildGates($v, low(v)$,‘Negative’, $QC$)
    ApplyControlToChildGates($v, high(v)$,‘Positive’, $QC$)
    InsertHalfPGatesForRNodes($v, low(v)$,‘Negative’, $QC$)
    InsertHalfPGatesForRNodes($v, high(v)$,‘Positive’, $QC$)
    return

Proc ApplyControlToChildGates($v, child, control\_type, QC$):
    for $i = 0, \ldots, n-1$ do
        foreach MCgate with $G$ on $q_i \in QC[child]$ do
            MCgate. AddControl(control_type, Index($v$))
            QC[$v$].AddMCgate(MCgate)
    return

Proc InsertHalfPGatesForRNodes($v, child, control\_type, QC$):
    for $i = Index(v) + 1, \ldots, Index(child)$ do
        MCgate = CreateMCgate($\frac{1}{2}, q_i$)
        MCgate. AddControl(control_type, Index($v$))
        QC[$v$].AddMCgate(MCgate)
    return
```

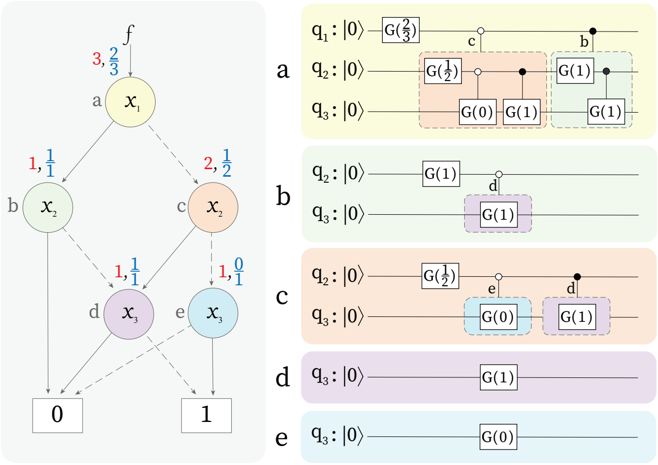


Figure 6.5: BDD representation of f_W and the procedure of the preparing it.

To compute the number of ones of the zero-child and the one-child, I consider the effect of the reduced nodes between these nodes and the current node using *ComputeOnesFromChild*. Secondly, in the procedure *ComputeMCgates*, I traverse again in top-down post-order to extract multi-controlled single-target gates (MC-gates). For each node, I insert a G gate with its zero probability. Next, I connect a negative-control and a positive-control to the gates from the zero-child and the one-child, respectively, using procedure *ApplyControlToChildGates*. I also consider the effect of the reduced nodes in the path between the current node and its children. Both children of the reduced nodes have ones in the Boolean function, so I insert gates with $\frac{1}{2}$ probabilities in procedure *InsertHalfGatesForRNodes*. Finally, at the root-node, I have all MC-gates.

Example 6.5.1. *I illustrate the BDD-based synthesis algorithm for the f_W function as an example in Fig. 6.5. The BDD consists of 5 nodes. I traverse the BDD in top-down post-order to count the number of ones and the zero probabilities by dividing the number of ones from the zero-child (dashed line) by the number of ones of the current node, for each node. The number of ones and the probabilities for each node are shown within the figure on the left-hand side in red and blue colors, respectively.*

To construct MC-gates, I again traverse the BDD in top-down post-order. In a recursive manner, I construct for each node a circuit that consists of one $G(P)$ gate, a negative-controlled application of the circuit constructed by the zero-child, as well as a positive-controlled application of the circuit constructed by the one-child.

These gates are located in the figure using boxes at the right-hand side of the figure for each node with the same color. As an example, I explain the construction of the gates for node 'b' (green color). First, I have to apply $G(\frac{1}{1})$ to the qubit corresponding to 'b', then I connect the negative-control to the gates of node 'd'. The one-child is connected to constant zero, meaning that it is not included in the minterms of the function and I do not need to consider it, so I do not insert any gate for the one-child.

6.5.2 Experimental evaluation

In this section, I discuss the experimental setup and results.

I implemented the proposed approach into *angel*, a C++ library for quantum state preparation, which is introduced in detail in Chapter 9. I used the *CUDD*[1] library for BDD representation and traversal. To compare my results with the state of the art, I made use of IBM's Qiskit [111] that implemented the algorithm in [101]. I performed experiments for the two most-known quantum states GHZ and W. As there is no standard benchmark set for quantum state preparation and my method utilizes Boolean functions, I further used the ISCAS benchmarks as practical benchmarks to extract large functions with different numbers of variables which correspond to the number of qubits (n). Since ISCAS benchmarks have multiple outputs, I extracted the logic cone for a given primary output. All experiments have been conducted on an Intel Core i7, 2.7 GHz with 16 GB memory.

Experimental results are shown in Table 6.1. The first column names the benchmark. The suffixes for the ISCAS benchmarks correspond to the indices of the extracted logic cones. I evaluate the proposed method for run-time and circuit size. I further discuss a way to optimize the number of gates.

Run-time: I track the actual run-time, and also report the number of nodes in the BDD, since the algorithm's complexity depends on the number of nodes. There are two columns in Table 6.1 that show run-time for the proposed method and the state of the art. Experimental results show that my proposed method reduces the run-time significantly for all cases. Moreover, as the proposed method uses decision diagrams in the implementation, the number of nodes is extracted only for the proposed method. A timeout of 9000 seconds is considered to extract the results. The results show that generating circuits for quantum state preparation using the proposed method is fast. When the number of qubits grows too large, approaches based on explicit state representation require too much time (see the cells marked TO).

Circuit size: Experimental results regarding circuit size are evaluated in terms of the number of MC-R_ys, the number of elementary quantum gates (CNOT and single-qubit gates), and ancilla qubits. These results are summarized in Table 6.1 for both the proposed method and the state-of-the-art approach. Note that I only extracted MC-R_ys for the proposed method. The number of MC-R_ys is reduced using decision diagrams by removing redundancies. As shown in the table, I reduce the number of MC-R_ys from $2^n - 2$ into $n-1$ for GHZ and W. Moreover, MC-R_ys are reduced for ISCAS benchmarks instead of growing exponentially. As I generate MC-R_ys, my algorithm has the potential to represent these gates in a compact way in terms of Boolean functions. It is an advantage of the proposed approach to provide the high-level representations for MC-R_y gates such that different decomposition methods may be applied later.

I transform the MC-R_ys into elementary quantum gates using decomposition methods such as [98]. The method presented in [98] requires ancilla qubits that are shown in the table. The results show that reducing CNOTs and single-qubit gates (R_ys for the proposed method) is comparable over the state of the art. As I discussed before, the upper bound on CNOTs and

[1] CUDD: CU Decision Diagram package, https://github.com/ivmai/cudd.

Table 6.1: Experimental results regarding the number of MC-R_y rotation gates, elementary quantum gates and time.

		Proposed approach						State-of-the-art approach		
QSs	Qubits	Nodes	MC-R_ys	CNOTs	SQGs	ancilla	Time	CNOTs	SQGs	Time
GHZ	15	29	14	539	720	6	< 0.01	32752	97857	60.57
GHZ	18	35	17	809	1083	8	< 0.01	262125	786071	929.03
GHZ	20	39	19	1027	1389	9	< 0.01	1048555	3145488	7809.02
GHZ	27	53	26	1951	2628	12	< 0.01	*TO*	*TO*	*TO*
GHZ	30	59	29	2437	3279	14	< 0.01	*TO*	*TO*	*TO*
W	15	29	14	539	720	6	< 0.01	32752	98175	104.8
W	18	35	17	809	1083	8	< 0.01	262125	786255	5087.05
W	27	53	26	1951	2628	12	< 0.01	*TO*	*TO*	*TO*
W	30	59	29	2437	3279	14	< 0.01	*TO*	*TO*	*TO*
c17-0	4	6	11	14	15	1	< 0.01	11	39	0.03
c17-1	4	6	8	14	15	1	< 0.01	11	37	0.03
c432-0	18	18	2024	103074	126514	8	< 0.01	262125	3399580	1801.26
c432-1	27	3795	1256482	98417906	108854256	12	16.35	*TO*	*TO*	*TO*
c7552-65	29	7389	2847926	241536510	277309441	13	45.34	*TO*	*TO*	*TO*
c7552-66	26	3501	681374	45046398	48877056	12	9.21	*TO*	*TO*	*TO*
c7552-67	23	1653	163474	6994910	7928448	10	1.73	*TO*	*TO*	*TO*
c7552-68	20	793	41188	1048574	1048575	9	0.37	1048576	3015488	5948.47

TO: time-out of 9000 seconds.

single-qubit gates are $2^n - 2$ and $2^n - 1$, respectively. The results show that I reduce the number of elementary quantum gates as much as possible while for the state of the art are close to the upper bounds.

Optimization: To prepare the GHZ state, my method generates one MC-R_y on each qubit with positive controls. Fig. 6.6 on the left-hand side shows this sequence of gates for GHZ state on 4 qubits. As all qubits in the beginning are assumed to be in state 0, each qubit can alter to 1 only in one case, when the corresponding controls are 1. Hence, the last control is 1 when all previous controls are 1. Therefore, I can reduce redundant controls and only keep the last control, which results in a sequence of $n - 1$ single-controlled R_y gates (decomposing these gates results in $n - 1$ CNOTs and one R_y rotation gate). The optimized circuit for GHZ on 4 qubits is shown on the right-hand side of Fig. 6.6.

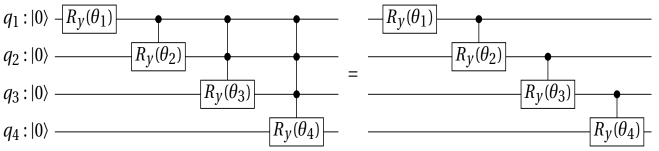


Figure 6.6: Reducing the number of controls for GHZ state.

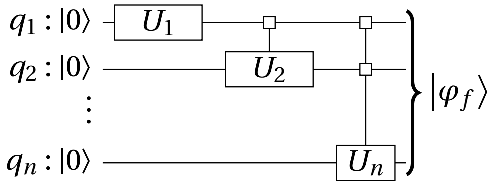


Figure 6.7: The general structure of the sequential preparation of qubits using uniformly-controlled single-target gates.

6.6 UQSP using dependency analysis methods

For each qubit q_i, the recursion generates 2^k MC-gates. The variable k denotes the number of already prepared qubits that can be used as control qubits for preparing the current qubit q_i. The variable k can range from 0 to $n-1$.

This sequence of multi-controlled single-target gates on the same target-qubit can be fused into a uniformly-controlled single-target gate U_i per qubit. Applying this procedure for all variables results in the structure depicted in Fig. 6.7, where each qubit is prepared with a uniformly-controlled single-target gate in the order from 1 to n.

On one hand, the structure of the quantum circuit presented in Fig. 6.7 shows that preparing qubits depends on all previously prepared qubits. On the other hand, formulating the UQSP problem using Boolean functions allows me to identify functional dependencies among variables. If a dependency function is recognized, meaning that a qubit depends on a subset of previously prepared qubits, then often a more compact quantum circuit structure can be synthesized. This helps to further reduce the number of control qubits and CNOTs. An identified functional dependency for q_i can be utilized in three ways: 1) to reduce the number of control qubits if q_i depends only on a subset of the previously prepared qubits, 2) to reduce the number of elementary quantum gates if the functional dependency can be well-expressed with a library of hardware-supported quantum gates, and 3) to reduce the number of control qubits for preparing other next qubits.

6.6.1 Proposed method

Algorithm 6.2 shows the UQSP using *Functional Dependencies* (UQSP$_{\mathrm{FD}}$) in pseudo-code. As inputs, it takes a Boolean function f that defines a uniform quantum state and two strategies R_A and D_A for variable reordering and dependency analysis. Both strategies can be either implemented as exact or heuristic algorithms, which allow choosing between different runtime-quality trade-offs. Different methods for dependency analysis and variable reordering are explained in Sections 6.6.2 and 6.6.4, respectively.

Algorithm 6.2: UQSP using functional dependencies.

```
Input: Boolean function $f$, dependency analysis algorithm $D_A$, variable reordering algorithm $R_A$
Output: Quantum circuit $QC$, qubits order
Proc $UQSP_{\mathrm{FD}}(f, R_A, D_A)$:
    $QC$ = SynthesizeQC$_{\mathrm{FD}}(f, D_A(f))$
    $cost$ = CNOTs($QC$)
    $f_{\mathrm{best}} = f$
    foreach reordered $f' \in R_A(f)$ do
        $QC_{f'}$ = SynthesizeQC$_{\mathrm{FD}}(f', D_A(f'))$
        $cost_{f'}$ = CNOTs($QC_{f'}$)
        if $cost_{f'} < cost$ then
            $QC = QC_{f'}$
            $cost = cost_{f'}$
            $f_{\mathrm{best}} = f'$
    return $QC$, Order($f_{\mathrm{best}}$)

