Bachelor Thesis
Academic year 2011

# Constructing a software framework for synthesizing high accuracy quantum circuits

Faculty of Environment and Information Studies
Keio University

Pham Tien Trung

Note: This is a revised version of the thesis, updated on 2012/6. The following changes have been made: sections 6.2 and 6.3 (p.40 - p.47) were reordered for comprehension; the figure 8.2 updated with more plot data.

# Abstract

This research concentrates on finding an efficient way to decompose an arbitrary quantum gate into sequences of quantum gates in a certain library set in order to construct a efficient quantum compiling program, which translates a program modelling a quantum mathematical-level algorithm into a diagram describing the machine-dependent quantum circuit which used to realize the algorithm. Our main contribution is enhancing the performance of the Solovay-Kitaev decomposition algorithm proposed by Christopher M. Dawson and Michael L A. Nielsen, by applying an efficient geometric searching technique as well as widening the search space for the basic approximation step in their algorithm.

These techniques give a large improvement. Compared to the previous naive approach, the length of the gate sequence required to approximate an arbitrary single qubit quantum gate of special unitary 2 matrix reduces by at least nearly a factor of three or even seven whereas the accuracy is still in an acceptable range (around $10^{-4}, 10^{-5}$ using the matrix trace distance function in the experiment) in an acceptable time (no longer than five minutes for one gate decomposition procedure in the experiment). Thus, some quantum subroutines can be executed nearly seven times as fast by compiling using our techniques. This result encouraged us to develop a complete language system: an input language, its compiler, and the corresponding output formats for simple quantum algorithms and subroutines, as the final outcome of this research.

Keywords:
1. Quantum algorithm 2. Compiler 3. Gate decomposition
4. Solovay-Kitaev theorem 5. Geometric search

Keio University,Faculty of Environment and Information Studies
Pham Tien Trung

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

For decades, the computer industry has been matching the predictions of Moore's law by doubling the number of transistors that can be placed inexpensively on an integrated circuit every two year. Recently, however, we are seeing this law is reaching its limit. Transistor size is hitting the size of atom. Therefore, due to problems such as heat over-generation and quantum effects in the micro-scale chip, the classical computer, (the architecture principles developed by Turing, Eckert, Mauchly and others and often attributed to von Neumann), is going to get stuck. One solution is, instead of avoiding quantum effects, we take advantage of them as a computation power, to form a entirely novel computer paradigm, called quantum computation.

Research in quantum computing so far has produced great success in both physical device construction; for example, quantum dot or ion trap machines; and quantum abstract subroutine and algorithms (mathematical model of the algorithms which works on a quantum computer) development; for example, quantum Fourier transform which is integrated in quantum integer factoring algorithm (Shor in [1]). However, accompanying the level of research in the two fields, the gap between quantum abstract algorithm research and quantum physical computer implementation is getting larger as algorithm research seems to grow much faster. This problem might lead to difficulties in constructing the structure of the realistic device such as a quantum circuit which has to implement a given quantum algorithm.

## 1.2 Research purpose

This research aims to contribute a useful software tool bridging the gap between quantum abstract algorithm research and research into the physical implementation of circuit, in order to complete the framework of designing a

circuit-based quantum computer. In detail, our research's aim is to provide a user-friendly software by which, an abstract quantum algorithm (given by the user, including mathematical elements for quantum computation: unitary matrices, which discussed in next chapter) can be easily automatically mapped into its implementing circuit model/diagram on a certain quantum computer, for which an instruction set and accuracy constraints are given. This work is the same as the job of a compiler on classical computer, which converts human-readable code into machine-readable code to execute an algorithm on a physical device. The only difference here is that the output cannot be run directly on the same the machine as the compiler (currently a classical computer).

The evaluation criteria for the outcome software of this research are:

1. Qualitative criteria
   (a) Whether the software works or not.
   (b) The software is easy to use or not.
   (c) The input and output are in standard form, understandable or not. This criterion will determine in part if the software can be widely used or not.
2. Quantitative criteria
   (a) How accurately the output circuit implements the input algorithm.
   (b) The cost of the output circuit for a given precision range in term of the amount of resource required. The lower the cost the better. This is the hardest part of this project, minimizing the quantum gates used in the output circuit diagram to approximate a quantum gate in the input program with an acceptable accuracy.

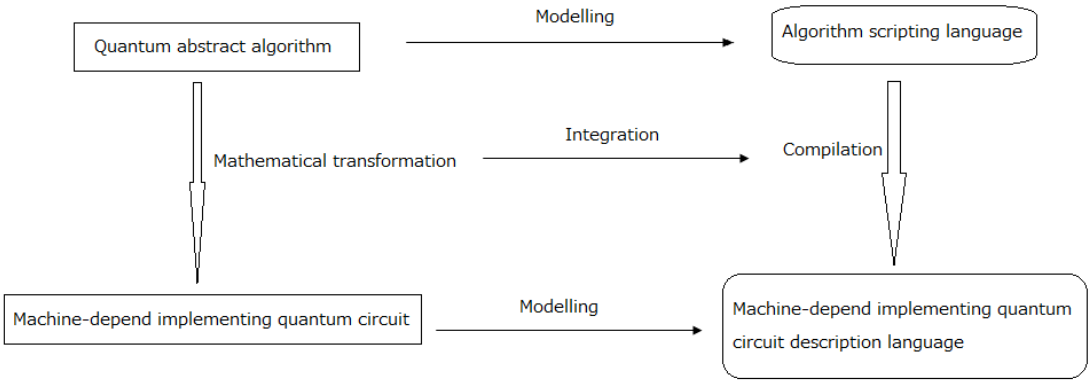
## 1.3 Proposed approach

As seen above there is a strong similarity between the goal of the research and the function of the compiler, giving a strong argument to view the problem in the scope of language and compilers. In other words, if we can model a quantum algorithm as the input language and a quantum physical device description as the output of a compilation process in which the mathematical transformation between them is automatically integrated, we will have a straightforward approach for the research project. This process also allows us to reflect results the mathematical transformation of quantum circuit into real world task. Figure 1.1 below illustrates this approach.

With this system, when a user wants to have a machine-dependent circuit diagram which implements an abstract quantum algorithm, he/she can follow these steps:

1. Transcribe the algorithm into the input language we designed.

Figure 1.1: Abstract system diagram




2. Use our system to find the implementing circuit diagram for the algorithm.
3. Bring the output circuit diagram to the physical device development team to build the realistic circuit.

## 1.4 Main contributions

This research so far makes the following contributions to the research field

- Created a simple but relatively general purpose language to model quantum abstract algorithms.
- Improved the performance of the Solovay-Kitaev single quantum gate decomposition procedure, which has the main role in the compilation process, by approximating an arbitrary abstract quantum gate by a sequence of fixed quantum gates in a given instruction set.

## 1.5 Thesis structure

This thesis is written in top down style from basic background to deep research topics, in order to give the audience not only the comprehension needed but also an illustration of the actual working progress.

The next chapter gives basic background, which is the material for the whole work. The $3^{rd}$ chapter present our language model of quantum algorithms, which gives a concrete image of the above approach idea. The $4^{th}$ chapter focuses on the compilation process and the output format of the system. The $5^{th}$ and $6^{th}$ chapters are written to describe the research

process, from encountering problems, to finding related research and then coming up with an original solution. The $7^{th}$ and $8^{th}$ chapters talk about implementation and evaluation of the research and we found that our solution can reduce the resource used for the circuit to reach an acceptable precision range. The last chapter is a conclusion with directions for future work.

# Chapter 2

# Background

## 2.1 Basic concepts of quantum information and computation

### 2.1.1 Quantum system and quantum states

The discovery of quantum mechanics in the early days of $20^{th}$ century brought a completely different image of the micro world: An uncertainty about the properties of a system intrinsically exist, no matter the measurement method and devices. A system which has the uncertainty of properties is called a quantum system.

From this definition, we have a mathematical concept to model the complete description of a physical system, including quantum system, called a quantum state. It is defined by a ray (more simple, a vector) in Hilbert space. A quantum state for the system called $\Psi$ is often written as a ket vector $|\Psi\rangle$, which can be expressed as

$$|\Psi\rangle = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ . \\ . \\ . \end{pmatrix} = \sum_{i=0} a_i |i\rangle \qquad a_i \in C$$

$|i\rangle$: Possible outcome state of one measurement on the system
$|a_i|^2$: Probability that the event $|i\rangle$ is the outcome of the measurement
$\Sigma |a_i|^2 = 1$

### 2.1.2 Quantum bit

Like the better-known classical bit, a quantum bit (qubit) is the atomic unit of information in the field of quantum computation. It is the instance

of a quantum state which has 2 basis states, call $|0\rangle$ and $|1\rangle$. Such a system is a two-level system in quantum mechanics. The *simultaneous* existence of more than one state in one qubit make the information in 1 qubit very large, compared to the classical bit has only one state in a certain moment. The presentation of a qubit depends on the quantum state of it.

1. Independent qubit

   If the qubit is itself a independent quantum state, called a *pure state*, it can be defined by a state vector. A popular mathematical formula of 1 qubit is the same as a 2 basis states quantum state

   $$|\Psi\rangle = a|0\rangle + b|1\rangle$$
   $$|a|^2 + |b|^2 = 1$$

   As a qubit is a two-level system, it also has the uncertain property. In contrast to classical bit which has a deterministic value 0 or 1 at any moment, a qubit is in the *both* states at the same time, or in a *superposition*. However this property of a qubit dissapears after conducting a measurement. We will return to this phenomenon more detail in the part of quantum measurement.

   A comprehensive way to visualize a qubit is using the Bloch sphere.

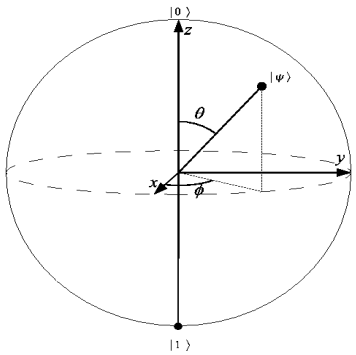



Figure 2.1: Qubit $|\Psi\rangle$ in a Bloch sphere with states basis
Image from [2]
$|\Psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle$

2. Multi-qubit quantum state

   When the quantum system consists of more than 1 independent qubits, its state can be calculated by the following formula:

   Take a quantum state made up from 2 qubits A and B, with the respective state vector $v_A$ and $v_B$. Call the state vector of the whole system $v_{AB}$, then

   $$v_{AB} = v_A \otimes v_B \tag{2.1}$$

$\otimes$ is the symbol for tensor product of 2 vectors.
For example, if

$$v_A = |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$v_B = |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

then

$$v_{AB} = v_A \otimes v_B = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

According to the equation 2.1 and the property of the tensor product, it is easy to see that a quantum state of $n$ qubits has a state vector of $2^n$ elements

3. Qubit in a mixed state

In some multi-qubit quantum states, the state vector of the whole system cannot be written as a tensor product. The state of each qubit which has entanglement with other qubit(s)-or in other words, as a result, is impossible to be written in state vector form (do not mess with the state of the whole system, which is always possible to be written in state vector notation). We say the qubits in this case are in a mixed state, although entanglement is not the only way to generate mixed state. The entire system, as the same time, is in a pure state though.

Example: In the 2-qubit state $|AB\rangle = \frac{1}{\sqrt{2}}(|0\rangle|1\rangle + |1\rangle|0\rangle)$, qubit $A$ and $B$ are in mixed state. We have lost information of each qubit if we do not have access to the other one, unlike in the case of independent qubit A and B.

Density matrix, which is computed from the pure state of the many-qubit system, is the method which solves the presentation problem. We introduce briefly about the meaning and computing method of density matrix

Given the 2-qubit state $|AB\rangle$

Density matrix to presenting the state of qubit A, $\rho_A$, is computed by the following equation

$$\rho_A = \sum_i p_i |\Psi_i\rangle\langle\Psi_i| \tag{2.2}$$

$|\Psi_i\rangle$: Possible states of qubit A
$p_i$: Probability of measured state $|\Psi_i\rangle$ in the corresponding basis measurement.

In example of the state $|AB\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$, we can see that the 2 possible states of qubit A after a measurement (see the part of measurement later in this section for detail) with the observable Z are 0 and 1 with respective probability 0.5 and 0.5. So the density matrix for qubit A (and the same for qubit B) is calculated as below:

$$\begin{aligned} \rho_A &= \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| \\ &= \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{aligned}$$

In fact, a density matrix $\rho$ can be expressed as a point on Bloch Sphere with axis as basis operators X ,Y and Z: the Pauli matrices as it can be rewritten by this formula

$$\begin{aligned} \rho &= aX + bY + cZ + I/2 \\ &= a\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} + b\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} + c\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}/2 \\ 1 &\geq a^2 + b^2 + c^2 \end{aligned}$$

($I$:identity matrix size 2)

### 2.1.3 Quantum state evolution

Over time, a closed quantum state has the ability to evolve into another state. This evolution obeys one of the most important postulates in quantum mechanics, the *Schrödinger equation*

$$i\hbar\frac{d|\Psi\rangle}{dt} = H|\Psi\rangle \tag{2.3}$$

where $\hbar$ is Plank's constant divided by $2\pi$ and $H$ is a fixed Hermitian operator called the *Hamiltonian* of the closed system.
From this equation, we have the following equation apply with any time interval $[t_1, t_2]$

$$|\Psi(t_2)\rangle = exp\left[\frac{-iH(t_2 - t_1)}{\hbar}\right]|\Psi(t_1)\rangle \tag{2.4}$$

Define the generator $U$ as

$$U(t_1, t_2) \equiv exp\left[\frac{-iH(t_2 - t_1)}{\hbar}\right] \tag{2.5}$$

Representing the evolution of the system from $|\Psi(t_1)\rangle$ to $|\Psi(t_2)\rangle$, we realize that as $U$ is a unitary matrix, the quantum state is evolved by *unitary operator*.

### 2.1.4 Quantum entanglement

Quantum entanglement can result in the mixed state mentioned above. Entanglement is the phenomenon when different particles interact in such a way that the state of the whole system can not be decomposed into separate states of each particle. This means that their measurement outcomes can maintain their property of uncertainty but be correlated (in classical systems, correlation may also occur but in quantum mechanics is much stronger). This correlation is called entanglement. Entanglement is not restricted to a pair of qubits but to can occur among any number of qubits, called entangled qubits.

### 2.1.5 Quantum measurement

By measuring a quantum system, we cause it to interact with the *external* world, to have an outcome. The outcome is probabilistic due to the superposition of quantum states. However, after measurement, the quantum system is no longer closed, and the superposition also collapsed into one deterministic states, so that we will get the same outcome if we continue the same measurements later.

In the mathematical model, measurement is also the linear operations of *measurement operators* on the quantum system state vector.

## 2.2 Quantum circuit model

1. Quantum gate

   In the classical paradigm, a logic gate is the unit performing processing of information, converting it from one form to another. Binary gates like AND, OR, NAND, NOR conducting binary operations are the basis of any current computer processor.

   In the quantum computation paradigm, in the sense of processing information (hidden in quantum state), we also have the concept of " quantum gate ". As discussed in the section of quantum state evolution, it is straightforward that unitary operator which change a quantum state into another can be considered as " quantum gate " in term of quantum information processing unit. Like the truth table for classical gate, unitary matrix is the specification of a quantum gate.

   Example: Quantum Not Gate, which convert qubit $|0\rangle$ into qubit $|1\rangle$ and vice versa, is specified by the matrix $X$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.6}$$

Operating on $|0\rangle$, we see

$$X|0\rangle = X \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle \tag{2.7}$$

and operating on $|1\rangle$ we see

$$X|1\rangle = X \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle \tag{2.8}$$

If a quantum gate is operating on an $n$-qubits quantum system, because the state vector of the quantum state has $2^n$ elements, the size of a matrix that can operate with this state vector is $2^n \times 2^n$, result in the size of the specifying unitary matrix of the quantum gate is also $2^n$. See for example the CNOT-gate (A.1) and Toffoli(A.2) gate in the Appendix 1 for instance

Quantum gates which operate on 1 qubit are called single qubit gate.

2. Quantum circuit

   As in the classical paradigm, in quantum computation, there is the concept of a circuit: quantum circuit. It contains connected quantum gates, give the circuit the ability of performing a sequence of quantum information process along time; and often contains measurement devices to read out the result as the outcome of measuring a quantum object and/or classical controlling part.

   However, compare to classical circuit, there exist the following points to discriminate quantum computation

   (a) Gate connecting " wire "
   In classical circuit, gates are normally connected by wires to propagate the output of one gate to the input of another. The term " wire " in a quantum circuit refers to other things, perhaps the passage of time (in many cases) or perhaps the movement of photon, from one location to another.

   (b) No Independent copies
   Copying a qubit is not allowed in quantum computation, shown in the " No cloning theorem " (Wooters[3]). It completely different in classical circuit, where we can copy the output of one gate for the input of several others.

   (c) Output calculation
   In general, the output of a quantum circuit, if in the form of a state vector, can be calculated from the input by evolving the input state vector by quantum gates of the circuit in time order, one after another.

3. Example

The quantum teleportation (teleporting the qubit $|q_0\rangle$) circuit has the

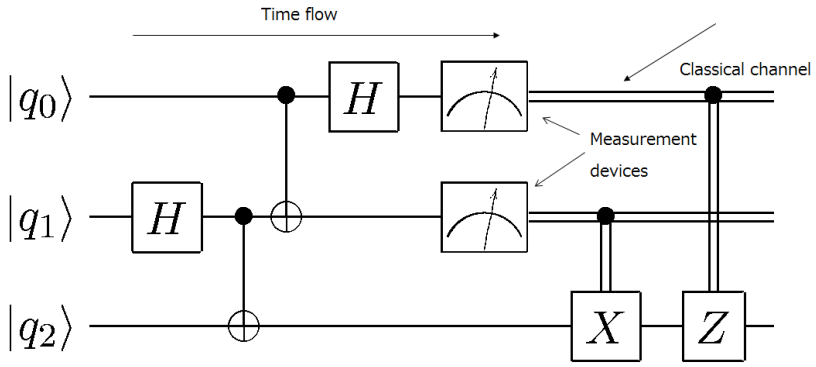



Figure 2.2: Quantum teleportation circuit
Image from http://www.media.mit.edu/quanta/qasm2circ/

input of 3 qubits $|q_0\rangle$,$|q_1\rangle$,$|q_2\rangle$, in which $|q_1\rangle$ and $|q_2\rangle$ is initialize to $|0\rangle$. It deploys quantum gates of $H$,$Z$,$X$, Control-Not;measurement devices and a classical " bus " to transfer measurement results as classical signals. See the figure 2.2 for conception.

4. Universal quantum gates

   In the classical computation paradigm, there are sets of gates (e.g. AND, OR, NOT) that can be used to compute any arbitrary classical function. In quantum computation, there are also gate sets which can be used to compute any quantum evolution, called universal gate set. In other words, any quantum gate is constructable by the gates in a universal set. The term " constructable " means any unitary operation can be approximated to an arbitrary accuracy by a quantum circuit involving only gates in the universal set.

   Some examples of the known universal gates sets: Set of single qubit gates and controlled-NOT gate, set of Hadamard gate, phase gate, controlled-NOT gate and Toffoli gates[4]. See more detail about this issue in chapter 5 :" Key technique problem " and chapter 6: " Solution " in this thesis.

5. Physical implementation

   There are five requirements for the implementation of quantum computation, almost focused on generation, preservation, evolution and measurement of qubit, summarized by Vincenzo in[5]

   - Qubit presentation
   - Controllable unitary evolution (gate implementation using universal set)
   - Initialization of qubit state
   - Measurement of final state
   - Long coherence time (avoiding decoherence phenomenon)

Some systems that can meet these requirements are Cavity quantum electrodynamics, Trapped Ion and Nuclear spin. Surveying them is not in the scope of this thesis.

## 2.3 Language and compiler

This section gives the readers fundamental background about the field of grammar, language, and compiler. For more detail technical reading, please refer lecture notes in [6] (in Japanese) and [7], which discuss these fields in a high level.