Proc $SynthesizeQC_{\mathrm{FD}}(f, D)$:
    $QC$ = CreateNewQC()
    $n$ = NumberOfVariables($f$)
    for $i = n-1, \ldots, 0$ do
        $QC$.CreateQubit($i$)
        $d_i$ = $D$.FindDependency($i$)
        if $d_i \neq \perp$ then
            $QC$.CreateGatesForDependencyFunction($i$, $d_i$)
        else
            $QC$.CreateGatesRecursively($i$)
    return $QC$
```

From a birds-eye perspective, UQSP$_{\mathrm{FD}}$ applies the dependency analysis algorithm D_A to the Boolean function f and all reordered functions f' of f suggested by R_A. The algorithm then synthesizes a quantum circuit by considering dependency functions extracted from D_A, counts the number of CNOTs, and returns the best quantum circuit realized for the function. The considered cost function focuses on reducing CNOTs which correlates with the reduction of rotation gates such that the algorithm minimizes CNOTs and rotation gates.

The procedure *SynthesizeQC*$_{\mathrm{FD}}$ shows how a quantum circuit is realized from a Boolean function f with a fixed order of variables and a fixed set of functional dependencies D. The procedure iterates one by one over the variable indices, checks if a dependency function for this index is in D, and then either synthesizes the quantum circuit for the given dependency function or uses the recursive decomposition of Eq. 6.8. Dependency functions are only computed if they are guaranteed to reduce the number of CNOTs when compared to the recursive decomposition.

The detailed explanation of proposed dependency analysis methods (D_A), CNOTs cost functions, and variable reordering methods (R_A) are described next.

6.6.2 Dependency analysis methods

Suppose that $f : \mathbb{B}^n \to \mathbb{B}$ is a Boolean function over Boolean variables $x = x_1, \ldots, x_n$. I attempt to compute a series $g_2, \ldots, g_n$ of dependency functions $g_i(x_1, \ldots, x_{i-1})$ such that

$$x_i \cdot f(x) = g_i(x_1, \ldots, x_{i-1}) \cdot f(x) \qquad \text{for} \quad i = 2, \ldots, n. \tag{6.10}$$

In other words, I compute a dependency for each variable (qubit) that represents it as a function of the previously prepared qubits. Hence, it starts with the second qubit in the sequence (i.e., at index $i = 2$):

$$x_i = g_i(x_1, \ldots, x_{i-1}) \qquad \text{for} \quad i = 2, \ldots, n. \tag{6.11}$$

I define γ as a cost function that assigns integers to Boolean operators. As the cost function, I consider the number of CNOTs required to prepare the qubits with the dependency function or the recursive construction of Eq. 6.8 if no dependency function exists. As a result, $\gamma(g_i)$ shows the number of CNOTs required for creating dependency function g_i. Finally, my goal is to minimize

$$\gamma = \sum_{i=2}^{n} \gamma(g_i). \tag{6.12}$$

I propose two algorithmic strategies to compute dependency functions: (i) pattern search and (ii) SAT-based ESOP synthesis. For a given Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ over Boolean variables $x = x_1, \ldots, x_n$, both strategies iterate over the x_is for $1 \le i \le n$ and attempt to identify one dependency function g_i.

1. Pattern search. I iterate over all k-tuples $(v_1, \ldots, v_k)$ of the set $\{x_1, \ldots, x_{i-1}\}$ over Boolean variables for increasing values of k, $1 \le k \le K$, where K is a fixed upper bound, and test if

   $$x_i \cdot f(x) = g(v_1, \ldots, v_k) \cdot f(x), \tag{6.13}$$

   for all dependency functions g from some fixed set of patterns. If the test succeeds, $g(v_1, \ldots, v_k)$ is a dependency function for x_i. I consider three different types of dependency functions: the identity function with a single argument to identify a dependency on a single variable or its negation, the k-ary XOR function to identify XOR relations between multiple variables or their negations, and the k-ary AND function to identify AND relations between multiple variables or their negations.

2. SAT-based ESOP synthesis. I attempt to compute a dependency function in two steps:

   Determining functional support: In the first step, I decide on the support of the dependency function using distinguishing bit-pairs [115]. I compute the distinguishing bit-pairs of $t(x) = x_i \cdot f(x)$ and sort the variables $x_1, \ldots, x_{i-1}$ by the number of distinguished bit-pairs with respect to $t(x)$. I accumulate a set $S \subseteq \{x_1, \ldots, x_{i-1}\}$ of variables in this order until $t(x)$ is guaranteed to be "reconstructable" using S, i.e., for each distinguishing bit-pair in $t(x)$, there is a variable in S with the same distinguishing bit-pair. ***Synthesizing structure***: In the second step, I use SAT-based synthesis [66] to derive an ESOP form g of the dependency function with support S that satisfies Eq. 6.10. Note that my cost function prioritizes XORs with many fanins and ANDs with few fanins.

6.6.3 CNOT costs

I define a cost function γ that counts the number of CNOTs required to realize a dependency function $g_i(x)$ as a quantum circuit based on the general constructions proposed by Schuch and Siewert [95], Welch et al. [94], and Mottonen et al. [42]. Since in the beginning, all qubits are zero, the relative phase of the rotation gates is zero, which allows reducing the number of gates.

I distinguish three cases:

1. *XOR clause.* Let $g_i(x)$ be an m-ary XOR clause $t(x) = l_1 \oplus \cdots \oplus l_m$ of literals l_j, $1 \leq j \leq m$. The qubit q_i can be prepared by a quantum circuit using $\gamma_1(t)$ CNOTs, where

$$\gamma_1(t) = |t| \tag{6.14}$$

   is the number of literals.

2. *Product term.* Let $g_i(x)$ be a product term $t(x) = l_1 \cdots l_m$ of literals l_j, $1 \leq j \leq m$. The qubit q_i can be prepared using a quantum circuit which requires $\gamma_2(t)$ CNOTs, where

$$\gamma_2(t) = \begin{cases} |t| & \text{if } |t| \in \{0,1\} \\ 2^{|t|} & \text{if } |t| > 1. \end{cases} \tag{6.15}$$

   In the case of no dependency, the number of CNOTs is zero. When the number of literals is one, it indicates that there is an equality dependency, so the number of CNOTs is one. For more than one literal the dependency pattern is a multi-controlled NOT gate. If I consider a multi-controlled NOT gate and some identity gates, I can utilize the decomposition method presented in [42] for uniformly-controlled single-target gates. Applying this decomposition method results in $2^{|t|}$ CNOTs.

3. *ESOP form.* Let $g_i(x) = t_1(x) \oplus \cdots \oplus t_m(x)$ be an ESOP form of product terms $t_j(x)$, $1 \leq j \leq m$. I express the dependency pattern by a sequence of multi-controlled NOT gates. Then, I propose two different ways to decompose this sequence, which require different numbers of CNOTs.

   Multi-controlled NOT gate decomposition prepares the qubit q_i using $\gamma_3(t_1 \oplus \cdots \oplus t_m)$ CNOTs, where

$$\gamma_3(t_1 \oplus \cdots \oplus t_m) = \gamma_2(t_{j^\star}) - \gamma_4(t_{j^\star}) + \sum_{i=1}^{m} \gamma_4(t_i), \tag{6.16}$$

   with

$$\gamma_4(t_i) = \begin{cases} |t_i| & \text{if } |t_i| \in \{0,1\} \\ 2^{|t_i|+1} - 2 & \text{if } |t_i| > 1 \end{cases} \tag{6.17}$$

and

$$j^\star = \underset{i=1,\ldots,m}{\operatorname{argmax}} |\gamma_2(t_j) - \gamma_4(t_j)|. \tag{6.18}$$

As the initial state is zero in the beginning, I decompose the first gate using [42] with fewer CNOTs and its cost is represented in Eq 6.15. For the rest I use the methods presented in [95, 94] that do not require ancilla qubits. Their corresponding cost is represented in Eq. 6.17. To reduce the number of CNOTs, I bring the gate with more controls to the beginning and its index is computed by Eq. 6.18. As the summation $\sum_{i=1}^{m} \gamma_4(t_i)$ in Eq. 6.16 accumulates the cost for all gates, the first gate's cost ($\gamma_4(t_{j^\star})$) is subtracted from the result.

Uniformly-controlled single-target gate decomposition is applicable if the literals of the product terms t_j are subsets of each other. Then, they can be considered as controls of a uniformly-controlled single-target gate. In comparison to the recursive construction, using uniformly-controlled single-target gates leads to fewer controls and reduces the number of CNOT gates, which are computed as follows:

$$\gamma_5(t_1 \oplus \cdots \oplus t_m) = \gamma_2(t_{j^\star}), \tag{6.19}$$

where

$$j^\star = \underset{j=1,\ldots,m}{\operatorname{argmax}} |t_j|. \tag{6.20}$$

For a given dependency function g_i in the ESOP form, I compute both $\gamma_3(g_i)$ and $\gamma_5(g_i)$, and return the minimum of them as the number of CNOTs.

Table 6.2 shows common cases of dependency functions, their CNOT costs, and realizations as quantum circuits by example.

Example 6.6.1. *Consider f_W in Example 6.3.1. It can be represented with its minterms as*

$$f_W(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2\bar{x}_3 + x_1\bar{x}_2\bar{x}_3. \tag{6.21}$$

I derive the dependency function

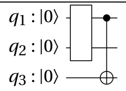
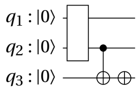
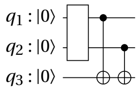
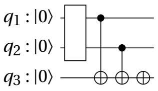
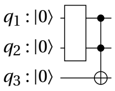
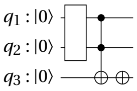
$$x_3 = g_3(x_1, x_2) = \neg(x_1 \oplus x_2) \tag{6.22}$$

with the CNOT cost

$$\gamma(g_3) = 2. \tag{6.23}$$

By considering the quantum circuit corresponding to the XNOR dependency function in Table 6.2, I get the circuit in Fig. 6.8 to prepare f_W using the UQSP$_{FD}$ algorithm. Comparing two figures 6.4 and 6.8, shows that I reduce the number of CNOTs by 4. Note that the 2-controlled $R_y(\pi)$ gate in Fig. 6.4 decomposes into 6 CNOTs.

Table 6.2: A list of common patterns of dependency functions, their CNOT costs, and an example of their realization as a quantum circuit (for two inputs).

Dependency function	Cost (#CNOTs)	Quantum circuit (for one and two inputs)
$g_i = l_1$	1	 $q_1 : \|0\rangle$ $q_2 : \|0\rangle$ $q_3 : \|0\rangle$
$g_i = \neg l_1$	1	 $q_1 : \|0\rangle$ $q_2 : \|0\rangle$ $q_3 : \|0\rangle$
$g_i = l_1 \oplus \ldots \oplus l_m$	m	 $q_1 : \|0\rangle$ $q_2 : \|0\rangle$ $q_3 : \|0\rangle$
$g_i = \neg(l_1 \oplus \ldots \oplus l_m)$	m	 $q_1 : \|0\rangle$ $q_2 : \|0\rangle$ $q_3 : \|0\rangle$
$g_i = l_1 \cdots l_m$	2^m	 $q_1 : \|0\rangle$ $q_2 : \|0\rangle$ $q_3 : \|0\rangle$
$g_i = \neg(l_1 \cdots l_m)$	2^m	 $q_1 : \|0\rangle$ $q_2 : \|0\rangle$ $q_3 : \|0\rangle$

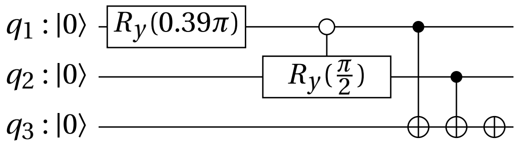


Figure 6.8: The quantum circuit to prepare f_W by UQSP$_{\text{FD}}$ algorithm.

6.6.4 Variable reordering methods

Variable reordering affects both extracting dependencies and quantum state preparation, e.g., the *AND* operation is not reversible such that variable reordering changes dependency extraction. Three variable reordering methods are evaluated to reduce the number of CNOTs:

1. *Exhaustive Reordering* preforms quantum state preparation for all permutations of the variables of the Boolean function. For each variable order, a quantum circuit is constructed. The quantum circuit with the smallest number of CNOTs is returned. Exhaustive reordering does not scale and is only practical for the Boolean functions of a few Boolean variables.
2. *Random Reordering* generates prior fixed numbers of different random permutations of the variables relying on an implemented pseudo-random number generation.

3. *Greedy Reordering* generates the set of all variable orders obtained by locally swapping two variables of the function. The algorithm evaluates the number of required CNOTs to synthesize the quantum circuit under the considered orders and repeats its task using the variable order with the lowest costs as the starting point. The algorithm proceeds until a local optimum is found and the number of CNOTs cannot be improved by swapping variables.

6.6.5 Results & discussion

I compare the proposed UQSP using dependency analysis methods with Qiskit [111] which implements the method presented by Iten et al. [101]. The UQSP_{FD} method is implemented in an open-source tool called *angel*, which is introduced in Chapter 9. The maximum level of optimization is specified in Qiskit to generate the circuit with the minimum number of CNOTs. All experiments are conducted on an Intel Core i7, 2.7 GHz with 16 GB memory. I show, by comparing to Qiskit, that UQSP_{FD}, specialized for uniform quantum states, is more scalable and can substantially reduce the number of CNOTs over methods for arbitrary state preparation. I examine the effect of using different dependency analysis and variable reordering methods and construct a trade-off between them. Finally, I select my UQSP_{FD} as an improved algorithm and compare my results with the results extracted from QisKit for the practical benchmarks.

Benchmarks

I present experiments for a large set of uniform quantum states including the well-known quantum states GHZ and W. I also evaluate my algorithm to prepare uniform quantum states required by the *Quantum Byzantine Agreement* (QBA) and *Quantum Coupon Collector* (QCC) problems. Moreover, as I map uniform quantum states to Boolean functions and I use some techniques from logic synthesis, I extract several Boolean functions from the EPFL and ISCAS benchmarks, which are frequently used in logic synthesis.

Using dependency analysis

I evaluate UQSP_{FD} in terms of runtime and the number of synthesized elementary quantum gates. I primarily focus on the number of CNOTs which are more expensive than single-qubit gates for NISQ quantum computers, but I also reduce rotation gates. I use the EPFL and ISCAS benchmarks to evaluate my dependency analysis and variable reordering methods to construct a good trade-off between them regarding runtime and the number of CNOTs. Next, I compare my final results with Qiskit for the practical benchmarks such as the GHZ state, W state, and uniform quantum states required by the QBA, and QCC with different k values (k-QCC).

Dependency analysis methods

I compare the general functional decomposition method, presented in Section 6.4, against UQSP_{FD} with two different functional dependency methods. I implemented all methods using

Table 6.3: Experimental results regarding different dependency analysis methods.

			Baseline [53]		Pattern search			SAT-based ESOP synthesis		
Bench	#Qs	#funcs	#CNOTs	Time (s)	#CNOTs	Improve	Time (s)	#CNOTs	Improve	Time (s)
EPFL	4	367	4272	0.01	3990	6.60%	0.02	3972	7.02%	0.03
EPFL	5	1954	47430	0.12	45086	4.94%	0.30	44714	5.73%	0.42
EPFL	6	6424	289036	0.72	276424	4.36%	2.62	272753	5.63%	3.17
EPFL	7	14334	1208777	3.62	1087473	10.04%	18.02	1065106	11.89%	21.96
EPFL	8	23904	4137028	10.50	3525308	14.79%	91.68	3488039	15.69%	91.09
ISCAS	4	225	2358	0.01	2266	3.90%	0.01	2248	4.66%	0.02
ISCAS	5	676	14441	0.04	13527	6.33%	0.09	13266	8.14%	0.18
ISCAS	6	1150	48655	0.12	44874	7.77%	0.44	43495	10.61%	0.88
ISCAS	7	1452	124255	0.32	101603	18.23%	1.72	97912	21.20%	3.36
ISCAS	8	1571	273233	0.50	210139	23.09%	4.91	203904	25.37%	7.42

the same *truth table* package to represent Boolean functions. Truth tables are an effective representation for Boolean functions of up to 16 variables. For larger states with more variables, symbolic representations such as decision diagrams have to be used.

Table 6.3 presents the results of the general functional decomposition method as the baseline, and UQSP$_{\text{FD}}$ with the proposed dependency analysis methods, pattern search, and SAT-based ESOP synthesis. For each benchmark (bench) the number of variables and number of functions are shown by #Qs and #funcs, respectively. The state preparation is done for one fixed variable order $1, \dots, n$. For each benchmark, the number of CNOTs and the runtime in seconds are shown accumulated over all functions. The CNOT reduction increases with the number of qubits. The number of CNOTs are reduced by up to 14.79% and 15.69% for the EPFL benchmarks and by up to 23.09% and 25.37% for the ISCAS benchmarks with pattern search and SAT-based ESOP synthesis, respectively. The comparison between pattern search and SAT-based ESOP synthesis shows that the ESOP-based approach reduces the number of CNOTs further at the cost of requiring more runtime. Selecting the strategy allows users to trade runtime for quality-of-results depending on the application scenario.

Variable reordering methods

I further examine the impact of the proposed variable reordering methods, *Exhaustive Reordering* (ER) which considers all $n!$ orders, *Random Reordering* (RR) with n^2 different randomly-chosen orders using a fixed random seed, and *Greedy Reordering* (GR) which dynamically reorders until no further local improvement is achieved, considering SAT-based ESOP as the dependency strategy.

The experimental results using ESOP-based dependency analysis with ER, RR, and GR are summarized in Table 6.4. For each reordering method, I show the number of CNOTs, the

Table 6.4: Experimental results regarding different variable reordering methods.

			ER with $n!$ orders			RR with n^2 orders			GR		
Bench	#Qs	#funcs	#CNOTs	Improve	Time (s)	#CNOTs	Improve	Time (s)	#CNOTs	Improve	Time (s)
EPFL	4	367	3646	14.65	0.44	3686	13.72%	0.25	3938	7.82%	0.11
EPFL	5	1954	39509	16.70	24.03	39895	15.89%	3.89	44247	6.71%	1.91
EPFL	6	6224	232679	19.50	953.33	237048	17.99%	45.46	270673	6.35%	16.73
EPFL	7	14334	730474	39.57	36258	790067	34.64%	325.38	1035187	14.36%	136.11
ISCAS	4	225	2064	12.47	0.30	2080	11.79%	0.17	2238	5.09%	0.07
ISCAS	5	676	12043	16.61	10.13	12104	16.18%	1.49	13248	8.26%	0.78
ISCAS	6	1150	38828	20.20	233.99	39294	19.24%	10.83	43226	11.16%	4.48
ISCAS	7	1452	72043	42.02	5170.97	76162	38.71%	42.55	94305	24.10%	22.41

relative improvement over the baseline, and the required runtime in seconds. Exhaustive reordering reaches the best result, but requires too much runtime and can only be applied to small functions in practice. Comparison of RR and GR shows that the CNOTs are reduced using RR and runtime is increased because the number of considered orders using RR are more than GR. But as CNOTs reduction is more important and runtime consumption using RR is still less, I select RR over GR. As a result, to set the best trade-off between the number of CNOTs and runtime, for a small number of qubits, I do preparation using ER and for a large number, using RR.

Comparison to Qiskit for the practical quantum states

The experimental results for the quantum state preparation of the practical quantum states are presented in Table 6.5. As benchmarks, the Boolean functions for the GHZ and W states as well as QBA and k-QCC are considered, over n qubits (#Qs). I consider the Boolean functions for different numbers of qubits in the range 8-18 qubits. My approach improves the number of CNOTs, rotation gates, and runtime over Qiskit. The proposed method converges to the optimum circuit for the GHZ state, whereas Qiskit uses a fixed precomputed optimal template for this state. Note that in Table 6.5, Qiskit's results come from applying their algorithm, not using a template. For the W state, my approach reduces CNOTs significantly whereas Qiskit's results are close to the upper bound of $2^n - 2$.

The QBA benchmark consists of a sequence of 1-bits in the beginning and a sequence of 0-bits for the rest in its truth table. This means one can divide qubits into three parts. For the first part, all bits are 1, and I can prepare them using only rotation gates without any control qubits. For the second part, there is both 1s and 0s, and I can utilize dependencies to reduce control qubits. For the third part, there is all 0s, which means the qubits will be in zero state and I do not need to add any gate. Results in Table 6.5 show that my method can deal better over QisKit for this benchmark. For example, my method prepares QBA with 16 qubits (QBA(16)) only with 12 rotation gates whereas QisKit requires 180 CNOTs. From Eq. 6.5, QBA(16) consists of

Table 6.5: UQSP$_{\text{FD}}$ results in comparison to Qiskit results for the practical quantum states.

		UQSP$_{\text{FD}}$ (ESOP+RR)			Qiskit			Improve (%)		
Bench	#Qs	#CNOTs	#Rgs	Time (s)	#CNOTs	#Rgs	Time (s)	#CNOTs	#Rgs	Time (s)
GHZ	10	9	1	0.00	1013	1023	6.36	99.11	99.90	100.00
GHZ	12	11	1	0.00	4083	4094	25.63	99.73	99.98	100.00
GHZ	15	14	1	0.00	32752	32767	246.81	99.96	100.00	100.00
W	10	519	512	0.01	1013	1023	7.09	48.77	49.95	99.86
W	12	2057	2048	0.01	4083	4095	28.73	49.62	49.99	99.96
W	15	16396	16384	0.01	32752	32767	244.91	49.94	50.00	99.99
QBA	12	289	289	7.56	435	515	28.21	33.56	43.88	73.20
QBA	15	1438	1439	22.52	32620	32632	455.31	95.59	95.59	95.05
QBA	16	0	12	44.58	180	302	518.01	100.00	96.03	91.39
QBA	18	2343	2343	345.41	114643	114656	4138.01	97.96	97.96	91.65
QCC_{64}	8	127	128	0.15	247	255	2.25	48.58	49.80	93.33
QCC_{3996}	12	4094	4095	5.45	4063	4069	48.96	-0.76	-0.64	88.87
QCC_{1024}	14	16382	16383	13.14	16367	16380	205.62	-0.09	-0.02	93.61
QCC_{256}	16	32766	32768	40.7	65365	65248	912.32	49.87	49.78	95.54
total		76445	76404	479.54	309616	309826	6868.22	75.31	75.34	93.02

2^{12} 1-bits that shows I have all input assignments for the first 12 qubits. This shows that my method prepares QBA state efficiently.

For the k-QCC benchmark, I select 4 different benchmarks with different k values, randomly. As shown in the table, for some cases I reduce the number of CNOTs by exploiting dependencies between qubits. But for some other cases, for example 3996-QCC and 1024-QCC, there is not any dependencies between qubits and my results are close to the upper bound whereas the Qiskit's results are a little bit better. This is due to the fact that Qiskit uses post-optimization methods whereas I do not apply any post-optimization.

In total, UQSP$_{\text{FD}}$ reduces CNOTs, rotation gates, and runtime for quantum states of up to 18 qubits in average by 75.31%, 75.34%, and 93.02%, respectively.

6.7 Summary

To address the general problem of preparing quantum states, a Boolean method specialized for preparing uniform quantum states is proposed, called UQSP. Uniform quantum states are an important family of states that are frequently used in quantum algorithms, e.g., quantum cryptography. My method uses Shannon's decomposition and cofactoring. I implemented UQSP using both truth table and BDD representations as I can efficiently apply Shannon's decomposition to them. Using BDDs helps reducing runtime and prepare quantum states with larger number of qubits where state-of-the-art methods are not applicable.

To furthere reduce the number of CNOTs, I introduced the idea of identifying functional dependencies among qubits and used them to construct optimized quantum circuits. I added functional dependencies as an extension to my UQSP, called UQSP$_{\mathrm{FD}}$. I implemented UQSP$_{\mathrm{FD}}$ using truth tables. My state preparation method requires an exponential number of CNOTs in the worst case but it reduces CNOTs significantly for practical benchmarks. Moreover, my method generates an exact representation of quantum states without using ancilla qubits. I compared my algorithm with Qiskit. The comparison shown that my method is capable to reduce the average number of CNOTs by 75.31% for the practical benchmarks. The runtime is almost reduced by a factor of 2.

7 Cyclic Quantum State Preparation

Cyclic states, a family of quantum state, can be described as a uniform superposition of cyclic permutations on a basis state where the ones are located adjacently to each other. In this chapter, I propose an efficient algorithm for preparing cyclic states. I further present its circuit construction. The idea is based on creating cyclic permutations step-by-step. I design my algorithm such that creating each permutation requires only a constant number of 2-qubit and 3-qubit gates regardless of the total number of qubits n. Notably, the number of qubits required for creating each permutation is independent of n, since 2-qubit and 3-qubit gates require only constant numbers of elementary quantum gates. As a result, my algorithm requires only $O(n)$ elementary quantum gates.

7.1 Related works

As cyclic states are a special subset of uniform quantum states, one can use the methods presented in [53, 4]. The method described in [53] utilizes symbolic representations to reduce runtime and elementary quantum gates. In [4], dependencies between qubits are identified to reduce the number of elementary quantum gates. The experiments show that these methods can reduce the number of gates but they cannot provide a linear complexity for preparing cyclic states. The details on these methods are available in Chapter 6.

Authors in [2] present an efficient algorithm for preparing Dicke states, which are fully symmetric states and uses $O(kn)$ quantum gates. However, their method works only for cyclic states with $k = 1$ or $k = n - 1$, not any value of k. In [3], the authors show a method to prepare quantum states associated with graphs. The method prepares some specific subsets of Dicke states including cyclic states with $k = 2$. In contrast, I propose an algorithm for preparing cyclic states with arbitrary values of k ranging from 1 to n. Note that, in [3], it is claimed that the method can be extended to cyclic states with other values of k, but the extension is highly non-trivial as far as I can see. In addition, the complexity of preparing generic cyclic states using the method in [3] is unclear.

I compare my results with the results of [2, 3, 4]. Comparison over [4] shows that I reduce elementary quantum gates from exponential to linear complexity. While, my evaluation against [2, 3] shows that my circuit construction works for any value of k as well as gains a significant reduction of elementary quantum gates.

7.2 Cyclic states and their properties

Dicke states are an important family of n-partite (for $n \in \mathbb{N}^* = \{1, 2, 3, ...\}$) quantum states [116]. For example, their robustness against photon-loss noise makes them a desirable resource in building noise-resilient quantum sensors [117, 118]. Due to the importance of Dicke states, their preparation has been demonstrated experimentally in various settings [119, 120, 121], and quantum algorithms have been proposed to prepare them efficiently [2, 122, 123, 121].

Dicke states are fully symmetric, i.e., they are invariant under any permutation in the symmetric group $S(n)$ of n parties. Many realistic problems, nevertheless, only have partial symmetry rather than the full symmetry. Consider, for instance, a quantum network consisting of n nodes associated with a graph. A common type of tasks is to prepare a global (i.e. n-partite) quantum state such that each node is only entangled to a certain subset of nodes. Such tasks are growing more crucial as the quantum internet is being established [124, 125]. Dicke states obviously fail to achieve this goal unless in the special case where the network graph is complete.

In this chapter, I focus on cyclic states: a new family of partially symmetric multipartite states that are more versatile than Dicke states in network applications. Cyclic states are generated by performing the group of cyclic permutations $C(n)$ in coherent superpositions on a computational basis state $|1\rangle^{\otimes k}|0\rangle^{\otimes m}$ ($m := n - k$):

$$|C_k^n\rangle := \frac{1}{\sqrt{n}} \sum_{\pi \in C(n)} \pi\left(|1\rangle^{\otimes k}|0\rangle^{\otimes m}\right). \tag{7.1}$$

In other words, they are the uniform superpositions of computation basis states whose Hamming weights are equal and whose ones are adjacent.

Note that $|C_1^n\rangle$ coincides with the W state [47]. One also gets back the original definition of Dicke states by replacing $C(n)$ with the full symmetry group $S(n)$ in the above definition.

Cyclic states, as promised, have intriguing entanglement and coherence properties, which promise their applications in quantum internet [124] and quantum metrology [126]. Their arbitrary bipartite marginal state can be evaluated. Assuming w.o.l.g. $k \geq m$, straightforward calculation shows that the bipartite marginal state $\rho_{ij} := tr_{\overline{ij}}|C_k^n\rangle\langle C_k^n|$ is given by

$$\rho_{ij} = \begin{cases} \frac{k-1}{n}|11\rangle\langle 11| + \frac{m-1}{n}|00\rangle\langle 00| + \frac{1}{n}|01\rangle\langle 01| + \frac{1}{n}|10\rangle\langle 10| \quad , \\ \qquad\qquad\qquad\qquad\qquad \text{if } |i-j| \neq m, |i-j| \neq k \\ \\ \frac{2}{n}|\Psi^+\rangle\langle\Psi^+| + \frac{m-1}{n}(|01\rangle\langle 01| + |10\rangle\langle 10|) + \frac{k-m}{n}|11\rangle\langle 11| \quad , \\ \qquad\qquad\qquad\qquad\qquad \text{if } |i-j| = m \text{ or } k \end{cases} \tag{7.2}$$

where $|\Psi^+\rangle := (|01\rangle + |10\rangle)/\sqrt{2}$. One can immediately see that any two nodes are entangled if and only if they are separated by $m-1$ nodes (as the entanglement of formation [127] is non-zero). This is in contrast to Dicke states, for which there is the same amount of entanglement between any two subsystems. In more details, by the Hashing inequality [128, 129], one can lower bound the distillable entanglement E_D of the marginal state. For the case when $|i-j| = m = k$, the distillable entanglement [130, 131] satisfies

$$E_D \geq \left(\frac{m+1}{2m}\right)\log_2\left(\frac{m+1}{m}\right) - \left(\frac{m-1}{2m}\right)\log_2\left(\frac{m}{m-1}\right) > 0. \tag{7.3}$$

The above implies that if n nodes in a quantum network share multiple copies of a cyclic state, the relevant nodes (those whose distance is either m or k) can distill Bell states via local operation and classical communication. This means that each node in the network can establish secure quantum communication with one certain node in the network, while sharing only classical correlation with others. Such a property can be used in quantum communication tasks such as network routing [132, 133].

The above property of cyclic states also has useful applications in quantum metrology [126] or, more precisely, multipartite phase estimation problems. Imagine that a cyclic state $|C_k^n\rangle$ is, again, shared by n individual nodes in a quantum network. The j-th node passes its qubit through a phase gate $P_j := e^{i\phi_j}|0\rangle\langle 0| + e^{-i\phi_j}|1\rangle\langle 1|$ with an unknown phase ϕ_j, resulting in the global state $|C_k^n(\vec{\phi})\rangle := \left(\otimes_{j=1}^n P_j\right)|C_k^n\rangle$. The goal is for arbitrary two nodes to jointly measure their phase difference $\delta_{ij} := \phi_i - \phi_j$. Then, two nodes can jointly estimate δ_{ij} if and only if either $|i-j|$ equals m or k. Otherwise, they cannot extract any useful information since the marginal state is independent of δ_{ij}.

Given the above desired features, it is therefore meaningful to consider how to *prepare cyclic states efficiently.*

7.3 Proposed method

As mentioned before, a *cyclic state* is defined as Eq. 7.1.

Example 7.3.1. *The cyclic state $|C_3^5\rangle$ is represented by*

$$|C_3^5\rangle = \frac{1}{\sqrt{5}}(|11100\rangle + |01110\rangle + |00111\rangle + |10011\rangle + |11001\rangle). \tag{7.4}$$

In this section, I propose an algorithm to prepare cyclic states deterministically. Next, I propose a quantum circuit construction and a pseudo code for my algorithm. Moreover, a proof of correctness is presented.

7.3.1 Cyclic state preparation algorithm

To prepare cyclic states, I design a unitary operator $C_{n,k}$, which takes as input the classical state $|1\rangle^{\otimes k}|0\rangle^{\otimes m}$ and generates the cyclic state $|C_k^n\rangle$

$$C_{n,k}(|1\rangle^{\otimes k}|0\rangle^{\otimes m}) = |C_k^n\rangle. \tag{7.5}$$

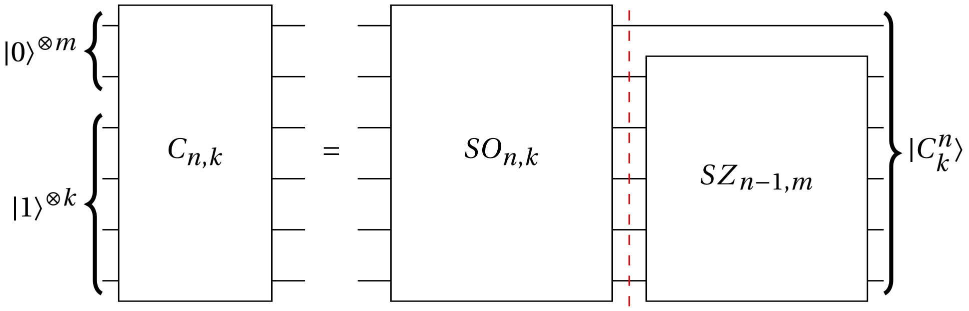


Figure 7.1: General structure of cyclic state preparation algorithm.

From the definition of cyclic states in Eq. 7.1, and by considering the Example 7.3.1, it is obvious that cyclic states are the superposition of two types of basis states. For the first type, all the ones are together in the binary string of the basis states. In the second, ones are split in the beginning and the end of the string. Hence, I divide the preparation into two parts. First, I apply a block to Shift Ones (SO). This block generates all basis states of the first type and results $|0\rangle^{\otimes m}|1\rangle^{\otimes k}$ in the end that goes to the second block as input. Second, I create the basis states of the second type. They are generated by circular shifting of $|0\rangle^{\otimes m}|1\rangle^{\otimes k}$ that corresponds to Shift Zeros (SZ). To keep ones in the beginning and in the end of the string, and to avoid creating repetitious basis states, I shift zeros one less time. Hence, I apply SZ block on the last $n-1$ qubits. The general construction of my algorithm is depicted in Fig. 7.1.

7.3.2 Cyclic state circuit construction

In this section, I propose the detailed structure of SO and SZ blocks in creating all desired basis states. I further present a pseudo code of my algorithm.

Explicit construction of $SO_{o,k}$

In the following, I describe a construction of shifting ones unitary $SO_{o,k}$. In this notation o shows the total number of qubits entering the subroutine. The Fig. 7.2 shows its structure. First, I apply a block on the last $k+1$ qubits to shift k ones one position to the right called $ShiftOnes(o,k)$. Next, I apply same procedure iteratively on the first $o-1$ qubits that is shown by $SO_{o-1,k}$.

***ShiftOnes*(*o, k*) Building Block.** I need to transform $|1\rangle^{\otimes k}|0\rangle$ to $|0\rangle|1\rangle^{\otimes k}$. In this regard, I design a quantum circuit that maps:

$$|1\rangle^{\otimes k}|0\rangle \rightarrow \sqrt{\frac{1}{o}}|1\rangle^{\otimes k}|0\rangle + \sqrt{\frac{o-1}{o}}|0\rangle|1\rangle^{\otimes k}. \tag{7.6}$$

This transformation is constructed by a 1-controlled $R_y(2cos^{-1}\sqrt{\frac{1}{o}})$ which maps $|0\rangle \rightarrow \sqrt{\frac{1}{o}}|0\rangle + \sqrt{\frac{o-1}{o}}|1\rangle$, and a CNOT, that is shown in Fig. 7.3. On one hand, as there are k successive ones, I only need to check one of them. On the other hand, I know q_{o-k+1} is modified in the previous

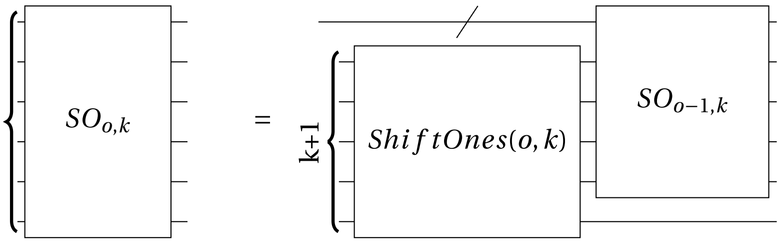


Figure 7.2: The construction of SO block iteratively.

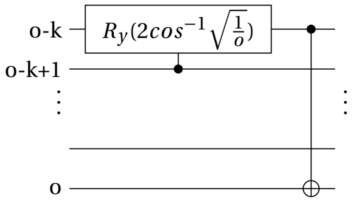


Figure 7.3: The circuit implementation of $ShiftOnes(o,k)$.

step for creating previous permutation. Hence, I select q_{o-k+1} as the control-qubit for the 1-controlled R_y gate to create the current permutation. Then, I apply a CNOT to convert last qubit to $|0\rangle$ state.

Explicit construction of $SZ_{z,m}$

Here, a construction of shifting zeros unitary $SZ_{z,m}$ is described. Its structure is shown in the Fig. 7.4. Here, z shows the number of input to the block. First, $ShiftZeros$ block is applied on the last $m+1$ qubits to shift m zeros one position to the right. Next, I iteratively apply SZ block on the first $z-1$ qubits that is shown by $SZ_{z-1,m}$.

***ShiftZeros*(*z, m*) Building Block.** This time, I need to transform $|0\rangle^{\otimes m}|1\rangle$ to $|1\rangle|0\rangle^{\otimes m}$. Hence, I design a quantum circuit to transform

$$|0\rangle^{\otimes m}|1\rangle \to \sqrt{\frac{1}{z-m+1}}|0\rangle^{\otimes m}|1\rangle + \sqrt{\frac{z-m}{z-m+1}}|1\rangle|0\rangle^{\otimes m}. \tag{7.7}$$

Note that the first part (SO) consists of m $ShiftOnes$ blocks. I consider its effect in adjusting amplitudes in Eq. 7.7. This transformation is constructed by two CNOTs and a 2-controlled $R_y(-2cos^{-1}\sqrt{\frac{1}{z-m+1}})$ mapping $|1\rangle \to \sqrt{\frac{1}{z-m+1}}|1\rangle + \sqrt{\frac{z-m}{z-m+1}}|0\rangle$, in between, with negative controls that is shown in Fig. 7.5. Here, to transform q_{z-m} into $|0\rangle$, all previous m qubits should be $|0\rangle$, but in practice it is not essential to check all the m qubits. In fact, I find that it is enough to check q_{z-1} and q_{z-m+1} to guarantee proper functionality.

Algorithm 7.1 shows the pseudo-code of my deterministic algorithm for preparing the cyclic states that I have already explained its detail.

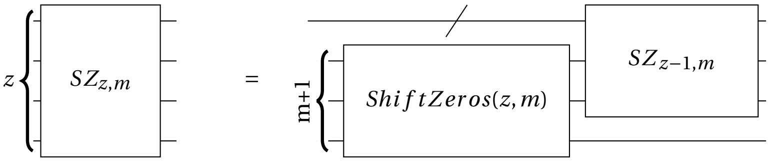


Figure 7.4: Construction of SZ block iteratively.

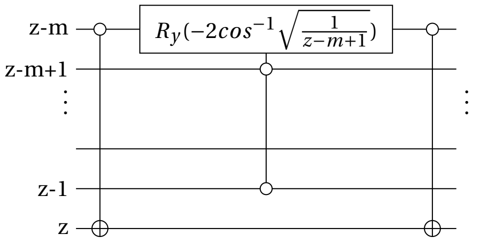


Figure 7.5: The circuit implementation of $ShiftZeros(z,m)$.

Algorithm 7.1: Deterministic Preparation of Cyclic States.