### 2.3.1 Context-free grammar

**Definition**

We begin this discussion from the concept of a phrased-structured grammar (PSG). Call $G$ a instance of PSG. $G$ is defined as below.

$$\begin{aligned} G &= (V_N, V_T, P, S) \\ V_N &: \text{Set of non-terminal symbols.} \\ V_T &: \text{Set of terminal symbol.} \\ S &: \text{Start symbol. } S \in V_N \\ P &: \text{Rewrite rule or production.} \end{aligned}$$

More about rewrite rule or production of a PSG

$$\begin{aligned} V &= V_N \cap V_T \\ P &\subseteq V^* V_N V^* \times V^* \end{aligned}$$

If $(u, v) \in P$, write $u \to v$

A context-free grammar is the type 2 of PSG, with more constraint for the rewrite rule $P$:$P \subseteq V_N \times V^*$. In simple words, it can be said that a symbol in the non-terminal set has its own rewrite rule, independent on other ones; the reason which gives this type of PSG the name " Context-free grammar ". For instance, given $G1$ a context-free grammar.

$$\begin{aligned} G1 &= (V_N, V_T, P, S) \\ V_N &: \{s, u\} \\ V_T &: \{A, B, \epsilon\} \\ S &: s \\ P &: \{s \to uB, u \to AB|\epsilon|B\} \end{aligned}$$

### Phrase-structured language

Each PSG specifies a its own language. This language consists of " sentences ", which can be generated from the start symbol of the grammar by applying the rewrite rule. The language of PSG $G$ is denoted $L(G)$ and defined as below:

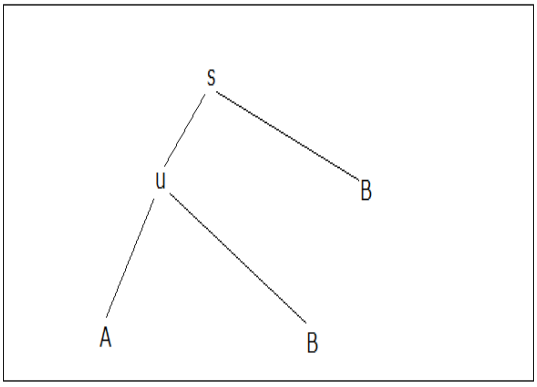
$$L(G) = \{x \in V_T^* | S \Rightarrow_G^* x\} \quad (2.9)$$

$$\Rightarrow_G^* : \text{Transitive closure of G} \quad (2.10)$$

### Derivation tree

A derivation tree is a way to graphically express how a context-free grammar generate a sentence in its language by using tree structure. Tree root is the start symbol ($S$) of the grammar. Each tree node is a left-hand side of a production ($P$) in the grammar, i.e a non-terminal symbol, and child nodes are the right-hand side of this production. The leaves of a derivation tree can be considered as terminal symbols of the context-free grammar. Figure 2.3 is the derivation tree of the above grammar instance $G1$ which derives the string $ABB$ from the start symbol.

Figure 2.3: Derivation Tree Example




### Parsing

The problem of parsing can be stated as follow: Given a context-free grammar $G$ and a string $w$, how do we know if $w \in L(G)$ and if so how do we indicate the derivation tree or the sequence of production rules that produce $w$. A program which does the parsing procedure is called a parser.

There are 2 main approach of constructing a parser: The top-down approach and the bottom-up approach. A top-down parser, like JavaCC, starts

from the start symbol at the top of the parse tree and works downward, driving productions in forward order until it generate the terminal leaves exactly. A bottom-up parser, like YACC/BISON, starts from the string of terminals itself and builds from the leaves upward, working backwards to the start symbol by applying the productions in reverse. Both approaches have their strengths and weakness. Three well-known representatives of the 2 approaches are recursion descent (top-down approach), LL(k) parser and LR(k) parser (bottom-up approach).

- Recursion descent parser
  The top-down parser which eliminates recursive derivation in the parsed derivation tree.
- LL(k) parser
  Traverse the input string **l**eft-to-right, using $k$ tokens (symbols) of looking ahead to generate a **l**eftmost derivation.
- LR(k) parser
  Traverse the input string **l**eft-to-right,using $k$ tokens (symbols) of looking ahead to generate a **r**ightmost derivation.

### 2.3.2 Programming language and Compiler design

The work of designing a programming language consists of defining a context-free grammar, which can be transformed into formats understandable and executable for the computer. Some important points in designing a programming language:

- For a programming language grammar, the terminal symbols, which can be considered as the vocabulary of the language, are often regular expressions.
- Should be designed to limit the ambiguous cases (the parser can parse a string-program into various derivation tree, leading to making the computer " misunderstand " the programmer's demand).

The compiler is a computer program which takes a program written in a programming language as the input, process it, and compose an output, usually machine-executable instructions. Fundamental components of a popular compiler

- Scanner (lexical analyser):
  Recognizes and translates a sequence of characters in the input program into a corresponding sequence of tokens
- Parsers (syntax, semantic analyser):
  Consumes the tokens generates by the scanner, then generates the program structure (derivation tree) by applying the language grammar with a certain approach (LL(k),LR(k),...)
- Code generator:
  Uses the output of the parser to generate target code (can be understood as another language).

- Code Optimizer:
  Optimizes the output code, for instance, to reduce the resource required by the output code by removing redundant code.

## 2.4 Chapter summary

This chapter discussed about the main background required for this graduate project, which covers not only quantum computation theory but also very well-known fields in current computer science like language, compiler and grammar, in order to give readers a conception of what is going next in this thesis . In the next chapter, we are talking about the first step approaching this research purpose: modelling the quantum algorithm by a language called QAS (quantum algorithm scripting) language.

# Chapter 3

# QAS-Quantum algorithm scripting language

## 3.1 Why QAS?

This chapter begins from the point: Why can a programming language be an answer for reaching the research purpose of creating a connection between quantum abstract algorithm research and quantum physical implementation circuit research?

Currently, because of the popularity of the quantum circuit model in quantum computation research, quantum algorithms are often modelled by a quantum circuit diagram .This describing way works well in pure algorithm research thanks to its comprehensive and illustrative power; however, when entering the implementation phase on a certain quantum computer (when one is built), we encounter two big problems. Firstly, there is the difficulty of making a standard, computer-understandable circuit diagram without a tailored graphical tool. Secondly, the more serious one, it is the problem of how to automatically convert an algorithm circuit diagram into a logical-equivalent machine-dependent specification which, in many case, has completely different circuit elements.

A plausible solution is modelling the quantum algorithm (the circuit diagram in the case of quantum circuit model) in the form of a programming language, which can be both comprehensible to a human, and flexibly manipulated by a computer program with the help of a compiler.

## 3.2 Language introduction

The Quantum algorithm scripting language (QAS) is a programming language designed for researchers who want to create a quantum program by

specifying a high-level quantum circuit model computation, i.e. designing a circuit with input qubits and their operating quantum gates operating on them in mathematical expression, for a desired output. A program in this language should be later used as the input of the accompanied compiler to synthesise a lower-level description of a quantum circuit which can be considered as " machine-code " on a certain quantum computer. This lower-level description can be called the output language, which is introduced in next chapter, but it has almost the same structure and grammar as the input program in QAS. The remaining work of making this language become executable on a practical quantum computer is beyond the scope of my work.

## 3.3 Language features

QAS is designed as a simple scripting language, without a main function or difficult control flows, so that even programming inexperienced users, like quantum physics researcher can easily use it for their own purpose. On the other hand, this language is dedicated for "quantum programmers", who are researching quantum algorithm, so it takes from quantum computation field the concepts of qubit and quantum gate (quantum operator) as basic elements of a program.

In the source code of a program, qubits are treated as variables, whereas quantum gates (in current version, up to 2-qubit gate or 2-control-1-target gate) iare considered as the only operators in this language, though also treated as variable at the same time. More details are mentioned in the next parts of this chapter

Lastly, though simple, QAS is nevertheless a programming language, it contains some popular features of an ordinary programming language like arrays of variables and " for loop " in order to save time for iterated code writing, and to make the program more clearly.

## 3.4 Program structure

Although QAS is a scripting language, implying that it does not have strict structure requirement like a main function, a standard program in QAS is recommended to consist of 3 parts:

1. Qubit variables and classical variables declaration
   Declares qubits register and classical variables (used for loop and array manipulation inside the program to save time for writing iterating code, not to be confused with classical control in quantum circuit)

with names. Name syntax for qubit variables and classical variables are distinguished. See detail in the next section.

2. Quantum gate definition declaration
   Declares quantum gates (operators) with names, operated qubits and specifying mathematical parameters. For example, a single qubit quantum gate is specified by parameters for it s equivalent unitary matrix: Angle of rotations around y and z gates, and phase angle if required. More detail is mentioned in Chapter 2, the background about quantum computation and in the next section, language specification.
3. Operating blocks
   In quantum computation, a strict operation schedule is required because one qubit in one time step can only interact with one operator in general. Therefore, in the main part of a program, describing interactions between qubits and quantum gates, the programmer has to divides operations into time steps in their code. The compiler will automatically check the validation of the programmer-define scheduling strategy as a part of semantic error handling procedure.

Example of structure in a sample program (The code lines in this program will be explained in next section)

```
/*This is a comment*/
QVAR r1,r2,r3,q[];
CVAR $i;
/*-------End of variable declaration part-------*/
/*-----Begin of gate definition declaration part-----*/
DEFINE mygate 0.6,0.8,0.9~1.2;
DEFINE mygate2 0.4,0.9,1.6;
/*-------End of gate definition declaration part-------*/
/*-------Begin of time step 1 operation block-------*/
STEP:
BEGIN
GATE CONTROL CONTROL mygate     r1,r2,r3;
END
/*-------End of time step 1 operation block-------*/
/*-----Begin of time step 2 operation block-----*/
STEP:
BEGIN
GATE CONTROL mygate r1,r2;
GATE mygate2 r3;
END
/*-------End of time step 2 operation block-------*/
/*-----Begin of a 4 times iterated operation block-----*/
```

```
ITERATE=4:
/*-------Block operations definitions-------*/
STEP:
BEGIN
GATE mygate2 r3;
GATE mygate2 r1;
FOR $i=1:3
BEGIN
GATE mygate2 q[$i];
END
END
/*-----End of 4 times iterated  operation block-----*/
```

## 3.5 Language specification

In fact, the context free grammar used in constructing programming language does not explicitly distinct notation from grammar. The below explanation is just for clarity. For the theoretical explanation of terminology " Terminal symbol ", " Non-terminal symbol ", " grammar ", refer Chapter 2:" Background ", Section 2:" Language and compilation "

### 3.5.1 Notation

*The concepts used in this section are mentioned in chapter 2, section "Language and compiler"*

#### Terminal Symbols

Terminal symbols-explicit vocabulary of the language, in term of any program in the language only contains words in the set of terminal-symbols.In the tree structure of a program written in the language, instances of terminal symbols set are leaf nodes.The terminal symbols in QAS are defined in the table 3.1

Table 3.1: Terminal symbols-explicit vocabulary of the language

| Name | Regular expression |
|---|---|
| $\langle VALUE \rangle$ | (["-"])*(["0"-"9"])+ \| (["-"])*(["0"-"9"])+"."(["0"-"9"])+ |
| $\langle VAR \rangle$ | (["a"-"z"])+(["0"-"9"])*("["("$"(["a"-"z"])+(["0"-"9"])*)¿']")? |
| $\langle CVAR \rangle$ | " $"(["a"-"z"])+(["0"-"9"])* |
| $\langle OPERATOR \rangle$ | "GATE" |
| $\langle SEPARATE \rangle$ | ([" "])+ |
| $\langle TYPE \rangle$ | "QVAR" |
| $\langle CTYPE \rangle$ | "CVAR" |
| $\langle DEFINE \rangle$ | "DEFINE" |
| $\langle CONTROL \rangle$ | "CONTROL" |
| $\langle STEP \rangle$ | "STEP:" |
| $\langle BEGIN \rangle$ | "BEGIN" |
| $\langle END \rangle$ | "END" |
| $\langle FOR \rangle$ | "FOR" |
| $\langle ITERATE \rangle$ | "ITERATE" |