```
Input: The number of qubits $n$, The number of ones $k$
Output: The quantum circuit $qc$
Proc $C(n, k)$:
    $qc$ = CreateAndInitializeQC($n$, $k$)
    ApplySO($qc$, 1, $n$, $k$)
    ApplySZ($qc$, 2, $n$, $n - k$)
    return $qc$

Proc $CreateAndInitializeQC(n, k)$:
    $qc$ = CreateNewQC()
    for $i = 1,\dots,n$ do
        $qc$.CreateQubit($i$)
        if $i > n - k$ then
            $qc$.ApplyNOT($i$)
    return $qc$

Proc $ApplySO(qc, q_{start}, q_{end}, k)$:
    $o = q_{end}$ - $q_{start}$ + 1
    if $o \leq k$ then
        return
    $qc$.ApplyShiftOnes($o$, $k$)
    return ApplySO($qc$, $q_{start}$, $q_{end} - 1$, $k$)

Proc $ApplySZ(qc, q_{start}, q_{end}, m)$:
    $z = q_{end}$ - $q_{start}$ + 1
    if $z \leq m$ then
        return
    $qc$.ApplyShiftZeros($z$, $m$)
    return ApplySZ($qc$, $q_{start}$, $q_{end} - 1$, $m$)
```

7.3.3 Proof of correctness

Here I show that the circuit indeed prepares the cyclic state as desired. To this purpose, I first define two families of intermediate states:

$$|\psi_l\rangle := \sqrt{\frac{1}{n}}\sum_{i=0}^{l-1}|0\rangle^{\otimes i}|1\rangle^{\otimes k}|0\rangle^{\otimes m-i} + \sqrt{\frac{n-l}{n}}|0\rangle^{\otimes l}|1\rangle^{\otimes k}|0\rangle^{\otimes m-l} \tag{7.8}$$

and

$$\begin{aligned}|\phi_l\rangle := \sqrt{\frac{1}{n}}\sum_{i=0}^{m-1}|0\rangle^{\otimes i}|1\rangle^{\otimes k}|0\rangle^{\otimes m-i} + \sqrt{\frac{1}{n}}\sum_{j=0}^{l-1}|1\rangle^{\otimes j}|0\rangle^{\otimes m}|1\rangle^{\otimes k-j}\\ +\sqrt{\frac{n-m-l}{n}}|1\rangle^{\otimes l}|0\rangle^{\otimes m}|1\rangle^{\otimes k-l}.\end{aligned} \tag{7.9}$$

By definition, the initial state is $|\psi_0\rangle = |1\rangle^{\otimes k}|0\rangle^{\otimes m}$. First, I analyse the action of $SO(n,k)$ on the input state. In the process of preparing cyclic states, any input to the shift one operation is in a superposition of qubit strings of the form $|0\rangle^{\otimes i}|1\rangle^{\otimes k}|0\rangle^{\otimes m-i}$. Notice that, according to Fig. 7.3, $ShiftOnes(n-l,k)$ acts non-trivially only if the $(m-l)$-th qubit is $|1\rangle$, and

$$ShiftOnes(n-l,k)|0\rangle^{\otimes i}|1\rangle^{\otimes k}|0\rangle^{\otimes m-i} = |0\rangle^{\otimes i}|1\rangle^{\otimes k}|0\rangle^{\otimes m-i} \qquad i<l, \tag{7.10}$$

and

$$\begin{aligned}ShiftOnes(n-l,k)|0\rangle^{\otimes l}|1\rangle^{\otimes k}|0\rangle^{\otimes m-l} = \sqrt{\frac{n-l-1}{n-l}}|0\rangle^{\otimes l}|1\rangle^{\otimes k}|0\rangle^{\otimes m-l}\\ +\sqrt{\frac{1}{n-l}}|0\rangle^{\otimes l+1}|1\rangle^{\otimes k}|0\rangle^{\otimes m-l-1}.\end{aligned} \tag{7.11}$$

Substituting into Eq. 7.8, I have

$$ShiftOnes(n-l,k)|\psi_l\rangle = |\psi_{l+1}\rangle \tag{7.12}$$

and thus

$$SO(n,k)|\psi_0\rangle = |\psi_m\rangle. \tag{7.13}$$

It remains to be shown that $SZ(n,m)|\psi_m\rangle = |C^n_k\rangle$. By Eqs. 7.8 and 7.9, I reach to $|\psi_m\rangle = |\phi_0\rangle$. Next, notice that

$$\begin{aligned}&ShiftZeros(n-l,m)|1\rangle^{\otimes l}|0\rangle^{\otimes m}|1\rangle^{\otimes k-l} =\\ &\sqrt{\frac{n-l-m}{n-l-m+1}}|1\rangle^{\otimes l}|0\rangle^{\otimes m}|1\rangle^{\otimes k-l} + \sqrt{\frac{1}{n-l-m+1}}|1\rangle^{\otimes l+1}|0\rangle^{\otimes m}|1\rangle^{\otimes k-l-1}\end{aligned} \tag{7.14}$$

and

$$ShiftZeros(n-l,m)|1\rangle^{\otimes j}|0\rangle^{\otimes m}|1\rangle^{\otimes k-j} = |1\rangle^{\otimes j}|0\rangle^{\otimes m}|1\rangle^{\otimes k-j} \qquad j < l. \tag{7.15}$$

In addition, since I assumed $m \leq k$, I also get

$$ShiftZeros(n-l,m)|0\rangle^{\otimes i}|1\rangle^{\otimes k}|0\rangle^{\otimes m-i} = |0\rangle^{\otimes i}|1\rangle^{\otimes k}|0\rangle^{\otimes m-i}. \tag{7.16}$$

Therefore, substituting both into Eq. 7.9 I get

$$ShiftZeros(n-l,m)|\phi_{l-1}\rangle = |\phi_l\rangle SZ(n-1,m)|\phi_0\rangle = |\phi_{k-1}\rangle. \tag{7.17}$$

The proof is concluded since by definition in Eq. 7.9 $|\phi_{k-1}\rangle = |C^n_k\rangle$.

7.4 Results & evaluation

I compute the number of elementary quantum gates {CNOT, R_y, NOT} of my circuit construction. I assume $k \geq m$, otherwise I only need to add $2n$ extra NOTs at the beginning and end of the quantum circuit.

Shifting ones consists of m $ShiftOnes$ blocks. Considering the quantum circuit depicted in Fig. 7.3, m 1-controlled R_y and m CNOT gates are required.

Shifting zeros consists of $k-1$ $ShiftZeros$ blocks. As shown in Fig. 7.5, this part requires $k-1$ 2-controlled R_y gates except when $m=1$ or $m=2$ which it requires $k-1$ 1-controlled R_y gates, instead. It also requires $2(k-1)$ CNOT gates. Moreover, this part requires NOT gates to apply negative controls. Each $ShiftZeros$ block consists of 8 NOTs but at least one of them can be canceled by the next block. Hence, in between blocks require 6 NOTs and the total number of NOTs is upper bounded by $6(k-1)+2$. Note that when $m=1$ or $m=2$, as $ShiftZeros$ blocks are constructed by 1-controlled R_y gates, $6(k-1)$ NOTs are required.

To create elementary quantum gates, I decompose 1-controlled R_y and 2-controlled R_y gates into {2 CNOTs, 2 R_y gates}, and {4 CNOTs, 4 R_y gates}, respectively. As a result, the cost function regarding the number of CNOT gates (Cost$_c$), R_y gates (Cost$_r$), and NOT gates (Cost$_n$) are formulated as follows:

$$\text{Cost}_c(n,k) = \begin{cases} 7 & \text{if } k=2, n=3 \\ 4n-5 & \text{if } k=n-1, n>3 \\ 4n-6 & \text{if } k=n-2 \\ 3n+3k-6 & \text{otherwise} \end{cases}, \tag{7.18}$$

$$\text{Cost}_r(n,k) = \begin{cases} 4 & \text{if } k=2, n=3 \\ 2n-2 & \text{if } k=n-1, n>3 \\ 2n-2 & \text{if } k=n-2 \\ 2n+2k+4 & \text{otherwise} \end{cases}, \tag{7.19}$$

Table 7.1: Proposed method comparison over methods in [2, 3].

	#CNOT	#Ry
[3] $k = 2$	$\frac{7}{2}n+3$	$4n+2$
Proposed Method	$4n-6$	$2n-2$
[2] $k = 1$	$>5n$	$>4n$
Proposed Method	$4n-5$	$2n-2$
[2] $k = n-1$	$>5n$	$>4n$
Proposed Method	$4n-5$	$2n-2$

and

$$\mathrm{Cost}_n(n,k) = \begin{cases} 6 & \text{if } k = 2, n = 3 \\ 6n-12 & \text{if } k = n-1, n > 3 \\ 6n-18 & \text{if } k = n-2 \\ 6k-4 & \text{otherwise} \end{cases}. \tag{7.20}$$

I compare my approach with state-of-the-art methods [2, 3] that can not prepare cyclic states with arbitrary k. I compare the results in terms of the number of CNOTs that are more expensive than single-qubit gates in NISQ architectures. I also show the number of R_y gates. The results are shown in Table 7.1. To compare with [3], k should be only 2. As I assume $k \geq m$, it corresponds to $k = n-2$ with $2n$ extra NOTs. Considering Eqs. 7.18, 7.19, and 7.20, my method generates $4n-6$ CNOTs, $2n-2$ R_y, and $8n-18$ NOT gates. Results show that for large n, the method in [3] works a little better while it is limited to k to be only 2. To compare with Dicke state preparation method introduced in [2], I can only consider some specific Dicke states $|D_k^n\rangle$ that are equal to a cyclic state $|C_k^n\rangle$. From their definitions, they are equal when either $k = 1$ or $k = n-1$. As shown in the table, my method reduces both CNOT and R_y gates significantly. Hence, my circuit construction is more general (no constraints on k) and effective as compared to methods in [2, 3].

In addition, to compare for arbitrary k, I compare my method with uniform quantum state preparation method in [4]. The results regarding the number of CNOTs are presented in Table 7.2, for different number of n and k. The results show that my method reduces the number of CNOTs by a factor of 2. When the number of qubits (n) is small, for some cases (here for $n = 10$ and $k = 5$), my results are worse. This is due to the fact that, for these cases, there exist good dependencies between qubits which are utilized by [4] to reduce CNOTs. In the case that the number of qubits (n) is large, I reduce CNOTs with linear complexity, for all values of k . Conversely, [4] increases the number of CNOTs exponentially.

7.5 Summary

Efficient quantum state preparation is a crucial step to design quantum computing systems. In this chapter, I proposed a construction method as well as quantum circuits that deterministically generate cyclic states. My circuit construction method uses $6n-9$ CNOT gates in the worst case when $k = n-1$, which is significantly better than the state-of-the-art methods. I further provided experimental results that confirm this analysis.

Table 7.2: Comparing the number of CNOTs for the proposed method and the preparation method in [4].

n	k	[4]	Proposed Method	Improve (%)
10	1	519	35	93.3
	2	107	34	68.2
	5	35	39	-11.4
	7	72	45	37.5
12	1	2057	43	97.9
	4	262	42	84.1
	7	86	51	40.7
	10	300	42	86.0
15	1	16396	55	99.7
	2	1089	54	95.0
	5	1034	54	94.8
	7	282	60	78.7
17	1	65550	63	99.9
	2	2161	62	97.1
	5	570	60	89.4
	8	542	69	87.3
19	1	262160	71	99.9
	2	8304	70	99.1
	7	1058	72	93.2
	9	1058	78	92.6

8 Sparse Quantum State Preparation

In contrast to general quantum states, in most quantum computational tasks, the states to prepare are from subfamilies of n-qubit states, such as uniform quantum states [53, 4], Dicke states [2], and cyclic quantum states [54]. In these examples, all state subfamilies have classical descriptions with symmetric structures, which hints at the possibility of utilizing structured classical descriptions of quantum states to achieve efficient QSP. Here I exploit this possibility and propose a novel QSP algorithm for quantum states represented by reduced ordered decision diagrams. Decision diagrams are directed acyclic graphs over a set of Boolean variables and a non-empty terminal set with exactly one *root* node [79]. Decision diagrams avoid redundancies and lead to a more compact representation of logic functions.

8.1 Introduction

In this chapter, I consider the preparation of n-qubit quantum states $|\varphi\rangle = \sum_{s\in S}\alpha_s|s\rangle$, i.e., finding a unitary circuit U that consists of elementary quantum gates such that $U|0\rangle^{\otimes n} = |\varphi\rangle$. Here the *index set* $S \subset \{0,1\}^n$ contains every binary string s such that the amplitude α_s of $|s\rangle$ is non-zero, and $\sum_{s\in S}|\alpha_s|^2 = 1$. Without loss of generality, I assume that basis states in S are sorted in descending order. For two arbitrary n-bit strings s and s', there is a natural order $s \succ s'$ if s is no smaller than s' when both are regarded as binary numbers. In this way, I can order the elements of S as $s_1 \succ s_2 \succ \cdots \succ s_m$ and express the state to prepare as

$$|\varphi\rangle = \sum_{i=1}^{m} \alpha_{s_i}|s_i\rangle. \tag{8.1}$$

I use decision diagrams to represent the state in Eq. (8.1), where each basis state $|s_i\rangle$ and each amplitude α_{s_i} are represented by a path and a terminal node, respectively. I propose an efficient algorithm that prepares an arbitrary quantum state given its associated decision diagrams. The cost of my algorithm is $O(kn)$, where k is the number of paths in the decision diagram. Since k is always upper bounded by (and can be much smaller than) m, the number of non-zero amplitudes of the state in the computational basis, my algorithm efficiently prepares any *sparse state* with $m \ll 2^n$. Sparse quantum states have many applications for example in quantum linear system solvers [24], quantum Byzantine agreement algorithm [109] for large

n, and quantum machine learning [22]. Besides, many problems in classical computing are sparse such as sparse (hyper) graph problems [134]. To solve them using a quantum computer, I need to prepare their associated sparse quantum states. In addition, my algorithm can also efficiently prepare states with sparse decision diagrams ($k \ll 2^n$), even if the states themselves are not sparse ($m = \Omega(2^n)$).

Several algorithms have been proposed for sparse quantum state preparation [135, 136, 1] with $O(mn)$ cost. In all of them, the idea is based on preparing basis states one-by-one by applying several CNOTs and one multiple-controlled single-target gate. Tiago et al. [135] use one ancilla qubit to avoid disturbing prepared basis states while working on the others. Compared to [136], their results show that their algorithm performs well when the number of 1 bits in binary bit string representation of each basis state is almost 20%, which is a limitation. Emanuel et al. [136] propose an algorithm to prepare sparse isometries which include sparse states as well. Niels et al. [1] propose an algorithm that works in the opposite direction, i.e., they try to apply some gates to obtain $|0\rangle^{\otimes n}$ state from the desired sparse state. They repeat the same procedure in m iterations. In every iteration, they select two basis states and merge them into one by applying several CNOTs and one multiple-controlled single-target gate. Comparing methods in [136] and [1], they both perform well with small m, and their circuit costs are almost the same. However, the idea in [1] is simpler and its classical runtime, which is $O(nm\log_2(m))$, is less than that of the algorithm in [136], which is $O(\binom{n}{\log_2(m)} + nm^2)$. Hence, I regard [1] as the state of the art and compare my results to it.

Numerical experiments show that my algorithm outperforms the state of the art [1]. Depending on the sparsity m, my algorithm achieves an up to 31.85% reduction of the CNOT cost. The algorithm works very well for the states with sparse decision diagram representations, and uses up to 99.97% fewer CNOTs. In addition, my algorithm requires only one ancilla qubit, in stark contrast to many existing works [104, 112, 137] with ancilla qubit that grow with n.

8.2 Proposed representation of quantum states using ADDs

My algorithm works efficiently by making use of a data structure named decision diagram (DD). The detailed description on DDs is presented in Section 2.2.6. Here I present that how DDs can be used to represent quantum states.

Quantum states represented by DDs. Rather straightforwardly, an arbitrary n-qubit quantum state $|\varphi\rangle = \sum_{s\in S} \alpha_s|s\rangle$ can be represented by a decision diagram: for any $s \in S$, represent s by a path in the tree and set its internal nodes to the qubit registers $q_1, q_2, \ldots, q_n$, its edges to solid or dashed lines depending on the state of the registers, and its terminal node to α_s. I then simplify the decision tree by removing all the paths corresponding to $s \notin S$ and terminal nodes whose values are zero. Next, I further apply the reduction rules to get a ROADD (called ADD for short). When the state is uniform, i.e., all the amplitudes are equal, the ADD can be simplified to a BDD, where a terminal node with the binary value 1 indicates that the associated paths have non-zero amplitudes. Each path p of the reduced DD corresponds to one or more basis states $s \in S_p$, which is a subset of S. Denoting by P the set of the paths of the reduced DD, the

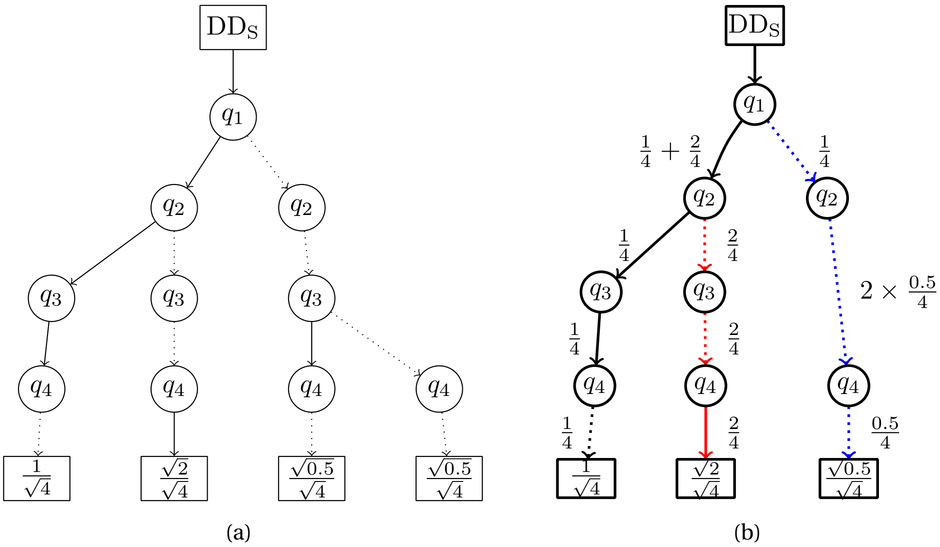


Figure 8.1: Decision diagram representation of the quantum state in the Example 8.2.1. (a) Before applying reduction rules. (b) After applying reduction rules.

state to prepare can be recast in the form:

$$|\varphi\rangle = \sum_{p\in P}\sum_{s\in S_p}\alpha_s|s\rangle. \tag{8.2}$$

Notice that all basis states $s \in S_p$ have the same amplitude.

Example 8.2.1. *The 4-qubit state*

$$|\varphi\rangle = \frac{1}{\sqrt{4}}(|1110\rangle+\sqrt{2}|1001\rangle+\sqrt{0.5}|0010\rangle+\sqrt{0.5}|0000\rangle) \tag{8.3}$$

has index set $S=\{1110,1001,0010,0000\}$ and non-zero amplitudes $\{\frac{1}{\sqrt{4}},\frac{\sqrt{2}}{\sqrt{4}},\frac{\sqrt{0.5}}{\sqrt{4}},\frac{\sqrt{0.5}}{\sqrt{4}}\}$. It can be represented by the decision diagram in Fig. 8.1.a. I represent each $s\in S$ with a binary string of qubits $q_1q_2q_3q_4$ where q_1, q_2, q_3, and q_4 are internal nodes. Each path shows a basis state s, and the terminal node connecting to each path shows its corresponding amplitude. For example, $\{s_1=1110,\alpha=\frac{1}{\sqrt{4}}\}$ expresses that I have a path in which $\{q_1=1,q_2=1,q_3=1,q_4=0\}$ that connects to the terminal node $\frac{1}{\sqrt{4}}$. Further noticing that on the right-side of the diagram (Fig. 8.1.a), two terminal nodes are equal which results in merging them. Furthermore, both left and right sub-graphs of q_3 are equal, so this node can be eliminated. Therefore, the decision diagram can be reduced to the ADD in Fig. 8.1.b which contains 3 paths instead of 4. Actually, the last two basis states $s_3=0010$ and $s_4=0000$ correspond to the same path $\{q_1=0,q_2=0,q_4=0\}$.

8.3 Proposed algorithm

In this section, I present my DD-based algorithm for quantum state preparation. I assume that the quantum state to prepare is represented by either an ADD or a BDD (when it is uniform). The use of DDs can aid in obtaining a quantum state that is free of redundancies, resulting in reduced circuit costs.

The algorithm works by preparing the paths in a DD one-by-one. For any n-qubit quantum state to prepare, my algorithm uses only one additional qubit q_A as an ancilla, whose value is tagged $|\text{yes}\rangle$ (regarded as $|0\rangle$ when used as a control qubit) or $|\text{no}\rangle$ (regarded as $|1\rangle$ when used as a control qubit). Intuitively, q_A serves as an indicator for whether a path has been created in the course of the state preparation. Paths that have been created are marked by $q_A \mapsto |\text{yes}\rangle$ and, by using q_A as control, I can avoid disturbing the created paths when creating a new path.

Each target-qubit in my quantum state preparation, transform $|0\rangle$ to a superposition of $\alpha|0\rangle + \beta|1\rangle$, where $|\alpha|^2$ ($|\beta|^2$) shows the probability of being zero (one) after measurement. To achieve this transformation, for some nodes, I need to apply a gate, called G, which is explained later. Therefore, I traverse the DD twice: 1) to compute the G gate for each node, and 2) to prepare the quantum state.

Post-order traversal to compute G gates. I traverse DD in post-order traversal (i.e. visiting one-child, zero-child, and parent nodes). For each node, I compute the probability of being one or zero from its corresponding one-child and zero-child. To compute zero probability (called p_0), for each node, I compute its portion from one-child (called t_1) and zero-child (called t_0) and then it equals to

$$p_0 = \frac{t_0}{t_1 + t_0}. \tag{8.4}$$

As an example, consider the state in Fig. 8.1.b, post-order traversal results in first visiting q_4 in the left-side. The portion from one-child is 0 and from zero-child is $|\frac{1}{\sqrt{4}}|^2$ (as it is amplitude I need to square it). Hence, the probability of being zero equals to $\frac{1/4}{0+1/4} = 1$. Next, I go through the upper node q_3, the portion from one-child comes from the summation of one-child and zero-child portions of q_4 which is $0 + \frac{1}{4}$. The portion from zero-child is 0 and the zero probability is $\frac{0}{1/4+0} = 0$. By continuing this procedure I obtain t_1 and t_0 which are written in the figure on the edges. Note that I need to consider the effect of eliminated nodes. If e nodes are eliminated along an edge, the portion is multiplied by 2^e. For example, in the right-side of the Fig. 8.1.b, on the zero-child of q_2, one node (q_3) is removed which results in $t_0 = 2^1 \times \frac{0.5}{4}$. Finally, α and β for G gates are computed by $\sqrt{p_0}$ and $\sqrt{1 - p_0}$, respectively which I show it as

$$G(p_0)|0\rangle = \sqrt{p_0}|0\rangle + \sqrt{1 - p_0}|1\rangle. \tag{8.5}$$

The above $G(p_0)$ can be implemented as a Pauli-y rotation: $G(p_0) = R_y(2cos^{-1}(\sqrt{p_0}))$.

Pre-order traversal to prepare the quantum state. The algorithm begins with an empty quantum circuit and all qubits initiated as:

$$|no\rangle_{q_A} \otimes |0\rangle_{q_1} |0\rangle_{q_2} ...|0\rangle_{q_n}. \tag{8.6}$$

Starting from the root, the algorithm traverses the DD with pre-order traversal (i.e. visiting parent, one-child, and zero-child nodes). To accomplish the traversal, I need to define a pointer *current_node* that points to the current node I are working on. To navigate through the DD, I define functions *one_child* and *zero_child* which return child of the current node regarding solid and dotted edges, respectively. While traversing through the DD, I compile the state preparation circuit according to the following rules:

1. **Preparation.** If the current node q is an internal node that is already on a path p_i, I do as follows.
   - If q is a *branching node*, which means it has both a zero-child and a one-child, I apply to the quantum circuit, a 2-controlled $G(p_0)$ gate [cf. Eq. (8.5)] on q with q_A and the last node on the path that has a one-child as control qubits, where the value of p_0 is determined by the post-order traversal. Otherwise, q either has a one-child or a zero-child. For the former case, I add a 2-controlled NOT gate on q with q_A and the last node on the path that has a one-child as control qubits. For the later case, I do nothing.
   - In addition, I need to consider the effect of reduced nodes between node q and its children. A node is reduced when both its one-child and zero-child point to the same thing. Hence, the qubit with half probability is zero and with half probability is one. If this is the case, I append to the quantum circuit, 2-controlled G($\frac{1}{2}$) gates on reduced nodes with q_A and the last node on the path that has a one-child as control qubits.
   - If q is the parent of the i-th terminal node, then I add a 2-controlled phase gate on q with q_A and the last node on the path that has a one-child as control qubits that adds a phase $e^{i\arg(\alpha_i)}$ to the path state $|s_i\rangle$.
2. **Computing the ancilla.** If the current node is a terminal node, it means that I have prepared the current path. Hence, I need to compute the ancilla qubit to mark that the current path is prepared. I append to the quantum circuit a multiple-controlled NOT gate on q_A with all qubits at branching nodes on path p_i being control.

This is a recursive traversal where I visit the current node, one-child and zero-child, respectively. In other words, I prepare paths from the largest (p_1) to the smallest (p_k). In this way, I can order the elements of P as $p_1 \succ p_2 \succ \cdots \succ p_k$. As a result, using this traversal I can prepare basis states in S from the largest (s_1) to the smallest (s_m). The pseudo-code of the proposed algorithm is shown in Algorithm 8.1. Note that, in the post-order traversal, I have already computed p_0 values of G gates corresponding to each node and here I pass it as an argument to the algorithm. Line 5 of Algorithm 8.1 shows the applying rule 2: Computing the ancilla, and lines 8, 11, 14, 16, 19, and 21 illustrate different conditions of rule1: Preparation. Additionally, I recursively visit one-child and zero-child in lines 18 and 23.

Fig. 8.2 shows the general structure of the output quantum circuit of my algorithm. Note that for preparing p_1, the ancilla qubit is not needed, because there is no other path prepared before p_1. Moreover, as p_k is the last path to prepare, I do not need to compute the ancilla qubit.

Algorithm 8.1: Deterministic Preparation of Quantum States using DD.