Semantic in the language, define the human-understandable meaning of each terminal symbol in the language. This also can not disappear in any language tutorial.

| Name | Representation for |
|---|---|
| $\langle VALUE \rangle$ | Real value, specifying gate parameter or classical variables assigned value |
| $\langle VAR \rangle$ | A qubit or gate name |
| $\langle CVAR \rangle$ | A classical variable name |
| $\langle OPERATOR \rangle$ | Prefix of a operation (interaction of qubit and quantum gate) |
| $\langle SEPARATE \rangle$ | Separate between token |
| $\langle TYPE \rangle$ | Type of qubit variable |
| $\langle CTYPE \rangle$ | Type of classical variable |
| $\langle DEFINE \rangle$ | Definition of a quantum gate |
| $\langle CONTROL \rangle$ | Controlled operator |
| $\langle STEP \rangle$ | Beginning of a time step of operations |
| $\langle BEGIN \rangle$ | Entering of a time step of operations (always follow STEP token) |
| $\langle END \rangle$ | End of a time step of operations |
| $\langle FOR \rangle$ | Beginning of a loop applying on classical variable |
| $\langle ITERATE \rangle$ | Beginning of a loop for operations block |

### Non-Terminal Symbols

Non-Terminal Symbols-hidden structures of the language. In the tree structure of a program written in the language, instances of non-terminal symbols

set are non-leaf node.3.2

Table 3.2: Non-Terminal Symbols-hidden structures of the language

| Name | Semantic |
|---|---|
| $\langle compound - of - stmt \rangle$ | The whole program |
| $\langle expr \rangle$ | A line of variable declaration or gate definition |
| $\langle block \rangle$ | A block of operations on one time step |
| $\langle loop \rangle$ | A loop applying on classical variable |
| $\langle range \rangle$ | Range for looping classical variable (min value-max value) |
| $\langle declare \rangle$ | A declaration line |
| $\langle multi - var \rangle$ | Multi quantum (qubit) variables declaration |
| $\langle var \rangle$ | Indexed Qubit (used in array) |
| $\langle multi - cvar \rangle$ | Multi classical variables declaration |
| $\langle define \rangle$ | Definition of a quantum gate with 3 or 4 parameter |
| $\langle operate \rangle$ | An operation (of gates acting on qubits) |
| $\langle block - iterate \rangle$ | Number of iterations of a operation block in time flow |

### 3.5.2 Grammar

The grammar of a language is the rule to product sequence of symbols from a non-terminal symbol in the vocabulary. The grammar is used to construct a language, as well as to parse a program written in this language into a structured form. Therefore it is the main of any language.3.3

Table 3.3: Language grammar

| Non-terminal symbol | Production |
|---|---|
| $\langle compound-of-stmt \rangle$ | $((\text{“}\langle expr \rangle\,\text{;”}) \mid \langle block \rangle)*$ |
| $\langle expr \rangle$ | $\langle declare \rangle \mid \langle define \rangle$ |
| $\langle block \rangle$ | $(\langle block-iterate \rangle)*$<br>$\langle STEP \rangle\,\langle BEGIN \rangle\,(\langle operate \rangle\,\text{“;”} \mid \langle loop \rangle)*$<br>$< END >$ |
| $\langle loop \rangle$ | $\langle FOR \rangle\ \langle SEPARATE \rangle\ \langle CVAR \rangle\text{“=”}\langle range \rangle$<br>$\langle BEGIN \rangle\ (\langle operate \rangle\,\text{“;”})*\ \langle END \rangle$ |
| $\langle range \rangle$ | $\langle VALUE \rangle\ \text{“:”}\langle VALUE \rangle$ |
| $\langle declare \rangle$ | $\langle TYPE \rangle\,\langle SEPARATE \rangle\,\langle VAR \rangle$<br>$(\langle multi-var \rangle)*$<br>$\mid \langle CTYPE \rangle\,\langle SEPARATE \rangle\,\langle CVAR \rangle$<br>$(\langle multi-var \rangle)*$ |
| $\langle multi-var \rangle$ | $\text{“ ,”}\langle VAR \rangle$ |
| $\langle var \rangle$ | $\langle PREFIX \rangle\,(index\,())?$ |
| $\langle multi-cvar \rangle$ | $\text{“ ,”}\langle CVAR \rangle$ |
| $\langle define \rangle$ | $\langle DEFINE \rangle\,\langle SEPARATE \rangle\,\langle VAR \rangle$<br>$\langle VALUE \rangle\ \text{“,”}\ \langle VALUE \rangle\ \text{“,”}\ \langle VALUE \rangle$<br>$(\text{“}\sim\text{”}\ \langle VALUE \rangle)?$ |
| $\langle operate \rangle$ | $\langle OPERATOR \rangle\,\langle SEPERATE \rangle$<br>$(LOOKAHEAD(3)$<br>$\langle CONTROL \rangle\,\langle SEPERATE \rangle$<br>$\langle VAR \rangle\,\langle SEPERATE \rangle\,\langle VAR \rangle\ \text{“,”}\ \langle VAR \rangle$<br>$\mid \langle CONTROL \rangle\,\langle SEPERATE \rangle$<br>$\langle CONTROL \rangle\,\langle SEPERATE \rangle\,\langle VAR \rangle$<br>$\langle SEPERATE \rangle\,\langle VAR \rangle\ \text{“,”}\ \langle VAR \rangle\ \text{“,”}\ \langle VAR \rangle$<br>$\mid \langle VAR \rangle\ \langle SEPERATE \rangle\,\langle VAR \rangle)$ |
| $\langle block-iterate \rangle$ | $\langle ITERATE \rangle\ \text{“}=\text{”}\ \langle VALUE \rangle\ \text{“}:\text{”}$ |

### 3.5.3 Sample Program Analysis

```
QVAR r1,r2,r3,q[];
//Declare 3 qubit variables r1,r2,r3 and
//an array of qubit variables, called q
CVAR $i;
//Declare classical variable called $i

DEFINE mygate 0.6,0.8,0.9~1.2;
//Define quantum gate(quantum operator) called mygate
//by 4 parameter specifying unitary matrix of mygate
//Phase and z,y,z axis rotation angles respectively:
```

```
//1.2,0.6,0.8,0.9

DEFINE mygate2 0.4,0.9,1.6;
//Define quantum gate (quantum operator) called mygate
//by 3 parameter specifying unitary matrix of mygate
//Phase and z,y,z axis rotation angles respectively:
//0.0,0.4,0.9,1.6

STEP:
//Start a time step
BEGIN
GATE CONTROL CONTROL mygate r1,r2,r3;
//Operate mygate on qubit r3 with 2 controlling qubit
//r1 and r2
END

STEP:
//Start a time step
BEGIN
GATE CONTROL mygate r1,r2;
//Operate mygate on qubit r2 with controlling qubit r1
GATE mygate2 r3;
//Operate mygate2 on qubit r3
END

ITERATE=4:
//Start a 4 times iteration of operations block
STEP:
BEGIN
GATE mygate2 r3;
//Operate mygate2 on qubit r3
GATE mygate2 r1;
//Operate mygate2 on qubit r1
FOR $i=1:3
BEGIN
GATE mygate2 q[$i];
END
//Operate mygate on q[1],q[2],q[3]
END
```

## 3.6 Program compilation overview

For the compilation procedure, which translates QAS into a form of quantum machine code with defined instructions set, I am using the Javacc (Java Compiler Compiler) with JTB (Java Tree Builder) utility to automatically generate a compiler with parser and lexical analyser for the language. The final "compiler" is a runnable Java program, which takes the QAS source file as the input, to produce output code. More about compiling process is discussed in the next chapter.
Usage in my development environment

1. JTB form grammar file generation using jtb

2. Lexer and Parser generation using javacc

3. Runnable file generation

4. Compile/ Output source generation

## 3.7 Chapter summary

This chapter has given a comprehension about QAS (Quantum Circuit Scripting) language, from the reason for establishing to the specification. In short, it is a language that can be easily used to encode a simple quantum algorithm as the description of its implementing circuit in quantum circuit computation model. After compiling process, QAS source code can be converted into more machine-dependent circuit description source code, which requires only fixed gate as instruction set. This language (say output language) is being talked in next chapter, however its grammar and structure are almost the same as QAS.

# Chapter 4

# Compiler construction process and output format

After constructing the input language grammar, the next step is to create a processing tool for it, called the compiler. This converts a program in this language into a data structure understandable by the computer, and then processes this data structure into the desired other data structure: in this case, the data structure depicts the machine-dependent quantum circuit, then continue to output this result data structure into the output language.

## 4.1 QAS Program compilation basic steps

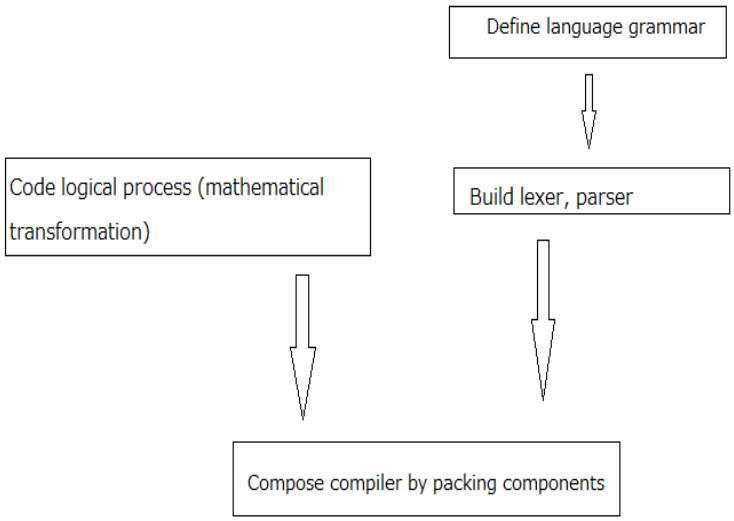
1. Generating the lexical analyser (lexer) to deal with the terminal symbol in the program
2. Generating the parser to parse the program structure
3. Creating the program (call the program A) to process the data structure extracted from the input program. Instead of generating executable code (assembly etc.) like an ordinary compiler, in this work, A can be thought as the mathematical transformation of abstract quantum algorithm to its machine-dependent implementing circuit description. This step is the most important work in this research.
4. Embedding all of those above into a final program:the compiler.

An overview of these steps is depicted in figure 4.1

## 4.2 QAS compiler construction methodology

In order to keep the software development process clear and flexible, we conduct the work in the first two steps independently of the work of step 3. The code for step 3 has to be written with as few as possible dependencies on the parser and lexer or any language-related features. In this way, we can

Figure 4.1: Intuition of work flow




easily integrate the transformation algorithm into the compilation of other languages, if any exists.

## 4.3 Tools

Considering the desired methodology, the following tools are chosen to create the software

1. For the lexer and parser generation work
   Use the Javacc (Java Compiler Compiler) utility of the Unix Operating System, powered by JTB (Java Tree Builder) http://www.cs.ucla.edu/ pals-berg/jtb/. The Javacc is very simple compiler making tool, however, along with JTB, it has the ability of parsing complex languages with loops, array and many other features.
   JTB is created in a great way: it allows the least confusion of logic processing and language components by applying the visitor design pattern explained in Doug Orleans and Karl Lieberherr in [8], which actually divide the work of a program into independent 2 parts: traversal strategy and visitor behaviour, those suit for the work of parsing program structure and process the components of the program structure.
2. For the algorithm processing work
   Use Java Language with an octave-bridge for some cases to calculate

unusual mathematical operations.

## 4.4 Output format

The output format of the compilation process, as actually stems from the logical data structure of a quantum circuit, can be customized by the user. It can be a program with the same grammar as the input QAS program but only uses gates in the library, or an other format like the AQUA language[9], Latex source code etc., depending on the user demand. The output should have the ability of being an input for another process, like the AQUA language[9] can be exported to circuit image file, so in this work, we have exported the output as a program with the same language as QAS with a fixed gate library. In this case, our tool should be considered as a preprocessor. In the future, to meet more users' requirements, we are considering adding a feature to the compiler allowing the user to define their output format.

# Chapter 5

# Key technical problems

## 5.1 The concept of quantum gate decomposition

In chapter 2, when discussing the quantum circuit model, we showed that a quantum circuit is constructed from quantum gates acting on quantum bits. We also indicated that the work of a quantum gate is changing the state of quantum bits by applying a unitary transformation on the state vector of the input quantum bits. Therefore, a quantum gate is characterized by a unitary matrix.

However, when creating a quantum computer, for its building block: quantum gates, due to physical implementation constraints as well as some purpose like keeping the circuit resistant to error, we are not always able to have an arbitrary quantum gate for a desired computation[4]. Instead, we use a fixed set of quantum gates as a universal library (set) of elementary gates [10]. In other words, any arbitrary quantum gate can be approximated to arbitrary accuracy by elementary gates in the universal library. This approximation process is called **quantum gate decomposition**. In a mathematical view, this approximation is equivalent to the approximation of an arbitrary unitary matrix by a product of elementary unitary matrices in a fixed set. There is a similarity in the classical computer when any computation is actually executed by instructions in a fixed instruction set.

For instance, we list some universal sets of elementary gates[4]:

1. Set of two-level unitary gates are universal.
2. Set of single qubit gates and CNOT gate is universal.
3. Set of Hadamard, phase, CNOT and $\pi/8$ gates is universal.

In the next chapter, we give the explanation of each of these universal set as well as their position in this research.

## 5.2 Quantum gate decomposition accuracy measurement method

There are many ways to estimate the extent to which two unitary matrices (and equivalently two quantum gate) approximate each other. One way is using an error function call E to measure the largest bias when operating on vectors (pure states (1 )) the 2 unitary matrix approximated U, V. The physical meaning of this approximation estimating method is, the lower the value of $E(U,V)$, the more similar the measured outcome probability distribution of the result state when U and V operate with one input state:

$$E(U,V) = \max_{|\omega\rangle} \|(U-V)|\omega\rangle\| \tag{5.1}$$

$$|\omega\rangle : \text{Pure state that can be multiplied by U and V} \tag{5.2}$$

Define the trace distance function of 2 unitary matrices U and V

$$D(U,V) = Trace\sqrt{(U-V)^\dagger(U-V)} \tag{5.3}$$

It was proven in [4] that with any unitary matrix U and V can be written as a rotation matrix, i.e. $U = Rotate_{\overrightarrow{n}}(\phi)$,$V = Rotate_{\overrightarrow{m}}(\theta)$ ($U$ is the unitary transform can be written as the rotation an angle of $\phi$ around the axis vector $\overrightarrow{n}$ and $V$ is the unitary transform can be written as the rotation an angle of $\theta$ around the axis vector $\overrightarrow{m}$). We have $D(U,V) = 2E(U,V)$. This condition is true for any unitary matrices U, V up to global phase. So it does not matter if we use the function $D(U,V)$ as the distance function to measure how different the 2 unitary matrices are. Of course $D(U,V)$ much easier to calculate than $E(U,V)$. In this research, we use the trace distance function $D(U,V)$ as the distance function. $distance(U,V) \leftarrow D(U,V)$

In the case of gate decomposition by approximating the unitary matrix $U$ by a sequence of unitary matrices called $V_1, V_2 ... V_n$ in the universal set, the accuracy of approximation is calculated as follows (smaller accuracy implies a better approximation):

$$Accuracy = distance(U, \prod_{i=1}^{n} V_i) \tag{5.4}$$

For example, you are seeing single-qubit quantum gate X with the corresponding unitary matrix $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, we can decompose it into a sequence of 2 gates H and T with the corresponding unitary matrices as below

$$
\begin{aligned}
H &= \frac{1}{\sqrt{2}} \begin{pmatrix} -i & -i \\ -i & i \end{pmatrix} \\
T &= \begin{pmatrix} e^{-\frac{i\pi}{8}} & 0 \\ 0 & e^{\frac{i\pi}{8}} \end{pmatrix} \\
HT &\approx \begin{pmatrix} -0.27060 - 0.65328i & 0.27060 - 0.65328i \\ 0.27060 - 0.65328i & 0.27060 + 0.65328i \end{pmatrix}
\end{aligned}
$$

So

$$
\begin{aligned}
distance(X, HT) &= Trace\sqrt{(X - HT)^{\dagger}(X - HT)} \\
&= Trace \begin{pmatrix} 1.49625 & 0.21045 - 0.50807i \\ 0.21045 + 0.50807i & 1.07535 \end{pmatrix} \\
&= 2.571\text{-a relatively poor approximation}
\end{aligned}
$$

## 5.3 Quantum gate decomposition problem in this research

In chapter 1, we mentioned that there is a mathematical transformation from a high-level algorithm description into an output of a machine-dependent circuit diagram description integrated into compilation process. As actually a machine-dependent circuit is, according to the previous section, specified by a set (library) of elementary gates and an threshold of precision for any gate approximation, whereas an abstract algorithm description is made from arbitrary quantum gates, this transformation turns out to be nothing but gate decomposition, or in the mathematical view, unitary matrix decomposition.

Based on the evaluation criterion, the problems we have to solve in this research are:

1. Find decomposition methods bringing the accuracy lower than an acceptable threshold given by user for each arbitrary quantum gate in the input algorithm.
2. Among these methods find the one that requires the least resources for the output circuit. To measure the resourcess, in this research we use the number of quantum gates used in the output circuit.

In the scope of this research, we do not have the ambition of decomposing any arbitrary unitary matrices. Instead, we focus on the step of decompos-

ing unitary matrices corresponding to 1,2 -controlled gates and especially unitary matrices corresponding to one qubit gates. Because all unitary operations on arbitrarily multi bits gate can be expressed as compositions of these gates[11], our program does not lose generality as a result.

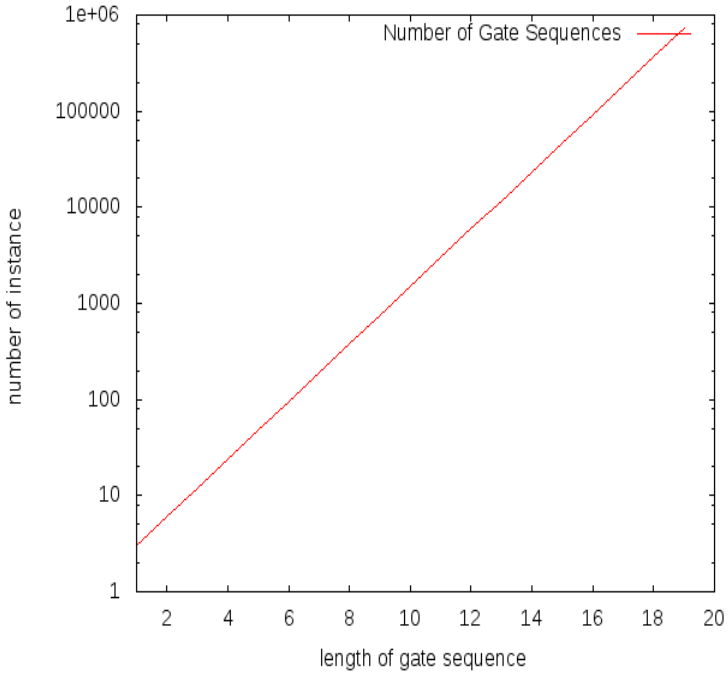
## 5.4 Prominent difficulties definition

In this research, as it focuses only on decomposing unitary matrices for 1 and 2 -controlled gates and especially unitary matrices corresponding to one qubit gates, the work can explicitly divided into 2 steps.

1. 1 and 2 -controlled gates decomposition step
   This problem is solved by using the same method as in [11] for this stage. Appendix 2 discusses this technique.
2. Arbitrary single qubit gate decomposition
   This step takes a single qubit gate as the input and produces an approximating sequence of gates in the universal library set as the output. We see this procedure is the same as a looking up procedure to find out the " nearest " sequence for a given gate, according to a certain distance function (an example is mentioned in the section 2) from a search space of possible sequences. The problem is assuming we have to generate and store all possible gates combinations as possible candidates, when the length of the candidate gate sequences increase, the search space size grows exponentially. This results in not only the time consumed for the looking up procedure probably growing exponentially (if we cannot found an efficient search algorithm), but also we may not have the ability of generating and storing all possible gate sequences, and therefore we cannot even conduct any looking up procedure.
   Figure 5.1 below shows how fast the number of possible gate sequence cases grows with the length of the sequence, even with duplicated instances removed: The number of cases double when the gate sequence length increases one. The library used in this case is the set of $\{H, T, T^{-1}\}$ gates. See Appendix 1 for more details. In our implementation environment, the maximum length of a gate sequence which can be generated and stored the whole instances set is 18, as shown in the figure.
   In this case, the storage required to store these gate sequence cases also grows fast at the same pace. Even the minimum information about a case of a gate sequence, the volume of the file which stores the cases of gate sequence of the length 18 reaches 25 MB. Therefore, even with powerful computers having the disk volume of 1000TB, by a simple calculation, we see that storing all the cases of gate sequences

of the length more than 44 gates is impossible

Figure 5.1: Number of possible gate sequences grows exponentially with the length of that sequence.