```
Input: DD representation of an $n$-qubit quantum state $|\varphi\rangle = \sum_{i=1}^{m} \alpha_{s_i} |s_i\rangle$, and $p_0$ values corresponding
    to each node of DD.
Output: The quantum circuit $qc$ that prepares the desired quantum state.
Create a quantum circuit $qc$ with $n+1$ qubits corresponding to $q_A q_1 q_2 ... q_n$.
I nitialize the $qc$ with $|1\rangle_{q_A} \otimes |0\rangle_{q_1} |0\rangle_{q_2} ... |0\rangle_{q_n}$.
I nitiate the pointer $current\_node$ as the root of DD.
PreOrder_traversal ($current\_node, qc, p_0\_values$) :
if $current\_node$ is a terminal node then
    Append to $qc$ a multiple-controlled NOT gate with the qubits of $branches$ being controls and $q_A$
      being the target.
    return
if $current\_node$ is a branching node then
    Append to $qc$ a 2-controlled gate $G(p_0)$ [cf. Eq. (8.5), with $p_0$ from $p_0\_values$ corresponding to
      $current\_node$] gate targeting the qubit corresponding to $current\_node$ controlled on ancilla
      qubit and the qubit corresponding to the last $|1\rangle$ in the path.
    Append the qubit corresponding to $current\_node$ and its value to $branches$.
else
    if $one\_child(current\_node) \neq nullptr$ then
        Append to $qc$ a 2-controlled NOT gate targeting the qubit corresponding to $current\_node$
          controlled on ancilla qubit and the qubit corresponding to the last $|1\rangle$ in the path.
if Some qubits are reduced between $current\_node$ and $one\_child(current\_node)$ then
    Append to $qc$ 2-controlled G($\frac{1}{2}$) gates targeting the reduced qubits with ancilla qubit and the qubit
      corresponding to the last $|1\rangle$ in the path as controls.
if $one\_child(current\_node)$ is a terminal node then
    Append to $qc$ a 2-controlled phase gate that adds a phase $e^{i\alpha}$ (corresponding to this path) targeting
      the qubit corresponding to $current\_node$ controlled on ancilla qubit and the qubit corresponding
      to the last $|1\rangle$ in the path.
PreOrder_traversal ($one\_child(current\_node), qc, p_0\_values$)
if Some qubits are reduced between $current\_node$ and $zero\_child(current\_node)$ then
    Append to $qc$ 2-controlled G($\frac{1}{2}$) gates targeting the reduced qubits with ancilla qubit and the qubit
      corresponding to the last $|1\rangle$ in the path as controls.
if $zero\_child(current\_node)$ is a terminal node then
    Append to $qc$ a 2-controlled phase gate that adds a phase $e^{i\alpha}$ (corresponding to this path) targeting
      the qubit corresponding to $current\_node$ controlled on ancilla qubit and the qubit corresponding
      to the last $|1\rangle$ in the path.
PreOrder_traversal ($zero\_child(current\_node), qc, p_0\_values$)
```

Example 8.3.1. *In this example, I show how to create a quantum circuit to prepare the state represented in Fig. 8.1.b. By employing pre-order traversal, I can navigate through the three paths represented by the colors black, red, and blue. To compute* p_0*, values of* t_0 *and* t_1 *are shown in the figure. Starting from the root, I need to append a* G($\frac{1}{4}$) *gate on* q_1 *that shows the probability of being zero for this qubit. Going through the black path* (p_1)*, on the next node* q_2 *there exists a branch which requires a 1-controlled* G($\frac{2}{3}$) *gate. This is the first basis state and I do not need to check the ancilla qubit. Next, on* q_3 *there is not any branch but it has a one-child, so it is required to append a CNOT gate with the last* $|1\rangle$ *in the path* (q_2) *as control. Next, for* q_4 *there is not any branch and there is only a zero-child that does not require any action. To compute the ancilla qubit, I need to add a multiple-controlled NOT gate on the ancilla qubit with 2 controls on branching nodes which are* $q_1 = 1$ *and* $q_2 = 1$.

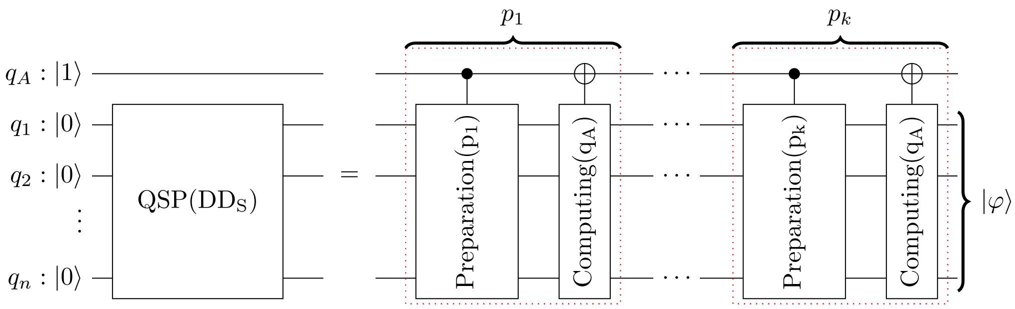


Figure 8.2: The general structure of the quantum circuit for QSP over DD$_S$.

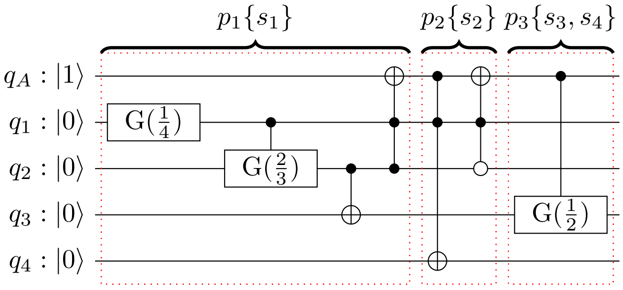


Figure 8.3: The generated quantum circuit for preparing the state presented as DD in Fig. 8.1.b.

Afterward, the traversal returns to q_2 and goes through the red path (p_2). It goes to q_3, there is not any branch and there is only a zero-child that does not require any action. Next, q_4 has a one-child and so I need to add a 2-controlled NOT gate on q_4 with ancilla and q_1 which is the last $|1\rangle$ in the path as control qubits. Then, to mark that p_2 is prepared, I add a 2-controlled NOT gate on the ancilla qubit with $q_1 = 1$ and $q_2 = 0$.

Finally, the algorithm goes back to the root again and traverses the blue path (p_3). q_2 has a zero-child and I do not need to add any gate for it. Next, the q_3 is removed which requires adding a G($\frac{1}{2}$) gate that shows with the half probability it is zero. There is not any last $|1\rangle$ in this path so it only has one control which is the ancilla. Then, q_4 has zero-child and again I do not need to add any gate for it. the reduced node q_3 enables the preparation of s_3 and s_4 together. This reduces the number of iterations and so circuit cost. Moreover, as this path corresponds to the last basis states s_3, s_4, I do not need to compute the ancilla qubit. Fig. 8.3 shows the generated quantum circuit.

8.4 Numerical experiments

In this section, I evaluate the proposed algorithm over the state of the art [1]. My algorithm is implemented in *angel*, which is introduced in Chapter 9. All experiments are conducted on an Intel Core i7, 2.7 GHz with 16 GB memory.

Random states. I evaluate my algorithm on randomly generated states with different amplitudes. The parameter m denotes the number of basis states with non-zero amplitudes. I

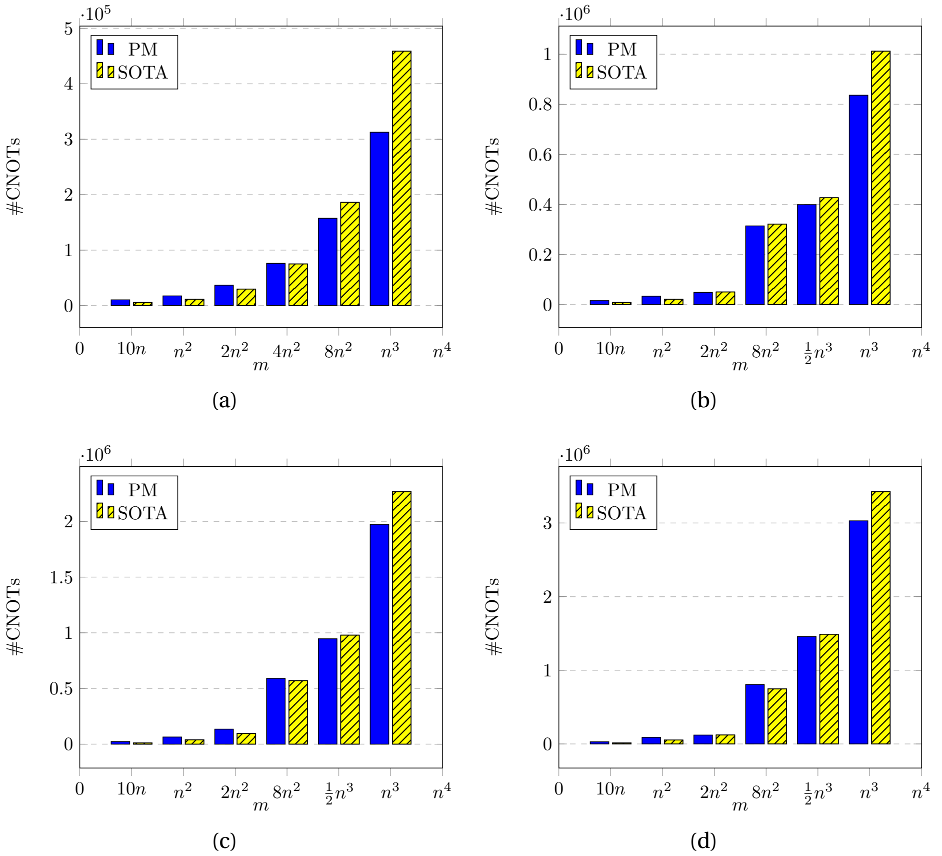


Figure 8.4: **Comparison between the CNOT complexities of my proposed method (PM) and the state-of-the-art (SOTA) method.** My PM is compared to the best-known algorithm (STOA) in [1] on random sparse states of n qubits. For different n, I plot the number of CNOTs required in both algorithms as a function of m, the number of non-zero amplitudes. It can be seen that PM requires fewer CNOTs in the interval between $2n^2$ and n^3 for $n = 16, 20$, and $8n^2$ and n^3 for $n = 25, 28$. Moreover, the more increasing of m results in the more reduction of CNOTs. (a) $n = 16$. (b) $n = 20$. (c) $n = 25$. (d) $n = 28$.

change m depending on n with different degrees.

I compare the size of the circuits produced by my proposed method (PM) with the state-of-the-art method (SOTA) presented in [1]. The final circuits consist of CNOTs and single-qubit gates as elementary quantum gates. I only consider the number of CNOTs as they are more expensive than single-qubit gates in the NISQ. But consider that reducing CNOTs means I are reducing single-qubit gates as well. Fig. 8.4 shows results for $n = 16$, 20, 24, and 28. For each combination of parameters shown in the figure, I sampled 10 random states and show the average values. Each sub-figure shows how the number of CNOTs grows as I increase m as a function of n. For small m, SOTA is better as it is an efficient idea for sparse states. But by increasing m my results closes to SOTA and finally for $m = n^3$, PM outperform SOTA up to 31.85%, 17.4%, 13.1%, and 11.4% for n equal to 16, 20, 25, and 28, respectively. The reason is that in the decision diagram representation, for large m, there is a better sharing

Table 8.1: Experimental results for quantum states (qs) that have a sparse DD.

			PM				SOTA	
qs	n	m	#nodes	#RNodes	#paths (k)	#CNOTs	#CNOTs	Imp. (%)
set 1	20	n	33	6	2	13	275	95.27
set 2	20	$10n$	41	25	5	190	9983	98.10
set 3	30	n	60	10	4	62	463	86.61
set 4	30	$10n$	78	41	9	568	17019	96.66
QBA	20	n^3	32	110	18	1165	1361456	99.91
QBA	25	n^3	37	123	19	1321	2974248	99.95
QBA	30	n^3	44	141	22	1591	5512726	99.97

between basis states which results in a sparse decision diagram. The results for $n = 16$ are better than those for larger values of n because the percentage of non-zero amplitudes is higher for $n = 16$. Considering the sparsity condition in [1], $m \in o(\frac{2^n}{n})$, these values of m are still sparse. I conclude that my method is more useful than SOTA for large m.

Special states. To show my improvement for small m, I extract special states whose DD representations are sparse and the reduction rules work well on them. These states are mostly uniform states that share paths better. These states benefit from the effect of reduced nodes which reduce the number of paths and branching nodes in each path. This results in reducing the number of multiple-controlled gates and their control qubits which is required for computing the ancilla qubit. Table 8.1 shows the average results for such states (set 1, 2, 3, and 4) in comparison with SOTA. I consider two different numbers of qubits 20, 30, and small $m = n, 10n$. For each quantum state set, using the proposed method, I extract results regarding the number of nodes, number of reduced nodes, number of paths, and number of CNOTs. The number of reduced nodes shows that I can prepare several basis states together which reduces the number of CNOTs. Moreover, the number of paths, which is important in my complexity, is much less than the number of basis states, which results in reducing CNOTs. I also extracted the number of CNOTs by SOTA. Comparison shows that I reduce the number of CNOTs up to 98%.

Quantum Byzantine agreement (QBA) represents the quantum version of Byzantine agreement which works in constant time. In this protocol, for n players, I need to prepare the quantum state

$$|\varphi\rangle = \frac{1}{\sqrt{n^3}} \sum_{i=1}^{n^3} |i\rangle \tag{8.7}$$

on n qubits. For large n, this state is sparse. Table 8.1 shows its results. The proposed method prepares this state more efficiently. As shown in the Table 8.1, I reduce number of CNOTs by 99.97 % for QBA when $n = 30$. The reason is that the number of paths is much less than the number of non-zero basis states.

8.5 Algorithm performance

Correctness. First I explain how my algorithm prepares an arbitrary n-qubit state, given by Eq. (8.2), without any approximation error. It is enough to show that, starting from the initial state $|\text{no}\rangle_{q_A} \otimes |0\rangle^{\otimes n}$, in each iteration, in which the path $p_i \in P$ is traversed, I create a part $|\text{yes}\rangle \otimes \sum_{s \in S_{p_i}} \alpha_s |s\rangle$ of the target state, where S_{p_i} is the collection of basis states that are merged into path p_i in the creation of DD.

Meanwhile, I keep the prepared parts $|\text{yes}\rangle \otimes \sum_{j<i} \sum_{s \in S_{p_j}} \alpha_s |s\rangle$ untouched. (Be reminded that $p_1 \succ p_2 \succ \cdots \succ p_k$.) In this way, after traversing the last path p_k, I end up with $|\text{yes}\rangle \otimes \sum_{j=1}^{k} \sum_{s \in S_{p_j}} \alpha_s |s\rangle$ as desired, where the system is in the target state and is uncorrelated with the ancillary qubit q_A.

To see how this is achieved in each iteration, first, notice that a path is uniquely characterised by its branching nodes and their values. For example, the path 000101 can be specified by $q_1 = 0$, $q_4 = 1$, $q_5 = 0$, and $q_6 = 1$, as in between q_1 and q_4 I adopt the convention that both q_2 and q_3 take the same value as q_1. Therefore, it is enough to prepare a branch without altering other branches, by acting on each node using its preceding branching nodes as the control. In my algorithm (more precisely, in preparation rule), I further reduce the cost by the following crucial observation: When working on a qubit q in p_i, consider its closest ancestor whose value is one in p_i, denoted by $\tilde{q}$. Since the sequence $p_1, p_2, \ldots, p_k$ is also ordered, only those completed parts (i.e. the partial state $\sum_{j<i} \sum_{s \in S_{p_j}} \alpha_s |s\rangle$) corresponding to paths $p_1, \ldots, p_{i-1}$ can have $\tilde{q} = |1\rangle$. On the other hand, for those paths where $\tilde{q} = |1\rangle$, they have already been completed and thus are tagged $|\text{yes}\rangle$ (regarded as $|0\rangle$ when used as a control qubit) on q_A. Therefore, it is sufficient to use two qubits ($\tilde{q}$ and q_A) as the control to make sure that other completed parts are unaltered in the course of preparing the i-th part. As a result, I can complete the i-th part without affecting the prepared paths by following the preparation rule of the algorithm. Since the branching nodes uniquely determine a path, I can flip the value of q_A of the i-th part from $|\text{no}\rangle$ to $|\text{yes}\rangle$ by following the the computing the ancilla rule.

Circuit complexity. In DD, p_i and p_{i-1} may share a common sub-path; therefore, I do not need to start preparation from the root for every p_i. As a result, this approach allows for the addition of fewer gates, thereby reduces the number of single-qubit gates and CNOTs.

My idea is based on DD which I use reduced ordered BDD or ADD to represent the quantum state. The use of these techniques results in a more compact representation of the quantum state and eliminates redundancies, thereby reducing circuit costs during preparation. Moreover, reduced nodes enables preparing some basis states together. Hence, in contrast to the previous works that the number of basis states (m) is considered in the circuit complexity, the number of paths (k) is important in my complexity, and always

$$k \leq m. \tag{8.8}$$

According to Subsection 8.3, preparing a path is divided into two parts: preparing the path and computing the ancilla qubit. As a quantum circuit, it requires a sequence of 2-controlled gates to prepare the corresponding basis state (or basis states), and a multiple-controlled NOT gate to compute the ancilla qubit.

To compute the circuit complexity, I need to compute the number of 2-controlled gates for the first part, and the number of controls for the second part. The number of 2-controlled gates depends on the number of branching nodes in the path, and the number of one-child in the path of the corresponding basis state. Moreover, in the DD, paths have overlap and I prepare each basis state from the last common node with the previous basis state instead of starting from root. Considering this optimization, my algorithm reduces the number of 2-controlled gates. But in the worst-case I require n 2-controlled gates. Decomposition of each 2-controlled gates require 4 or 6 CNOT, and so I need $O(n)$ CNOTs. For the second part, the number of controls is equal to the number of branching nodes in the path. Then, I make use of the method proposed in [89] to decompose multiple-controlled NOT gate using $O(n)$ CNOT gates and one ancilla. I repeat same procedure for k paths and so, in total, the number of CNOTs is equal to

$$\#CNOTs = k \times O(n). \tag{8.9}$$

Time complexity. I traverse DD twice to first compute G gates and secondly prepare the quantum state. As I visit each node once, each traversal is linear in the number of nodes, and such a number increases mildly (but not always) with problem size (i.e., qubits). The number of nodes depends on the number of paths and the number of qubits in each path. Hence, the number of nodes is always less than kn as there exist sharing nodes at least for the root. As a result, the classical runtime is less than $2kn$, which is less than the time required by the state of the art [1].

8.6 Discussion

In this chapter, I have proposed an algorithm to prepare quantum states deterministically. My idea is based on preparing basis states one-by-one instead of operating one-by-one on the qubits. The latter is the key idea in general quantum state preparation algorithms. I have utilized DDs to represent quantum states in an efficient way. This allows my algorithm to be dependent on the number of paths where related works [135, 136, 1] are dependent on the number of basis states. I prepare the paths from the largest to the smallest regarding their binary bit strings. To do so, I traverse the DD in the pre-order traversal. Through this traversal, I visit nodes on a path. For each node, depending on the existence of its two children (i.e. *branching node*), I decide to append either 2-controlled single-target gates with different targets or just skip that. Upon preparing the path, an ancilla qubit is computed by adding a multiple-controlled NOT gate with the number of controls equal to the number of branching nodes in the path. Considering the decomposition method in [89], preparing each path and computing the ancilla qubit require $O(n)$ CNOTs. As a result, the final circuit cost depends on the number of paths and equals $O(kn)$. DDs help in achieving a compact representation of the state vector by reducing redundancies. The main advantages of my DD-based approach are:

- For preparing each path, I do not need to start from the root node. I go back to the last common node with the previous path.
- When there are redundant nodes, removing them causes merging basis states to the same path and I can prepare them together. This helps in two ways. First, it reduces the

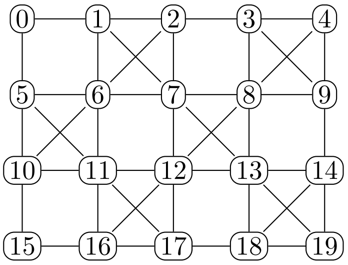


Figure 8.5: **Coupling map for the IBM Q Tokyo.** Here 0, 1, ..., 19 stand for physical qubits, and the edges indicate their connectivity.

number of iterations (k). Second, it reduces the number of branching nodes in paths which decreases the number of control qubits for computing the ancilla qubit.

Experimental results show that my idea works well for sparse DDs in which the number of paths and branching nodes are reduced. A sparse DD will be achieved when either m is small or m is not small but basis states share paths and can be prepared together. Hence, my algorithm besides SOTA, work very well to prepare sparse states and states with sparse DDs. As future work, I can consider variable reordering in DD to get a more sparse DD.

As a concluding remark, I note that analyses in this work are done assuming full connectivity between qubits, whereas a realistic *Quantum Processing Unit* (QPU) is often subject to limited qubit connectivity. In the following, I compare my algorithm to SOTA over an example that takes into account the limited qubit connectivity.

Example 8.6.1. *Consider preparing a uniform-amplitude quantum state corresponding to*

$$S = \{1000,\ 0100,\ 0011,\ 0010,\ 0001,\ 0000\}. \tag{8.10}$$

To prepare it on a QPU with full qubit connectivity, my method and SOTA require 10 and 12 CNOTs, respectively. When preparing it on IBM's 20-qubit Tokyo with a coupling map as shown in Fig. 8.5, the cost depends on the mapping from logical qubits to physical qubits, which I chose to be:

$$\{q_1 \to 0,\ q_2 \to 1,\ q_3 \to 6,\ q_4 \to 7\},\ q_A \to 2. \tag{8.11}$$

Under this mapping, compiling the circuit generated by my method and compiling the one generated by SOTA both result in two extra SWAP gates. As each SWAP is decomposed into three CNOTs, the final numbers of CNOTs for my method and SOTA are 16 and 18, respectively. Hence, for this example, my method outperforms SOTA both before and after the compilation.

8.7 Summary

While the complexity of preparing a generic quantum state is exponential in the number of qubits, in many practical tasks the state to prepare has a certain structure that allows for faster preparation. In this chapter, I considered quantum states that can be efficiently represented by (reduced) decision diagrams, a versatile data structure for the representation and analysis of Boolean functions. I designed an algorithm that utilises the structure of decision diagrams to prepare their associated quantum states. My algorithm has a circuit complexity that is linear in the number of paths in the decision diagram. Numerical experiments shown that my algorithm reduces the circuit complexity by up to 31.85% compared to the state-of-the-art algorithm, when preparing generic n-qubit states with n^3 non-zero amplitudes. Additionally, for states with sparse decision diagrams, including the initial state of the quantum Byzantine agreement protocol, my algorithm reduces the number of CNOTs by 86.61% ~ 99.9%.

Open-source Development Part IV

9 angel

Figure 9.1: angel's logo

angel, whose logo is shown in Fig. 9.1, is a modern *Quantm State Preparation* (QSP) library implemented in C++-17. The code and documentation is provided at "https://github.com/fmozafari/angel". As it is modular and header-only, it can be easily integrated with other tools and often outperforms implementation developed in high-level programming languages such as Python.

The *angel* library implements algorithms for QSP with the purpose of synthesizing an optimized quantum circuit to prepare a given quantum state. As an objective function, the algorithms focus on minimizing the quantum circuit's depth and the number of elementary quantum gates. In particular, the algorithms reduce the number of CNOT gates, which are in many experimental NISQ architectures relatively expensive when compared to other elementary quantum gates.

Finding the optimum quantum circuit with the minimum number of elementary gates, however, is in practice for arbitrary quantum states intractable. The angel library provides several different heuristics, which allow its users to trade-off runtime for quality. A good quantum circuit realization can be obtained fast and, if a user is willing to invest more runtime, the proposed algorithms are often capable to achieve substantial gate reductions. In combination with the tweedledum, a C++-17 header-only library for quantum circuit synthesis, angel can generate quantum circuits in standard quantum circuit formats, such as *Quantum Assembly* (QASM) or *Quantum Instruction Language* (QUIL).

In the current version, the *angel* library implements mainly algorithms for preparing uniform quantum states, a special class of quantum states that can be represented with Boolean functions, and sparse quantum states.

9.1 Uniform quantum state preparation

These states are superpositions of basis states, where all amplitudes are either zero or have the same value. As a key point, I map uniform quantum states to Boolean functions. By using Boolean functions to represent uniform quantum states, I can apply the Shannon decomposition method to recursively solve the problem of state preparation. The algorithm iterates over the variables of the Boolean function, which correspond to qubits, and prepares them one-by-one, by computing the probability of being zero for the variable depending on previously prepared variables. This computational step requires counting the number of ones for each recursive co-factor of the Boolean function. The probability is then the number of ones of the negative co-factor divided by the number of ones of the current function. To reduce the number of elementary quantum gates, I utilize decision diagrams and functional dependency analysis methods.

Using Decision Diagrams. The algorithm in [53] is implemented to prepare uniform quantum states using BDDs, as a representation of Boolean functions. BDDs are particularly suitable for my purpose because counting and co-factoring can be very efficiently implemented as BDD operations. More details are available in [53].

Using Functional Dependencies. Utilizing Boolean functions allows identifying functional dependencies among variables. In several cases, however, the recursive decomposition can be avoided in favor of more optimized constructions if a functional dependency among the current and the previously prepared qubits is recognized. Such functional dependencies have been developed in the context of logic synthesis. The identified functional dependencies for a qubit q_i can be utilized in three ways: (1) to reduce the number of control qubits if q_i depends only on a subset of the previously prepared qubits, (2) to reduce the number of elementary quantum gates if the functional dependency can be well expressed with the library of hardware supported quantum gates, and (3) to reduce the number of control qubits for preparing other next qubits to be prepared. Two approaches are presented to identify functional dependencies in [4]. These approaches are implemented using truth tables-based algorithms: the first approach, pattern search, identifies dependencies among variables that have a fixed and predefined structure; the second approach, ESOP synthesis, uses a SAT-based synthesis algorithm [66] for Exclusive-or Sum-Of-Product (ESOP) forms with a modified cost function. Finding dependencies in form of ESOP expressions with an XOR with many fanins and ANDs with only few fanins are particularly useful because they are the most general dependency structure that reduce the number of elementary gates. Moreover, I make use of variable reordering to ensure that no beneficial dependency is overlooked. More details are available in [4].

9.2 Sparse quantum state preparation

These states are superpositions over a given set $S \subset \{0,1\}^n, |S| << 2^n$. I present these states using *Algebraic Decision Diagram* (ADD). Each path in the ADD shows its corresponding basis state from the set S. I go through each path and prepare each basis state efficiently. Then, I use an ancilla qubit to show the basis state is prepared. As an alternative, I use BDDs to prepare sparse and uniform quantum states. Details on this algorithm are available in [55].

9.3 Examples

I show how to use a dependency analysis and variable reordering algorithm to synthesize a quantum circuit from a Boolean function given as a truth table using *angel* in combination with *kitty*[1] and *tweedledum*[2] as below:

```
#include <angel/angel.hpp>
#include <tweedledum/IR/Circuit.h>
#include <kitty/kitty.hpp>

/* Prepare a truth table */
kitty::dynamic_truth_table tt( 3 );
kitty::create_from_binary_string( tt,
        std::string( "1000" "0001" ) );

/* Prepare tweedledum's network type */
tweedledum::Circuit network;

/* Setup ESOP-based dependency analysis */
angel::esop_deps_analysis::parameter_type epars;
angel::esop_deps_analysis::statistics_type estats;
angel::esop_deps_analysis esop( epars, estats );

/* Setup exhaustive reordering strategy */
angel::exhaustive_reordering order;

/* Prepare parameters and statistics */
angel::state_preparation_parameters ps;
angel::state_preparation_statistics st;

/* Perform state preparation */
angel::uniform_qsp_deps
  <decltype(network), decltype( esop ), decltype( order )>
  ( network, esop, order, tt, ps, st);
```

[1] C++17 Library for analysis, compilation/synthesis, and optimization of quantum circuits, https://github.com/boschmitt/tweedledum.

[2] C++ truth table library, https://github.com/msoeken/kitty.

In the lines 6 to 8, I prepare the truth table of a 3-variable Boolean function using *kitty*. In the lines 14 to 16, I setup the dependency strategy, which takes `epars` and `estats` as arguments. The parameters are inputs and can be used to customize the dependency analysis algorithms. The statistics are outputs generated by the algorithm containing runtime as well as the number and types of identified dependencies. In line 19, I setup the reordering strategy, which considers all possible reordering of the 3-variable Boolean function. Other strategies, such as random or greedy reordering, exist. In line 26, I call my algorithm with the setup dependency and reordering strategies to prepare the quantum state given as a truth table. The final circuit is available as a network from *tweedledum*.

I show how to synthesize a quantum circuit for a sparse quantum state using *angel* in combination with *CUDD*[1] and *tweedledum* as below:

```
1  #include <angel/angel.hpp>
2  #include <tweedledum/IR/Circuit.h>
3  #include <cudd/cudd.h>
4  #include <cudd/cuddInt.h>
5
6  /* Create ADD from a map of
7                      basis states and their amplitudes */
8  Cudd cudd;
9  auto const f_add = create_add( cudd, map_amplitudes );
10
11 /* Prepare tweedledum's network type */
12 tweedledum::Circuit network;
13
14 /* Prepare statistics */
15 angel::sparse_qsp_statistics stats;
16
17 /* Perform state preparation  */
18 angel::sparse_qsp<network_type>( network, f_add, stats );
```

In the lines 8 to 9, I read a map consisting of basis states and their corresponding amplitudes and use CUDD to create its corresponding ADD. In line 12, I specify returning network type from *tweedledum*. In line 15, I setup the variable `stats` that results in the final statistics of my generated circuit. Finally, in line 18, I call my QSP algorithm.

9.4 Summary

This chapter introduces the modular C++ library *angel* that I developed and maintain for quantum state preparation. The library provides several algorithms to enable scalable quantum state preparation and reduce the number of CNOT gates required for the preparation process.

[1] CUDD: CU Decision Diagram package, https://github.com/ivmai/cudd.

10 Conclusions

This thesis focused on proposing efficient and effective algorithms to outperform multiple challenges faced by quantum compilers. A quantum compiler, is a software tool that takes a high-level quantum algorithm and translates it into a lower-level quantum circuit that can be executed on a specific quantum hardware platform, while taking into account the hardware constraints of that platform. These constraints include limitations on the number of qubits, the connectivity between qubits, the available hardware-specific gate sets, and the restricted circuit depth due to noise. I studied several problems including circuit synthesis for permutations, and quantum state preparation, which are crucial tasks for quantum compilers.

In Part II, I define a problem in quantum circuit synthesis and its importance. On circuit synthesis, the contribution is as follows:

- In Chapter 4, I propose an automated compilation algorithm that is tailored to specific quantum hardware, aimed at translating quantum operations for implementing permutations. The algorithm takes as its inputs a permutation over 2^n elements, the coupling constraints of the targeted quantum computer, and a gate library. It then returns a quantum circuit composed of gates from the given library, while ensuring that the circuit respects the coupling constraints. To achieve this, I employ a Young-subgroup based reversible logic synthesis technique [39] that finds a sequence of $2n-1$ single-target gates for a given permutation over n qubits. To translate a single-target gate into a quantum circuit consisting of Clifford+Rz library gates, I present a general algorithm. Finally, I utilize an explicit rewiring technique to reduce the number of quantum gates required for implementation.

In Part III, the problem of quantum state preparation is defined. I also presented the ways that research address this problem. Next, I defined several important families of quantum states and my proposed algorithm to prepare them. On quantum state preparation, the contributions are as follows:

- In Chapter 6, I introduce uniform quantum states that are an important family of quantum states. Many well-known quantum states are uniform, such as Bell state, GHZ state, and W state. These states are a uniform superposition of basis states, which all amplitudes are either zero or have same values. I showed that I can map uniform quantum states into Boolean functions. Hence, I proposed and algorithm based on functional decomposition to prepare the state qubit-by-qubit, recursively. There are 2 contributions on preparing uniform quantum states as follows:

  - First, I utilized BDDs as a symbolic representation of Boolean functions to reduce circuit cost. BDDs by reducing redundancies can create a more compact representation, if some exists. This helps reducing unnecessary and redundant gates in creating the circuit for a quantum state.
  - Second, I found sometimes that there is a relation between qubits, and I can exploit that to reduce circuit cost instead of applying the general recursive approach. To extract dependencies, I utilized functional dependency analysis and proposed two methods, pattern search and ESOP based synthesis. In pattern search, I assume some fixed dependency functions such as Equality, AND, XOR, and there negations to find dependency functions. In ESOP based I utilized the method in [66] to expresses the dependency function as an ESOP term. Exploiting dependencies between qubits helps to reduce CNOTs by creating simpler gates that their decomposition require less CNOTs.

- In Chapter 7, I introduced another family of quantum states that I called them *cyclic quantum states*. I presented their importance and motivation behind preparing them separately by an efficient algorithm. These states are a uniform superposition of a group of cyclic permutations where all ones of the basis state are adjacent (exp. $|011100\rangle$) or in the front and the end of the basis state (exp. $|110001\rangle$). I proposed an algorithm to prepare them efficiently in linear complexity regarding the number of qubits.

- In Chapter 8, I proposed a deterministic algorithm that mainly considers sparse quantum states. The proposed method is based on preparing the basis states one-by-one, rather than operating on each qubit individually. This approach deviates from the conventional idea behind general quantum state preparation algorithms. I employed ADDs to represent quantum states efficiently, which allows my algorithm to be dependent on the number of paths, whereas other related works are dependent on the number of basis states. After preparing a path, an ancilla qubit is computed using a multi-controlled NOT gate with the number of controls equal to the number of branching nodes in the path. My approach offers several key advantages, including: (1) For each path, it is not needed to start from the root node. Instead, I go back to the last common node with the previous path, which saves time and computational resources. (2) When redundant nodes are removed, it causes merging of basis states to the same path, allowing them to be prepared together. It reduces the number of iterations required for preparing the quantum state, and it decreases the number of branching nodes in paths, which in turn reduces the number of control qubits needed for computing the ancilla qubit.

- In Chapter 9, I presented my contribution to developing an open-source tool, called *angel*. *angel* is is designed for quantum state preparation and provides algorithms for preparing uniform quantum states using BDD and truth tables. In addition, the library includes a functional dependency analysis feature to find dependencies between qubits for further improvements. The implementation of sparse quantum state preparation using ADD is also provided. The *angel* library offers a modular and extensible framework for quantum state preparation.

10.1 Future directions

While this thesis makes a valuable contribution to the field, it is important to note that it represents only a small portion of the effort required to achieve practical quantum computing. The future of quantum computing faces several significant challenges that must be overcome to realize the full potential of this technology. These challenges include developing scalable quantum hardware with low error rates, improving quantum algorithms to solve real-world problems, and finding ways to efficiently integrate quantum computers into existing computational infrastructure. Additionally, new methods for quantum error correction and fault tolerance will be crucial for achieving practical, error-free quantum computation.

As a final remark, I would like to highlight several potential directions for my work on quantum state preparation and extending the *angel* tool.

Other families of quantum states. So far, I have presented three families of quantum states: uniform quantum states, cyclic quantum states, and sparse quantum states. While it takes an exponential cost in circuit and runtime to prepare arbitrary states, I have proposed new methods for efficiently preparing these three families of states. As future works, it is a good idea to investigate other important families of quantum states and find a way to prepare them more efficiently.

Arbitrary quantum states. As I mentioned before, the task of preparing arbitrary quantum states requires exponential quantum gates and circuit depth. I am thinking of finding a way to propose an idea for efficiently preparing arbitrary quantum states. My idea is based on utilizing current ideas that I proposed. To do that, I am looking for a way to break down an arbitrary quantum state into several smaller, separable superposition states which can make the task of preparing them more efficient. There may be multiple ways of dividing the arbitrary state into smaller. I will consider different partitioning schemes and determine which one yields the most efficient preparation strategy. Breaking down a state into smaller parts may introduce entanglement between the parts. I need to keep track of this entanglement and ensure that it does not lead to exponential overheads in preparing the final state.

Extending *angel*. One of my objectives is to expand the capabilities of *angel* by incorporating existing techniques for quantum state preparation. My goal is to develop a comprehensive tool for QSP by integrating a variety of approaches into *angel*.

Tensor network simulation. As quantum computing applications and complexity continue to grow, the simulation of quantum circuits is becoming increasingly important for research, development, and verification purposes. The recent development of tensor network-based simulation [138, 139] has garnered significant interest due to its potential for simulating large-scale quantum computers with over 100 qubits, which is currently not feasible using existing classical computers.

One of the main challenges in tensor network-based quantum circuit simulation is efficiently constructing a tensor network representation for an initial quantum state. This challenge becomes increasingly difficult with larger numbers of qubits, as the required storage grows exponentially. To address this issue, researchers have proposed using the *Matrix Product State* (MPS) [140] representation, which requires only linear storage. However, the traditional approach for creating MPS involves computationally expensive *Singular Value Decomposition* (SVD) [141], which quickly becomes intractable for larger numbers of qubits. This limitation makes tensor network simulation impractical for many cases.

An interesting future direction for research is developing efficient methods for preparing quantum states for tensor network simulations using MPS without relying on SVD computation. By addressing this challenge, researchers can significantly enhance the efficiency and scalability of tensor network-based quantum circuit simulation, paving the way for practical applications in the field of quantum computing.

Bibliography

[1] N. Gleinig and T. Hoefler, "An efficient algorithm for sparse quantum state preparation," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 433–438, IEEE, 2021.

[2] A. Bärtschi and S. Eidenbenz, "Deterministic preparation of dicke states," in *International Symposium on Fundamentals of Computation Theory*, pp. 126–139, Springer, 2019.

[3] A. Burchardt, J. Czartowski, and K. Życzkowski, "Entanglement in highly symmetric multipartite quantum states," *arXiv preprint arXiv:2105.12721*, 2021.

[4] F. Mozafari, H. Riener, M. Soeken, and G. De Micheli, "Efficient boolean methods for preparing uniform quantum states," in *IEEE Transactions on Quantum Engineering (TQE), in press*, pp. 170–175, IEEE, 2021.

[5] R. P. Feynman, "Simulating physics with computers," *International journal of theoretical physics*, vol. 21, no. 6, pp. 467–488, 1982.

[6] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[7] L. M. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, "Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance," *Nature*, vol. 414, no. 6866, pp. 883–887, 2001.

[8] J. Jones and M. Mosca, "Implementation of a quantum algorithm on a nuclear magnetic resonance quantum computer," *Journal of Chemical Physics*, vol. 109, no. 5, pp. 1648–1653, 1998.

[9] M. H. Devoret and R. J. Schoelkopf, "Superconducting circuits for quantum information: an outlook," *Science*, vol. 339, no. 6124, pp. 1169–1174, 2013.

[10] S. Bravyi, O. Dial, J. M. Gambetta, D. Gil, and Z. Nazario, "The future of quantum computing with superconducting qubits," *Journal of Applied Physics*, vol. 132, no. 16, p. 160902, 2022.

[11] Google, "A preview of Bristlecone, Google's new quantum processor," 2018. Article on Google AI Blog, posted online March 5, 2018.

[12] IBM, "IBM builds its most powerful universal quantum computing processors," 2017. Press release by IBM, posted online May 17, 2017.

[13] IBM, “Ibm unveils 400 qubit-plus quantum processor and next-generation ibm quantum system two.” https://newsroom.ibm.com/latest-news-quantum-innovation, 2022.

[14] K. R. Brown, J. Kim, and C. Monroe, “Co-designing a scalable quantum computer with trapped atomic ions,” *npj Quantum Information*, vol. 2, no. 1, pp. 1–10, 2016.

[15] J. L. O’brien, “Optical quantum computing,” *Science*, vol. 318, no. 5856, pp. 1567–1570, 2007.

[16] P. Kok, W. J. Munro, K. Nemoto, T. C. Ralph, J. P. Dowling, and G. J. Milburn, “Linear optical quantum computing with photonic qubits,” *Reviews of modern physics*, vol. 79, no. 1, p. 135, 2007.

[17] M. Freedman, A. Kitaev, M. Larsen, and Z. Wang, “Topological quantum computation,” *Bulletin of the American Mathematical Society*, vol. 40, no. 1, pp. 31–38, 2003.

[18] J. Preskill, “Quantum computing in the nisq era and beyond,” *Quantum*, vol. 2, p. 79, 2018.

[19] C. H. Bennett and G. Brassard, “An update on quantum cryptography.,” in *Crypto*, vol. 84, pp. 475–480, Springer, 1984.

[20] and others, “Advances in quantum cryptography,” *arXiv preprint arXiv:1906.01645*, 2019.

[21] G. H. Low and I. L. Chuang, “Hamiltonian simulation by qubitization,” *Quantum*, vol. 3, p. 163, 2019.

[22] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, “Quantum machine learning,” *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.

[23] S. McArdle, S. Endo, A. Aspuru-Guzik, S. C. Benjamin, and X. Yuan, “Quantum computational chemistry,” *Reviews of Modern Physics*, vol. 92, no. 1, p. 015003, 2020.

[24] A. W. Harrow, A. Hassidim, and S. Lloyd, “Quantum algorithm for linear systems of equations,” *Physical review letters*, vol. 103, no. 15, p. 150502, 2009.

[25] M. A. Nielsen and I. Chuang, “Quantum computation and quantum information,” 2002.

[26] C. Chen, B. Schmitt, H. Zhang, L. S. Bishop, and A. Javadi-Abhar, “Optimizing quantum circuit synthesis for permutations using recursion,” in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 7–12, 2022.

[27] A. Zulehner and R. Wille, “Compiling su (4) quantum circuits to ibm qx architectures,” in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pp. 185–190, 2019.

[28] W.-H. Lin, B. Tan, M. Y. Niu, J. Kimko, and J. Cong, “Domain-specific quantum architecture optimization,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 12, no. 3, pp. 624–637, 2022.

[29] B. Nash, V. Gheorghiu, and M. Mosca, “Quantum circuit optimizations for nisq architectures,” *Quantum Science and Technology*, vol. 5, no. 2, p. 025010, 2020.

[30] R. Wille, L. Burgholzer, and A. Zulehner, "Mapping quantum circuits to ibm qx architectures using the minimal number of swap and h operations," in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

[31] A. Kole, S. Hillmich, K. Datta, R. Wille, and I. Sengupta, "Improved mapping of quantum circuits to ibm qx architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2375–2383, 2019.

[32] B. Tan and J. Cong, "Layout synthesis for near-term quantum computing: Gap analysis and optimal solution," in *Design Automation of Quantum Computers*, pp. 25–40, Springer, 2022.

[33] B. Tan and J. Cong, "Optimal layout synthesis for quantum computing," in *Proceedings of the 39th International Conference on Computer-Aided Design*, pp. 1–9, 2020.

[34] B. Tan and J. Cong, "Optimality study of existing quantum computing layout synthesis tools," *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1363–1373, 2020.

[35] J. S. Otterbach *et al.*, "Unsupervised machine learning on a hybrid quantum computer," *arXiv preprint arXiv:1712.05771*, 2017.

[36] Intel, "The future of quantum computing is counted in qubits," 2018. Press release by Intel, posted online May 2, 2018.

[37] C. Figgatt, D. Maslov, K. A. Landsman, N. M. Linke, S. Debnath, and C. Monroe, "Complete 3-qubit Grover search on a programmable quantum computer," vol. 8, no. 1918, pp. 1–9, 2017.

[38] Alibaba, "Alibaba Cloud and CAS launch one of the world's most powerful public quantum computing services," 2018. Press release by Alibaba Cloud, posted online March 1, 2018.

[39] A. De Vos and Y. Van Rentergem, "Young subgroups for reversible computers," *Advances in Mathematics of Communications*, vol. 2, no. 2, pp. 183–200, 2008.

[40] V. Bergholm, J. J. Vartiainen, M. Möttönen, and M. M. Salomaa, "Quantum circuits with uniformly controlled one-qubit gates," *Physical Review A*, vol. 71, no. 5, p. 052330, 2005.

[41] P. Kaye and M. Mosca, "Quantum networks for generating arbitrary quantum states," in *International Conference on Quantum Information*, p. PB28, Optical Society of America, 2001.

[42] M. Mottonen, J. J. Vartiainen, V. Bergholm, and M. M. Salomaa, "Transformation of quantum states using uniformly controlled rotations," *Quantum Information and Computation*, vol. 5, no. 6, pp. 467–473, 2005.

[43] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of quantum-logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 1000–1010, 2006.

[44] P. Niemann, R. Datta, and R. Wille, "Logic synthesis for quantum state generation," in *2016 IEEE 46th International Symposium on Multiple-Valued Logic (ISMVL)*, pp. 247–252, 2016.

[45] M.-X. Luo, S.-Y. Ma, Y. Deng, and X. Wang, "Deterministic generations of quantum state with no more than six qubits," *Quantum Information Processing*, vol. 14, no. 3, pp. 901–920, 2015.

[46] N. Shenvi, J. Kempe, and K. B. Whaley, "Quantum random-walk search algorithm," *Physical Review A*, vol. 67, no. 5, p. 052307, 2003.

[47] W. Dür, G. Vidal, and J. I. Cirac, "Three qubits can be entangled in two inequivalent ways," *Physical Review A*, vol. 62, no. 6, p. 062314, 2000.

[48] D. M. Greenberger, M. A. Horne, and A. Zeilinger, "Going beyond bell's theorem," in *Bell's theorem, quantum theory and conceptions of the universe*, pp. 69–72, Springer, 1989.

[49] E. D'Hondt and P. Panangaden, "The computational power of the W and GHZ states," *arXiv preprint quant-ph/0412177*, 2004.

[50] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.

[51] M. Soeken, F. Mozafari, B. Schmitt, and G. De Micheli, "Compiling permutations for superconducting qpus," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1349–1354, IEEE, 2019.

[52] F. Mozafari, M. Soeken, and G. De Micheli, "Automatic preparation of uniform quantum states utilizing boolean functions," *In International Workshop on Logic Synthesis (IWLS)*, 2019.

[53] F. Mozafari, M. Soeken, H. Riener, and G. De Micheli, "Automatic uniform quantum state preparation using decision diagrams," in 50^{th} *International Symposium on Multiple-Valued Logic (ISMVL)*, pp. 170–175, IEEE, 2020.

[54] F. Mozafari, Y. Yang, and G. De Micheli, "Efficient preparation of cyclic quantum states," in 27^{th} *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 460–465, IEEE, 2022.

[55] F. Mozafari, G. De Micheli, and Y. Yang, "Efficient deterministic preparation of quantum states using decision diagrams," *Physical Review A*, vol. 106, no. 2, p. 022617, 2022.

[56] F. Mozafari, R. Heinz, and G. De Micheli, "Uniform quantum state preparation: A boolean approach for preparing uniform quantum states efficiently and precisely," *In International Workshop on Quantum Compilation (IWQC), Cambridge, UK*, September 2020.

[57] M. Soeken, G. Meuli, B. Schmitt, F. Mozafari, H. Riener, and G. De Micheli, "Boolean satisfiability in quantum compilation," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, p. 20190161, 2020.

[58] B. Schmitt, F. Mozafari, G. Meuli, H. Riener, and G. De Micheli, “From boolean functions to quantum circuits: A scalable quantum compilation flow in c++,” in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2021.

[59] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, and G. De Micheli, “The EPFL logic synthesis libraries,” Nov. 2019. arXiv:1805.05121v2.

[60] G. De Micheli, *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, 1994.