## 5.5 Some related research

Much work has been invested into the problem of decomposition of single qubit gates. Colin P. Williams and Alexander G. Gray in [12] proposed a genetic method to decompose unitary matrices of quantum gates by considering that a solution for decomposition is a chromosome and a gene is a unitary matrix of a quantum gate in the library. They take advantage of chromosome mutation, crossover and selection to avoid generating all possible candidate solutions (as chromosome form). However, they only significantly succeeded in the case of exact decomposition, i.e. when there is a candidate sequence that has the same unitary matrix as the query gate.

Another powerful approach for this problem is attempting to create an algorithm that efficiently implements the Solovay-Kitatev theorem, which allows us to find an approximation for a target gate with a lookup-space-limited enumerated looking up subroutine. In [13], M. Dawson and A. Nielsen discussed about an algorithm to approximate an arbitrary single-qubit quantum gate using only H and T gates. The algorithm can be found in the next chapter of this thesis. However, they had to assume that a adequate

preprocessing stage has been completed which allows us to find a basic approximation to an arbitrary unitary matrix for the bottom procedure of their recursion. This basic stage encounters the difficulties we have indicated: **Exponential growth** of the search space. However, simple analysis of their algorithm and the Solovay-Kitaev theorem indicated that the efficiency of the whole algorithm depends strongly on the basic approximation stage, so we are encouraged can achieve something better than the " sufficient " one. In this research, we actually inherit the algorithm proposed by M. Dawson and A. Nielsen, and enhance its performance by applying new techniques for the basic stage of matrix lookup.

## 5.6 Chapter summary

This chapter discussed the position of quantum gate decomposition problems in this research by considering the research outcome evaluation criteria pointed out in the first chapter as well as the difficulties of this problem. We also indicated the formula to calculate the precision of an approximated quantum gate decomposition. In the next chapter, solutions for the problems defined in this chapter will be the main topic.

# Chapter 6

# Proposed Solutions

In this chapter, we start from giving an insight about the Solovay-Kitaev theorem, which brings a potential approach for the problem of gate decomposition, and its implementing algorithm. We then introduce our techniques to improve the performance of the implementing algorithm. The results of our method will be discussed in Chapter 8.

## 6.1 Inherited techniques analysis

### 6.1.1 The Solovay-Kitaev (SK) theorem

The basic goal of the SK algorithm is to take an arbitrary quantum gate $U$ and find a good approximation to it by decomposing it into a sequence of gates $g_1, \ldots, g_m$ drawn from some finite set $\mathcal{G}$ which in the context of compiler, can be called the gate library set. The SK theorem may be stated as follows:

**The Solovay-Kitaev theorem.** *Let $\mathcal{G}$ be an instruction set for $SU(d)$, and let a desired accuracy $\epsilon > 0$ be given. There is a constant $c$ such that for any $U \in SU(d)$ there exists a finite sequence $S$ of gates from $\mathcal{G}$ of length $O(\log^c(1/\epsilon))$ and such that distance$(U, S) < \epsilon$.*

### 6.1.2 Current implementing algorithm (by Dawson-Nielsen)

We present the main ideas used in the Solovay-Kitaev algorithm by Dawson and Nielsen by a few lines of pseudo code with a detailed explanation following.

```
function Solovay-Kitaev(Gate U, depth n)
if (n == 0)
1:  Return Basic Approximation to U
else
2:  Set U_{n-1} = Solovay-Kitaev(U ,n − 1)
```

3: Set $V$, $W$ = GC-Decompose($UU_{n-1}^{\dagger}$)
4: Set $V_{n-1}$ = Solovay-Kitaev($V$,$n-1$)
5: Set $W_{n-1}$ = Solovay-Kitaev($W$,$n-1$)
6: Return $U_n = V_{n-1}W_{n-1}V_{n-1}^{\dagger}W_{n-1}^{\dagger}U_{n-1}$;

A detailed explanation and proof of this algorithm requires a very strong background about both quantum computation and mathematics and can be found in [14], so in this thesis, we only summarize the fundamental points which make the algorithm work, adapting the explanation from [13]

1. Input:
   An arbitrary single-qubit quantum gate specified by a unitary matrix $U$, which we desire to approximate, in SU() form, i.e $U * U^{\dagger} = I$ and $det(U) = 1$, and a non-negative integer $n$, which implies the accuracy level of the decomposition for $U$.
2. Work:
   The Solovay-Kitaev function is recursive, so that to obtain an $\epsilon_n$-approximation to $U$, it will call itself to obtain $\epsilon_{n-1}$-approximations to certain unitaries. The recursion terminates at $n = 0$, beyond which no further recursive calls are made:
   if ($n == 0$)
   Return Basic Approximation to $U$
   In order to implement this step we assume that a preprocessing stage has been completed which allows us to find a basic $\epsilon_0$-approximation to arbitrary $U \in SU(2)$.
3. Output:
   The function returns a sequence of instructions which approximates $U$ to an accuracy $\epsilon_n$, where $\epsilon_n$ is a decreasing function of $n$, so that as $n$ gets larger, the accuracy gets better, with $\epsilon_n \to 0$ as $n \to \infty$, assuming that the preprocessing stage on line 1 of the pseudo code can approximate U to a sufficient accuracy $c_{\text{approx}}$. Let $l_n$ be the length of the sequence of instructions returned by Solovay-Kitaev(U, n), and let $t_n$ be the corresponding runtime of the search algorithm. We describe these factors in detail below

$$\epsilon_n = \frac{1}{c_{\text{approx}}^2}\left(\epsilon_0 c_{\text{approx}}^2\right)^{\left(\frac{3}{2}\right)^n}. \tag{6.1}$$

$$l_n = O(5^n) \tag{6.2}$$

$$t_n = O(3^n). \tag{6.3}$$

   and therefore we have the following equations about the relation between $l_\epsilon$, $t_\epsilon$, $n$ and $\epsilon$

$$n = \left\lceil \frac{\ln\left[\frac{\ln(1/\epsilon c_{\text{approx}}^2)}{\ln(1/\epsilon_0 c_{\text{approx}}^2)}\right]}{\ln(3/2)} \right\rceil . \tag{6.4}$$

$$l_\epsilon = O\left(\ln^{\ln 5/\ln(3/2)}(1/\epsilon)\right) \tag{6.5}$$

$$t_\epsilon = O\left(\ln^{\ln 3/\ln(3/2)}(1/\epsilon)\right). \tag{6.6}$$

It is shown that $l_\epsilon$ and $t_\epsilon$ are both polylogarithmic in $(1/\epsilon)$, giving a pretty proof for the Solovay-Kitaev theorem.

### 6.1.3 Promising improvement indication

1. From the algorithm pseudo code, we see that the `Solovay-Kitaev`(Gate U, depth n) program has to call the basic approximation subroutine $3^n$ times, whereas for an acceptable accuracy $\epsilon$ very small, $n$ can not be too small (usually around 4 or 5), according to equation 6.4. As a result, the running time of this basic subroutine affects the running time of the whole function a great deal, especially if the function is itself a part of the compilation process of a quantum circuit with many gates need to decompose. The solution for this point is described in section 6.3
2. More important, having a look at equations 6.1 and 6.2, we come up with the following conclusion: If we can reach a higher precision than the sufficient one for the basic matrix looking up stage (by extending the looking up space over the limitation of the space of generatable matrices set by a **separate method**), the function $\epsilon_n$ of $n$ will decrease faster. This means, for instance, instead of running the function `Solovay-Kitaev (Gate U, 4)` we can run the function `Solovay-Kitaev(Gate U, 3)` with improved basic approximation to get the same precision outcome. As a result, according to equation 6.2, the outcome sequence should have significantly shorter length, so the outcome of the compilation process for a quantum gate requires fewer resources. However, to give the basic approximation step more accuracy, we have to widen the gate (or equivalently matrix) lookup space, i.e making it possible to check longer gate sequences as candidate gates, and we encounter the problem of limitation of resource to generate and store *all* of those candidate gates as described in Chapter 5. The solution for this approach is described in section 6.2

## 6.2 Solutions for search space expansion (SSE) problem

### 6.2.1 Proposed algorithm enhancing techniques

The above solution is feasible only if we have generated all the instances of gate sequences and store them in a database as 3-D points (or equivalently a 3-D vector). However, as mentioned in section 6.1, if we want to have a discovery looking up procedure on the set of gate sequences when we do not have enough time and space resource to generate and store all of them, we obviously need another method.
In this research, we propose the following solution to extend the range of matrix point search without explicitly generating and storing all of the candidate gate sequences having relatively large length (up to 56 gates/sequence).

1. Sample a sufficient number of instance of gate sequence, then store them as a set of 3-D points. For example, in the case of gate sequence length=28, we randomly sample a set of 100000 3-D points. Call the sequence length $l_1$
2. For each looking up procedure, we conduct the search on the sample point set (using advanced technique like the technique of GNAT described in the previous section), and collect the points in sample set which have the distance to the query point smaller than a certain $\epsilon_1$ value into a result set.
3. For each point in the result set, we generate the vicinity points. These are *hopefully* have a small distance to the query point by the following method:
   (a) Split (partition) the gate sequence of that point into a certain number of subsequence. For example, with a point representing a gate sequence of length 30, we can split the sequence into 3 parts of length 10 subsequences, for a gate sequence of length 28, we can split the sequence into 2 parts of length 14 subsequences. Call the length of each sub sequence $l_2$
   (b) For each subsequence called $p_1$ in the 3-D space, considering it as a gate with a respective unitary matrix, calculate its coordinates in 3-D space. Search it neighbour points (distance to $p_1$ limited by a certain value $\epsilon_2$ ) on the *stored* point set for gate sequences of length $l_2$.
   (c) Merge the results gate sequences and we have a potential candidate sequences of length $l_1$
4. Linear search for nearest points of the query from the set of generated candidate point.