[61] T. Sasao, *Logic synthesis and optimization*, vol. 2. Springer, 1993.

[62] Y. Crama and P. L. Hammer, *Boolean functions: Theory, algorithms, and applications.* Cambridge University Press, 2011.

[63] S. B. Akers, Jr, “On a theory of boolean functions,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 7, no. 4, pp. 487–498, 1959.

[64] S. Foldes and P. L. Hammer, “Disjunctive and conjunctive normal forms of pseudo-boolean functions,” *Discrete applied mathematics*, vol. 107, no. 1-3, pp. 1–26, 2000.

[65] T. Sasao, “An exact minimization of AND-EXOR expressions using reduced covering functions,” in *The Synthesis and Simulation Meeting and International Interchange*, pp. 374–383, 1993.

[66] H. Riener, R. Ehlers, B. d. O. Schmitt, and G. D. Micheli, “Exact synthesis of esop forms,” in *Advanced boolean techniques*, pp. 177–194, Springer, 2020.

[67] A. Mishchenko and M. Perkowski, “Fast heuristic minimization of exclusive-sums-of-products,” in *International Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pp. 242–250, 2001.

[68] T. Sasao, “Representations of logic functions using exor operators,” *Representations of discrete functions*, pp. 29–54, 1996.

[69] U. Kalay, D. V. Hall, and M. A. Perkowski, “A minimal universal test set for self-test of exor-sum-of-products circuits,” *IEEE Transactions on Computers*, vol. 49, no. 3, pp. 267–276, 2000.

[70] V. Kolesnikov and T. Schneider, “Improved garbled circuit: Free xor gates and applications,” in *International Colloquium on Automata, Languages, and Programming*, pp. 486–498, Springer, 2008.

[71] G. Meuli, B. Schmitt, R. Ehlers, H. Riener, and G. De Micheli, “Evaluating esop optimization methods in quantum compilation flows,” in *International Conference on Reversible Computation*, pp. 191–206, Springer, 2019.

[72] K. Fazel, M. A. Thornton, and J. E. Rice, “Esop-based toffoli gate cascade generation,” in *2007 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 206–209, IEEE, 2007.

[73] B. Schmitt, M. Soeken, G. De Micheli, and A. Mishchenko, “Scaling-up esop synthesis for quantum compilation,” in *2019 IEEE 49th International Symposium on Multiple-Valued Logic (ISMVL)*, pp. 13–18, IEEE, 2019.

[74] Y. Sanaee and G. W. Dueck, “Generating toffoli networks from esop expressions,” in *2009 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 715–719, IEEE, 2009.

[75] J. Rice, K. Fazel, M. Thornton, and K. Kent, “Toffoli gate cascade generation using esop minimization and qmdd-based swapping,” in *Proceedings of the Reed-Muller Workshop (RM2009)*, pp. 63–72, 2009.

[76] S. B. Akers, “Binary decision diagrams,” *IEEE Computer Architecture Letters*, vol. 27, no. 06, pp. 509–516, 1978.

[77] R. Drechsler and D. Sieling, “Binary decision diagrams in theory and practice,” *International Journal on Software Tools for Technology Transfer*, vol. 3, pp. 112–136, 2001.

[78] R. E. Bryant, “Binary decision diagrams,” *Handbook of model checking*, pp. 191–217, 2018.

[79] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, and et. al., “Algebric decision diagrams and their applications,” *Formal methods in system design*, vol. 10, no. 2-3, pp. 171–206, 1997.

[80] M. Kwiatkowska, G. Norman, and D. Parker, “Stochastic model checking,” in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pp. 220–270, Springer, 2007.

[81] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier, “Spudd: stochastic planning using decision diagrams,” in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pp. 279–288, 1999.

[82] J. Dudek, V. Phan, and M. Vardi, “Addmc: weighted model counting with algebraic decision diagrams,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 1468–1476, 2020.

[83] P. A. M. Dirac, “A new notation for quantum mechanics,” in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 35, pp. 416–418, Cambridge University Press, 1939.

[84] R. Tumulka, “Dirac notation,” in *Compendium of Quantum Physics*, pp. 172–174, Springer, 2009.

[85] B. Schumacher, “Quantum coding,” *Physical Review A*, vol. 51, no. 4, p. 2738, 1995.

[86] H. Mäkelä and A. Messina, “N-qubit states as points on the bloch sphere,” *Physica Scripta*, vol. 2010, no. T140, p. 014054, 2010.

[87] S. Tamate, K. Ogawa, and M. Kitano, “Bloch-sphere representation of three-vertex geometric phases,” *Physical Review A*, vol. 84, no. 5, p. 052114, 2011.

[88] R. P. Feynman, “Quantum mechanical computers,” *Foundations of physics*, vol. 16, no. 6, pp. 507–531, 1986.

[89] C. Gidney, “Constructing large controlled nots.” https://algassert.com/circuits/2015/06/05/Constructing-Large-Controlled-Nots.html, 2015.

[90] D. Deutsch and R. Jozsa, “Rapid solution of problems by quantum computation,” *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, 1992.

[91] F. T. Chong, D. Franklin, and M. Martonosi, “Programming languages and compiler design for realistic quantum hardware,” vol. 549, no. 7671, pp. 180–187, 2017.

[92] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, R. Cleve, and I. L. Chuang, “Experimental realization of an order-finding algorithm with an NMR quantum computer,” vol. 85, no. 25, pp. 5452–5455, 2000.

[93] M. Amy, P. Azimzadeh, and M. Mosca, “On the CNOT-complexity of CNOT-phase circuits,” *arXiv preprint arXiv:1712.01859*, 2017.

[94] J. Welch, D. Greenbaum, S. Mostame, and A. Aspuru-Guzik, “Efficient quantum circuits for diagonal unitaries without ancillas,” *New Journal of Physics*, vol. 16, no. 3, p. 033040, 2014.

[95] N. Schuch and J. Siewert, “Programmable networks for quantum algorithms,” *Physical review letters*, vol. 91, no. 2, p. 027902, 2003.

[96] L. K. Grover, “A fast quantum mechanical algorithm for database search,” pp. 212–219, 1996.

[97] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, “Elementary gates for quantum computation,” *Physical review A*, vol. 52, no. 5, p. 3457, 1995.

[98] D. Maslov, “Advantages of using relative-phase Toffoli gates with an application to multiple control Toffoli optimization,” *Physical Review A*, vol. 93, no. 2, p. 022311, 2016.

[99] M. Saeedi and I. L. Markov, “Synthesis and optimization of reversible circuits - a survey,” *ACM Computing Surveys*, vol. 45, no. 2, pp. 21:1–21:34, 2013.

[100] D. Deutsch, “Quantum computational networks,” vol. A 425, pp. 73–90, 1989.

[101] R. Iten, R. Colbeck, I. Kukuljan, J. Home, and M. Christandl, “Quantum circuits for isometries,” *Physical Review A*, vol. 93, no. 3, p. 032318, 2016.

[102] A. Zulehner, S. Hillmich, I. L. Markov, and R. Wille, “Approximation of quantum states using decision diagrams,” in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 121–126, IEEE, 2020.

[103] C. Zoufal, A. Lucchi, and S. Woerner, “Quantum generative adversarial networks for learning and loading random distributions,” *npj Quantum Information*, vol. 5, no. 1, pp. 1–9, 2019.

[104] I. F. Araujo, D. K. Park, F. Petruccione, and A. J. da Silva, "A divide-and-conquer algorithm for quantum state preparation," *Scientific reports*, vol. 11, no. 1, pp. 1–12, 2021.

[105] Y. R. Sanders, G. H. Low, A. Scherer, and D. W. Berry, "Black-box quantum state preparation without arithmetic," *Physical review letters*, vol. 122, no. 2, p. 020502, 2019.

[106] K. Resch, J. Lundeen, and A. Steinberg, "Quantum state preparation and conditional coherence," *Physical review letters*, vol. 88, no. 11, p. 113601, 2002.

[107] M. Plesch and Č. Brukner, "Quantum-state preparation with universal gate decompositions," *Physical Review A*, vol. 83, no. 3, p. 032302, 2011.

[108] S. Arunachalam, A. Belovs, A. M. Childs, R. Kothari, A. Rosmanis, and R. de Wolf, "Quantum coupon collector," in *15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[109] M. Ben-Or and A. Hassidim, "Fast quantum byzantine agreement," in *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pp. 481–485, 2005.

[110] M. Hillery, V. Bužek, and A. Berthiaume, "Quantum secret sharing," *Physical Review A*, vol. 59, no. 3, p. 1829, 1999.

[111] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, and et al., "Qiskit: An open-source framework for quantum computing," 2019.

[112] R. Babbush, C. Gidney, D. W. Berry, N. Wiebe, J. McClean, A. Paler, A. Fowler, and H. Neven, "Encoding electronic spectra in quantum circuits with linear t complexity," *Physical Review X*, vol. 8, no. 4, p. 041015, 2018.

[113] L. Grover and T. Rudolph, "Creating superpositions that correspond to efficiently integrable probability distributions," *arXiv preprint quant-ph/0208112*, 2002.

[114] Wikipedia, "Quantum Byzantine agreement — Wikipedia, the free encyclopedia." https://en.wikipedia.org/wiki/Quantum_Byzantine_agreement, 2020. [Online; accessed 18-November-2020].

[115] J.-H. R. Jiang and R. K. Brayton, "Functional dependency for verification reduction," in *International Conference on Computer Aided Verification*, pp. 268–280, Springer, 2004.

[116] R. H. Dicke, "Coherence in spontaneous radiation processes," *Physical review*, vol. 93, no. 1, p. 99, 1954.

[117] Z. Zhang and L. Duan, "Quantum metrology with dicke squeezed states," *New Journal of Physics*, vol. 16, no. 10, p. 103037, 2014.

[118] G. Tóth and I. Apellaniz, "Quantum metrology from a quantum information science perspective," *Journal of Physics A: Mathematical and Theoretical*, vol. 47, no. 42, p. 424006, 2014.

[119] N. Kiesel, C. Schmid, G. Tóth, E. Solano, and H. Weinfurter, "Experimental observation of four-photon entangled dicke state with high fidelity," *Physical review letters*, vol. 98, no. 6, p. 063604, 2007.

[120] W. Wieczorek, R. Krischek, N. Kiesel, P. Michelberger, G. Tóth, and H. Weinfurter, "Experimental entanglement of a six-photon symmetric dicke state," *Physical review letters*, vol. 103, no. 2, p. 020504, 2009.

[121] D. Hume, C.-W. Chou, T. Rosenband, and D. J. Wineland, "Preparation of dicke states in an ion chain," *Physical Review A*, vol. 80, no. 5, p. 052302, 2009.

[122] C. S. Mukherjee, S. Maitra, V. Gaurav, and D. Roy, "Preparing dicke states on a quantum computer," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–17, 2020.

[123] J. K. Stockton, R. Van Handel, and H. Mabuchi, "Deterministic dicke-state preparation with continuous measurement and control," *Physical Review A*, vol. 70, no. 2, p. 022106, 2004.

[124] H. J. Kimble, "The quantum internet," *Nature*, vol. 453, no. 7198, pp. 1023–1030, 2008.

[125] S. Wehner, D. Elkouss, and R. Hanson, "Quantum internet: A vision for the road ahead," *Science*, vol. 362, no. 6412, 2018.

[126] V. Giovannetti, S. Lloyd, and L. Maccone, "Quantum metrology," *Physical Review Letters*, vol. 96, no. 1, p. 010401, 2006.

[127] W. K. Wootters, "Entanglement of formation of an arbitrary state of two qubits," *Physical Review Letters*, vol. 80, no. 10, p. 2245, 1998.

[128] C. H. Bennett, D. P. DiVincenzo, J. A. Smolin, and W. K. Wootters, "Mixed-state entanglement and quantum error correction," *Physical Review A*, vol. 54, no. 5, p. 3824, 1996.

[129] I. Devetak and A. Winter, "Distillation of secret key and entanglement from quantum states," *Proceedings of the Royal Society A: Mathematical, Physical and engineering sciences*, vol. 461, no. 2053, pp. 207–235, 2005.

[130] C. H. Bennett, H. J. Bernstein, S. Popescu, and B. Schumacher, "Concentrating partial entanglement by local operations," *Physical Review A*, vol. 53, no. 4, p. 2046, 1996.

[131] C. H. Bennett, G. Brassard, S. Popescu, B. Schumacher, J. A. Smolin, and W. K. Wootters, "Purification of noisy entanglement and faithful teleportation via noisy channels," *Physical Review Letters*, vol. 76, no. 5, p. 722, 1996.

[132] F. Hahn, A. Pappa, and J. Eisert, "Quantum network routing and local complementation," *npj Quantum Information*, vol. 5, no. 1, pp. 1–7, 2019.

[133] M. Pant, H. Krovi, D. Towsley, L. Tassiulas, L. Jiang, P. Basu, D. Englund, and S. Guha, "Routing entanglement in the quantum internet," *npj Quantum Information*, vol. 5, no. 1, pp. 1–9, 2019.

[134] I. Streinu and L. Theran, “Sparse hypergraphs and pebble game algorithms,” *European Journal of Combinatorics*, vol. 30, no. 8, pp. 1944–1964, 2009.

[135] T. M. de Veras, L. D. da Silva, and A. J. da Silva, “Double sparse quantum state preparation,” *Quantum Information Processing*, vol. 21, no. 6, p. 204, 2022.

[136] E. Malvetti, R. Iten, and R. Colbeck, “Quantum circuits for sparse isometries,” *Quantum*, vol. 5, p. 412, 2021.

[137] X.-M. Zhang, T. Li, and X. Yuan, “Quantum state preparation with optimal circuit depth: Implementations and applications,” *Physical Review Letters*, vol. 129, no. 23, p. 230504, 2022.

[138] J. Biamonte and V. Bergholm, “Tensor networks in a nutshell,” *arXiv preprint arXiv:1708.00006*, 2017.

[139] J. Biamonte, “Lectures on quantum tensor networks,” *arXiv preprint arXiv:1912.10049*, 2019.

[140] R. Orús, “A practical introduction to tensor networks: Matrix product states and projected entangled pair states,” *Annals of physics*, vol. 349, pp. 117–158, 2014.

[141] K. Batselier, W. Yu, L. Daniel, and N. Wong, “Computing low-rank approximations of large-scale matrices with the tensor network randomized svd,” *SIAM Journal on Matrix Analysis and Applications*, vol. 39, no. 3, pp. 1221–1244, 2018.

Fereshte Mozafari

Personal Data

Address:	EPFL, Building INF 337, Station 14, Lausanne 1015, Switzerland.
E-mail:	fereshte.mozafari@epfl.ch
GitHub:	https://github.com/fmozafari
LinkedIn:	https://www.linkedin.com/in/fereshte-mozafari
Scholar:	scholar link

Award

I won **Google PhD Fellowship Award in Quantum Computing**. (Sep. 2020)

Topic of Interests

Computer Science, Quantum Computing, High Performance Computing, and Electronic Design Automation.

Education

2018-Present

Ph.D. in Electrical Engineering
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
Thesis: "Optimizing Quantum Compilers: Efficient and Effective Algorithms"
Supervisor: Prof. Giovanni De Micheli

2013-2015

M.Sc. in Computer Architecture
Sharif University of Technology, Tehran, Iran
Thesis: "Reliability Enhancement of SSD-based Storage Systems Using Erasure Codes"
Supervisor: Prof. Seyed-Ghasem Miremadi

2009-2013

B.Sc. in Computer Engineering
Sharif University of Technology, Tehran, Iran
Thesis: "Proposing one Fault Tolerance Method for Tag Part in Cache"
Supervisor: Prof. Seyed-Ghasem Miremadi

Work Experience

Aug. 2022-Present

Research and Development Intern, **NVIDIA**, Zurich, Switzerland
- I worked on developing NVIDIA cuQuantum circuit performance benchmark suite. link
- I proposed an efficient method to decompose quantum circuits for tensor network simulations that resulted in submitting a patent.

Feb. 2018-Present

Doctoral Assistant, EPFL, Lausanne, Switzerland
Integrated Systems Laboratory (LSI)

Feb. 2018-Present

C++ Library Development, EPFL, Lausanne, Switzerland
I developed a *Quantum State Preparation* (QSP) library based on **C++-17**, called **angel**. *angel* provides several algorithms to enable a scalable QSP and to reduce the number of CNOTs. *angel* is available here: link

Mar. 2021-Jul. 2021

Research Visitor, ETH, Zurich, Switzerland
I worked in *Quantum Information Theory Group* under the supervision of Yuxiang Yang in Renato Renner group. I proposed new ideas for preparing cyclic quantum states, and sparse quantum states. I published papers and developed ideas in *angel* tool.

SEP. 2019-JAN. 2020 | Machine Learning Project, EPFL, Lausanne, Switzerland

- I analyzed CERN datasets to recreate the process of discovering the Higgs particle by applying different machine learning algorithms, feature engineering, and modeling (Using **Python**).

- I developed a text sentiment classification for Twitter messages by applying different machine learning neural network models such as CNN, LSTM, and LSTM-CNN (Using **Python**, and **TensorFlow library**). The source code is available here: link

SEP. 2016-DEC. 2017 | Research Scientist, Sharif University of Technology, Tehran, Iran

I proposed an algorithm for DNA sequence alignment that can be implemented in optics, in collaboration with two groups. I published a paper that is available here: link.

PROFESSIONAL VOLUNTEERING EXPERIENCES

2019-2022 | Committee member of Electrical Engineering at EPFL

2019-2021 | Committee member of Quantum Computing Association at EPFL

SKILLS

Programming Languages:	C/C++, Python, Matlab, Android
Machine Learning Libraries:	TensorFlow, Keras, NLTK
Quantum Computing Packages:	cuQuantum, Cirq, Qiskit, Rigetti
Design & Simulation Tools:	Synopsis Design Compiler, Cadence Virtuoso, Modelsim, Tanner EDA Tools, AVR, Quartus, Hspice, Color Petri Net
Parallel Programming:	Cell BE, CUDA GPU
HDL:	VHDL, Verilog
Languages:	English (fluent), Persian (native)

PUBLICATIONS

Fereshte Mozafari, Yang Gao, and Yao-Lung L. Fang, "Method and apparatus for efficient representation of quantum oracles for tensor network simulation," Patent submitted in February 2023.

Fereshte Mozafari, Giovanni De Micheli, and Yuxiang Yang, "Efficient Deterministic Preparation of Quantum States Using Decision Diagram," In *Physical Review A*, 106.2, pp. 022617, APS, 2022.

Fereshte Mozafari, Yuxiang Yang, and Giovanni De Micheli, "Efficient Preparation of Cyclic Quantum States," In 2022 27^{th} *Asia and South Pacific Design Automation Conference* (ASP-DAC), pp. 460-465, IEEE, 2022.

Fereshte Mozafari, Giovanni De Micheli, and Yuxiang Yang, "Efficient Quantum State Preparation Using Decision Diagrams," In *Quantum Techniques in Machine Learning Conference* (QTML), Napoli, Italy, November 8-11, 2022. link

Fereshte Mozafari, Heinz Riener, Mathias Soeken, and Giovanni De Micheli,"Efficient Boolean Methods for Preparing Uniform Quantum States," In *IEEE Transactions on Quantum Engineering*, 2, pp. 1-12, 2021.

Bruno Schmitt, **Fereshte Mozafari**, Giulia Meuli, Heinz Riener, and Giovanni De Micheli, "From Boolean Functions to Quantum Circuits: A Scalable Quantum Compilation Flow in C++," In 2021 *Design, Automation and Test in Europe Conference and Exhibition* (DATE), pp. 1044-1049, IEEE, 2021.

Fereshte Mozafari, Heinz Riener, and Giovanni De Micheli, "Uniform Quantum State Preparation: A Boolean Approach for Preparing Uniform Quantum States Efficiently and Precisely," In *International Workshop on Quantum Compilation* (IWQC), Cambridge, UK, September 2020.

Fereshte Mozafari, Mathias Soeken, Heinz Riener, and Giovanni De Micheli, "Automatic Uniform Quantum State Preparation Using Decision Diagrams," In 2020 50^{th} *International Symposium on Multiple-Valued Logic* (ISMVL), pp. 170-175, IEEE, 2020.

Mathias Soeken, Giulia Meuli, Bruno Schmitt, **Fereshte Mozafari**, Heinz Riener, and Giovanni De Micheli, "Boolean Satisfiability in Quantum Compilation," In *Philosophical Transactions of the Royal Society A*, 378.2164, pp. 20190161-20190161, 2020.

Mathias Soeken, **Fereshte Mozafari**, Bruno Schmitt, and Giovanni De Micheli, "Compiling Permutations for Superconducting QPUs," In 2019 *Design, Automation and Test in Europe Conference and Exhibition* (DATE), pp. 1349-1354. IEEE, 2019.

Fereshte Mozafari, Mathias Soeken, and Giovanni De Micheli,"Automatic Preparation of Uniform Quantum States Utilizing Boolean Functions," In *International Workshop on Logic Synthesis* (IWLS), Lausanne, Switzerland, 2019.

Mathias Soeken, Heinz Riener, Winston Haaswijk, Eleonora Testa, Bruno Schmitt, Giulia Meuli, **Fereshte Mozafari**, and Giovanni De Micheli, "The EPFL Logic Synthesis Libraries," In arXiv:1805.05121v3, June 2022, link.

All my publications are available in my Scholar: link

Selected Projects

Fall 2013	**Multi-Core System Project**, Implementing Crossword Puzzle Game on GPU. (Using CUDA)
Spring 2012	**Microprocessor Project**, Developing a regular expression parser on Cell BE Processor. (Using Parallel Programming)
Spring 2012	**VLSI Project**, Developing a standard Cell Library. (Using Tanner EDA Tools & Synopsis Design Compiler)
Spring 2011	**Digital Electronics Project**, Developing a high performance, low power Dynamics Random Access Memory (DRAM). (Using Hspice)
Fall 2011	**Digital System Design Project**, Implementing Tetris Game on Altera DE2 FPGA. (Using Verilog)
Spring 2010	**Computer Architecture Project**, Design and implementation of schematic Pipeline MIPS processor. (Using Quartus)
Fall 2010	**Advanced Programming Project**, Implementing digital circuits generator. (Using C++)

Teaching Experience

2019-2022	**École Polytechnique Fédérale de Lausanne (EPFL)** Instructor, EDA based Design Laboratory, by Dr. Vachoux Alain. (Spring 2019) Instructor, Test of VLSI Systems Laboratory, by Dr. Alexandre Schmid. (Spring 2021) Instructor, Computer Language Processing, by Prof. Viktor Kuncak. (Spring 2022)
2011-2015	**Sharif University of Technology** Instructor, Computer Architecture Laboratory. Teaching Assistant, Logic Circuits, Electrical Circuits, Advanced Programming, Digital Design, Data Structures, Fundamental of Programming.

References

Professor Giovanni De Micheli, EPFL, Switzerland, Homepage.
Assistant Professor Yuxiang Yang, University of Hong Kong, Hong Kong, Homepage.
Dr. Yao-Lung L. Fang, NVIDIA, United States, Homepage.