**Pseudo-code**

```
function Ret=SSE-Search(Gate U, precision ϵ, precision ϵ1,
precision ϵ2, PointSet samples)

  PointSet firstResults = GNAT-Search(U,ϵ1,samples);
  foreach Point point in firstResults
     PointSet partitions = Split(point);
     PointSetSet VicinityOfPartitions = GNAT-Search-For-Set(partitions,ϵ2);
     PointSet cloud = Merge(VicinityOfPartitions);
     foreach Point pt in cloud
        if Distance(pt,U) < ϵ then
           Add-To-Result(p,Ret);
        endif
     endforeach
  endforeach

endfunction
```

**Analysis**

Our solution is based on the supposition that when approximating the gate sequence merged by 2 parts $V_1V_2$, the concatenating sequences of $U_1U_2$ are potential candidates, where $U1$ is a closed approximation for $V_1$ and $U_2$ is a closed approximation for $V_2$. We present the mathematical proof for this supposition as following:
Given

$$\begin{aligned} D(U_1, V_1) &\leq \epsilon_1 \\ D(U_2, V_2) &\leq \epsilon_2 \end{aligned}$$

We have

$$\begin{aligned} D(U_1U_2, V_1V_2) &\leq D(U_1U_2, V_1U_2) + D(V_1U_2, V_1V_2) \\ \text{The right side} &= D(U_1, V_1) + D(U_2, V_2) \\ &\leq \epsilon_1 + \epsilon_2 \end{aligned}$$

Since $\epsilon_1$ and $\epsilon_2$ are relatively small ($U1$ is a closed approximation for $V_1$ and $U_2$ is a close approximation for $V_2$), it is possible to say that a sequence $U_1U_2$ is a relatively close approximation for $V_1V_2$. With a large number of sequences $U_1$ and $U_2$ and their combinations $U_1U_2$, the probability to have a very close approximation for $V_1V_2$ is relatively high.

### 6.2.2 Further development

One of potential point in our proposed technique is, similar to the ordinary SK algorithm, the idea of making a " cloud " of vicinity points around a candidate point. This, using our technique, can be applied more than once, hierarchically, allowing the basic decomposition procedure to reach much higher precision. However, applying this technique we have to suffer that the running time gets bigger as the number of linear search subroutine increases. For instance, in the following pseudo-code, the procedure `Discover-Search` subroutine (using the new technique) plays the same role as the its invoked procedure `GNAT-Search` (using the old technique). We call this technique recursive-SSE in the rest of this thesis.

#### Pseudo-code

```
function Ret=recursive-SSE-Search(Gate U, precision ϵ, precision
ϵ1, precision ϵ2, PointSet samples)

  PointSet resultSet1 = Discover-Search(U,ϵ,ϵ1, ϵ2);
  foreach Point point in resultSet1
     PointSet partitions = Split(point);
     PointSetSet VicinityOfPartitions = Discover-Search-For-Set(partitions,ϵ2);
     PointSet cloud = Merge(VicinityOfPartitions);
     foreach Point pt in cloud
        if Distance(pt,U) < ϵ then
           Add-To-Result(p,Ret);
        endif
     endforeach
  endforeach
endfunction
```

### 6.2.3 Improving the entire system performance

The algorithm by M. Dawson and A. Nielsen is relatively crude and far from optimal as it almost quintuples the length of the gate sequence answer on each level of recursion. Therefore, if we can extend the basic search range **by a different method** like the method we proposed, we can make the basic approximation much more efficient, allowing us to reduce the answer gate sequence length to reach a desired accuracy, thanks to reducing the level of recursion.

## 6.3 Solution for large-size database matrix looking up subroutine speeding up

### 6.3.1 From the matrix look up problem to a geometry point search problem

#### Matrix and Vector relation

According to the following equations, we have a 1-1 map between the set of SU(2)($2 \times 2$ special unitary matrix) and the 3-D vector space inside the sphere $((0,0,0), 2\pi)$

$$\forall U \in SU(2) \Longleftrightarrow \exists \text{ only one } \vec{v} = (x, y, z) \in R^3$$

$$U = e^{-i/2 * \vec{v}.\vec{\sigma}}$$

$$\vec{v}.\vec{\sigma} = x * X + y * Y + z * Z$$

$$\|\vec{v}\| \leq 2\Pi$$

$X, Y, Z$: Pauli Matrices
We call $\vec{v}$ the specifying vector for matrix $U$

#### The properties of a distance function

Given matrices

$$U1, U2, U3 \in SU(2)$$

Call the distance function of a unitary matrix $D(U1, U2)$. The following properties have to be satisfied as shown in the previous chapter, the same as in the case of geometric distance, and can be rewritten as function of coordinates in the 3-D space.

$$D(U1, U1) = D(U2, U2) = 0$$

$$D(U1, U2) = D(U2, U1) \geq 0$$

$$D(U1, U2) + D(U2, U3) \geq D(U1, U3)$$

Because $U1, U2$ can be mapped onto $\vec{v1}, \vec{v2} \in R^3$, the matrix search problem can be equivalently treated as geometric search in 3-D space, which has been researched widely.

### 6.3.2 Help for the entire system performance

Applying the algorithm proposed by M. Dawson and A. Nielsen, the 2x2 unitary factoring procedure algorithm has linear run time with the run time of the matrix search routine described in the first section by this formula. However the linear proportion is very high ($\simeq \log(1/\epsilon)^{2.77}$) so the run time

of the matrix search procedure has a considerable effect on the whole SK decomposition algorithm run time. That means we have to find a time-efficient algorithm for that procedure. By considering the problem as geometric search, we can take advantage of previous research results as well as original research.

### 6.3.3 Geometric Near-neighbour Access Tree data structure

Except for linear search, the main trend of search algorithms is turning geometric searches into tree-based searches by space-partitioning techniques in order to reduce the cost of searching to $O(\log n)$ in the best or average case. Some popular solutions that should be mentioned here are R-Tree and Kd-Tree [15]. These approaches divide the search space into clusters specified by coordinates in the space. This approach probably results in bad effect in this case (matrix search) because that kind of clustering only works well when the distance function is Euclidean distance, whereas the distance function in the case of matrix points is little different.

### 6.3.4 Geometric near-neighbour access tree (GNAT)

This approach comes from Brin[16].The main idea of GNAT is partitioning the search space by using clusters associated with a number of fixed points called splitting points

#### Data structure

- Splitting points. Chosen from the search space. The selection algorithm is an open topic of research, but the desire is picking the points as far from each other as possible.
- Space cluster. All the points in the search space are divided into cluster associated with splitting points. In more detail, the cluster associated with a splitting point (called *sp*) is the set of points which has *sp* as the nearest splitting points in the splitting point set. A cluster with its own splitting point provides an imagination of a nucleus with the atom at its centre and electrons around. Figure 6.1 shows how a 2-dimension space divided into clusters with atoms are splitting points.

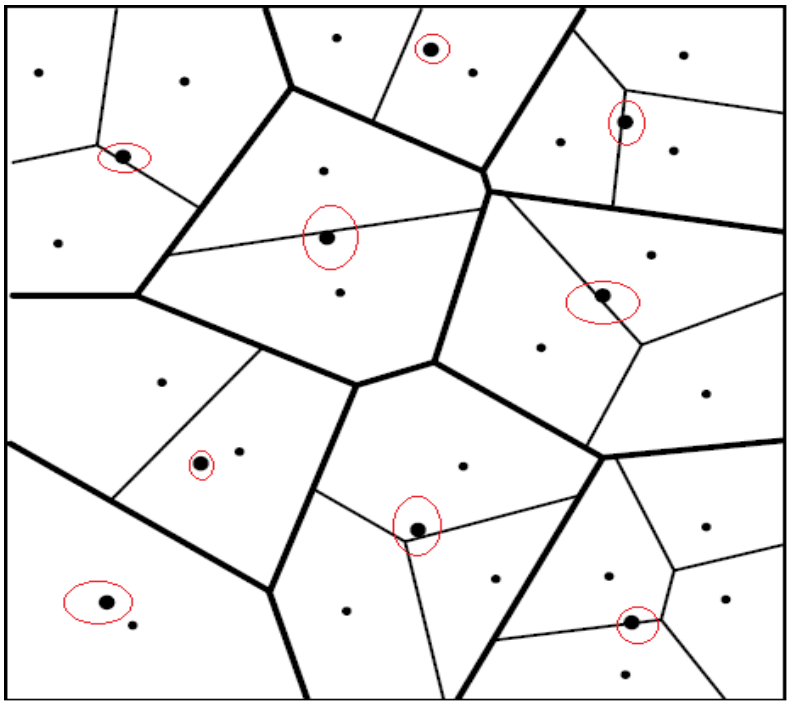


Figure 6.1: A simple GNAT with clusters. Adaptation of image from [16]. The points with surrounding red circle are the splitting points of the same level clustering.

### Search algorithm

The main idea of this algorithm based on tree pruning-and-traversal technique to reduce the candidate points to be scoured. We can model this algorithm as following:

1. Data structure as a tree
   Recall the space clusters. Each point cluster separates from same-level others (in term of clusters generated from a set of splitting points) and recursively contains a number of sub-clusters, or, at terminal level, points. This give us an perfect image of a tree structure, with nodes, sub-trees and leaves. The nearest neighbours search problem can be seen as a tree traversal problem, which we traverse from tree root to tree leaves for the answer.
2. Tree pruning and traversal
   As in the geometric nearest neighbours search problem, there is no comparison metric can be used to search on the tree like in the case of scalar value search: one path from the tree root to a tree leaf, we can only attempt to proximate this scenario by pruning the tree on each traversal step. A method of using pre-computed metadata for

tree pruning purpose is discussed below. Consider we have searching for nearest neighbours $\{p\}$ of a query point $x$ with $D(p, x) \leq \epsilon$

- Suppose we are visiting a tree node (an abstract word for a cluster $C$, with the corresponding splitting point $s$). We have the distance $D(s, x) = d$.
- Applying the triangle inequality, we can conclude that a point $p$ with $D(p, x) \leq \epsilon$ must relax the inequality $d - \epsilon \leq D(p, s) \leq d + \epsilon$. Thus $p$ can not exist in a cluster $Q$, which $Range(C, Q) \cap [d - \epsilon, d + \epsilon] = \emptyset$. As a result, we do not need to take clusters $\{Q\}$ into account afterwards.

The rest of tree traversal steps are rather identical to any tree traversal. We conduct Deep First Search on the tree, ignoring eliminated sub-tree generated from clusters $\{Q\}$, until reaching tree leaves, which present lowest level clusters. In a cluster like these, we simply calculate the distance of each point and the query point to determine it is an answer or not.

**Pseudo-code**

```
function Ret=GNAT-Search(Gate U, precision ϵ, PointSet space)

if Size-Of(space) <= Low-Threshold
  Ret = Linear-Search(U,ϵ,space);
  return;
elseif
  PointSet SplittingPoints = Get-Splitting-Points(space);
  PointSetSet clusters = Get-Cluster-Around-Splitting-Points(space);
  PointSetSet candidateClusters = clusters;
  foreach Point sp in SplittingPoints
     d = Distance(sp,U);
     foreach PointSet cluster in clusters
        r_max = Max-Distance(sp,cluster);
        r_min = Min-Distance(sp,cluster);
        if [d − ϵ, d + ϵ] ∩ [r_min, r_max] = ∅ then
           Remove-From-Candidates(candidateClusters,cluster);
        endif
     endforeach
  endforeach
  foreach PointSet candidateCluster in candidateClusters
     Ret = Ret ∪ GNAT-Search(U,ϵ,candidateCluster);
  endforeach
endif

endfunction
```

### Analysis

This algorithm's main disadvantage is the running time required for the data set processing stage to calculate the max and min distance from each splitting point to each cluster. However, in this project, for only one data set there are large numbers of search procedures, therefore this disadvantage does not matter much. The average running time required for one search procedure, the most important performance attribute, is conversely much shorter than the linear search time. This result comes from experiments and contributed to the significant reduction of candidate clusters, despite of having the same theoretical complexity.

# Chapter 7

# Implementation

In order to make the system work not only on powerful computers but also on resource-limited machines, towards tablets or even smart phones, we choose the following environment to implement the entire system.

Table 7.1: Implementation environment

| Platform and software environment | |
|---|---|
| OS | Ubuntu 10.04 virtual machine |
| Virtual Environment manager | VirtualBox 4.1 on Windows 7 Home Premium |
| Virtual memory | 2076 MB |
| Virtual disk | 30 GB |
| JDK | Java version 1.6.018 |
| JavaVM | JRE 6 |
| Octave | Octave 3.2.3 |
| Java-Octave binder | JavaOctave 0.6.1 from Kenai Project |
| Parser constructor | Java Compiler Compiler version 5.0 |
| Java Tree Builder | JTB 1.3.2 |
| Hardware environment | |
| CPU | Intel(R) Core(TM) i3 3.00GHz |
| Memory | 4 GB |
| HDD | 228 GB |

In the rest of this chapter, we present the system architecture of the whole software, which realizes the described techniques in the table 7.2. Each layer corresponds to a module in the whole system, which is invoked by the upper layer and invokes the lower layer to complete its work.

Table 7.2: Software layered architecture

| **Compilation program** |
| --- |
| **Lexical Analyser-Parser-Output generator**<br>JavaCC + JTB |
| **Logical Quantum Circuit Builder**<br>Pure Java |
| **Quantum Gate Decomposer**<br>Pure Java |
| **Subroutine of the Solovay-Kitaev algorithm**<br>Octave + Java |
| **Platform**<br>JVM + Octave Engine |
| **Operating System** |

# Chapter 8

# Experiments and Evaluation

## 8.1 Orientation

In this chapter, we discuss the evaluation process for the system. As this system is above all, a software, it has to be evaluated as a concrete entity. This evaluation is conducted based on the qualitative criteria indicated in chapter 1. However, as the system construction involves many research topics, and leads to solutions to overcome difficulties, it is also necessary to evaluated these solutions separately. Because the proposed solutions are all designed towards quantitative performance, their evaluation has to be done based on the quantitative criterion, also indicated in chapter 1.

## 8.2 System overall evaluation

The system currently works fair, with the output format also written in QAS, and only using library gates. The time consumed for compilation process of an input program containing up to 25 quantum gate is no longer than 1 hour, almost of time consumed is for the gate decomposition task.

## 8.3 Research points evaluation

### 8.3.1 Evaluation of search space expansing method

1. Comparison objective:
   M. Dawson and A. Nielsen's algorithm with traditional basic approximation step, i.e. conduct the matrix looking up procedure on entirely generated, stored instances of gate sequences of length no larger than $l_0 = 18$
2. Comparison value:
   Gate sequence length required to achieve a desired accuracy (distance between query matrix and returned matrix of the algorithm) $\epsilon_0$. Often

$\epsilon_0 \approx 10^{-4}$.

3. Evaluation method:
   Generate a sufficient number of special unitary random matrices. Conduct the M. Dawson and A. Nielsen algorithm with the traditional basic approximation step (called old technique) and with the proposed search range widening methods described in section 6.2 used for the basic approximation step (called SSE and recursive SSE technique respectively). Collect the accuracy and length of gate sequences returned for each technique. Statistically analyse the collected data. We also plot the results to provide a visual assessment for each technique.
4. Experiment and Evaluation results:
   We randomly generate a number of special unitary matrices, say 25, corresponding to 25 quantum gates, running the M. Dawson and A. Nielsen algorithm for the Solovay-Kitaev decomposition (6.3.4) with $n = 4$ with the old technique, $n = 3$ with the SSE technique and $n = 2$ with the developed new method, to have each obtain an acceptable accuracy. With the old method, the maximum length of the gate sequence outcome of basic gate approximation (decomposition) is 18, whereas the respective numbers of SSE technique and recursive SSE technique are 28 and 56.

   The collected data is in the table 8.1 below, with each row corresponding to a query gate, the length of the gate sequence returned by the algorithm with the old technique, the accuracy to the query gate of gate sequence returned by the algorithm with the old naive technique, the length of gates sequence returned by the algorithm with the SSEs techniques, the accuracy to the query gate of gates sequence returned by the algorithm with the SSEs techniques.

   In practice, as the returned gate sequence length is almost fixed by the algorithm itself (in case of the old technique, the length of the returned sequence is about 9000, whereas in case of the SSE technique, this number is about 3000, and in case of the recursive SSE technique, this number is about 1300) so we can consider that the SSE technique has significantly compacted the length of the resultant gate sequence by a factor around 3 and the recursive SSE technique has significantly compacted the length of the resultant gate sequence by a factor around 7, compared to the old technique.

   Now we deal with the accuracy outcome of the 3 techniques. **Assume** that the difference of the accuracy of the 3 techniques is approximately normal distributed so that we can use the Student paired test to calculate the confidence interval for the mean of it. According to the formula in [17] and the table of critical value for Student's distribution at http://www.scribd.com/doc/28555364/Student-s-t-Test-Table, we have the 95% confidence interval of the mean of difference of the accuracy of the SSE technique compared to the old technique is

Table 8.1: Outcome gates sequence length and accuracy by old and new method for randomly chosen queries

| Query | Old technique | | SSE technique | | Recursive SSE technique | |
|---|---|---|---|---|---|---|
| # | Result length | accuracy | Result length | accuracy | Result length | Accuracy |
| 1 | 9472 | 9.00E-05 | 3154 | 7.16E-05 | 1256 | 0.00020304 |
| 2 | 9253 | 0.00057555 | 3190 | 0.00024060 | 1264 | 0.00038174 |
| 3 | 8857 | 0.00034084 | 3146 | 9.20E-05 | 1282 | 0.00029360 |
| 4 | 8885 | 0.00014167 | 3250 | 4.54E-05 | 1180 | 8.07148E-05 |
| 5 | 9079 | 0.00025919 | 3092 | 4.17E-05 | 1280 | 0.00024141 |
| 6 | 9304 | 0.00018723 | 3210 | 5.16E-05 | 1292 | 0.00018779 |
| 7 | 8984 | 0.00050555 | 3150 | 0.00022057 | 1324 | 0.00016339 |
| 8 | 9029 | 0.00075836 | 3114 | 0.00015314 | 1294 | 0.00067551 |
| 9 | 9066 | 0.00021418 | 3216 | 0.00023754 | 1284 | 0.00044422 |
| 10 | 9326 | 0.00044602 | 3140 | 0.0002364 | 1268 | 0.00017198 |
| 11 | 9139 | 0.00079344 | 3170 | 0.00016378 | 1288 | 0.00021800 |
| 12 | 8566 | 0.00031750 | 3234 | 0.00034421 | 1312 | 0.00026985 |
| 13 | 9432 | 0.00051561 | 3294 | 0.00073163 | 1274 | 0.00019819 |
| 14 | 8827 | 0.00039050 | 3190 | 0.00016518 | 1300 | 4.03721E-05 |
| 15 | 8447 | 0.00012512 | 3116 | 0.00018926 | 1320 | 0.00011416 |
| 16 | 9127 | 0.00028644 | 3286 | 0.00011417 | 1298 | 0.00040065 |
| 17 | 9344 | 0.00014221 | 3160 | 7.54E-05 | 1284 | 0.00022579 |
| 18 | 9293 | 0.00026654 | 3178 | 9.91E-05 | 1294 | 7.83317E-05 |
| 19 | 9021 | 0.00102097 | 3174 | 0.00051386 | 1182 | 0.00033873 |
| 20 | 9484 | 0.00042235 | 3234 | 0.00030455 | 1288 | 0.00022479 |
| 21 | 9142 | 0.00019897 | 3256 | 0.00039183 | 1296 | 0.00027780 |
| 22 | 9072 | 0.00027672 | 3194 | 0.00029566 | 1178 | 0.00023652 |
| 23 | 8760 | 0.00013384 | 3244 | 0.00013375 | 1250 | 0.00023858 |
| 24 | 9087 | 0.00036537 | 3106 | 0.00029341 | 1260 | 0.00029718 |
| 25 | 9058 | 0.00058659 | 3058 | 9.73E-05 | 1322 | 0.00014402 |
| Mean | 9082.16 | 0.00037443 | 3182.24 | 0.00021214 | 1274.8 | 0.00024585 |
| STDEV | 261.488317 | 0.0002343 | 60.8038376 | 0.00016081 | 40.64480287 | 0.00013358 |

about $\left(7.40 \times 10^{-5}, 2.50 \times 10^{-4}\right)$. This result gives strong statistical evidence that we can expect a higher accuracy by using the SSE technique, compared to the old technique. Similarly, applying the same test to the recursive SSE technique and the old technique outcome, we have the 95% confident interval of the mean of difference of the accuracy of the SSE technique compared to the old technique is about $\left(3.55 \times 10^{-5}, 2.22 \times 10^{-4}\right)$. This result also gives strong statistical evidence that we can expect a higher accuracy by using the recursive SSE technique, compared to the old technique.

We also conducted experiments of each technique for most-used quantum gates decompositions, such as gates int the well-known quantum Fourier transform subroutine. For example, when decomposing the gates in the quantum Fourier transform having the corresponding unitary matrix of $R(k) = \begin{pmatrix} 1 & 0 \\ 0 & e^{2i\pi/2^k} \end{pmatrix}$ with $k = 2$ or $k = 3$, the old technique even may not converge to an enough accuracy, whereas the new methods proposed in this thesis reach high accuracy. The table 8.2 shows these differences. However, this still needs further work for verification.

Table 8.2: Gates in Fourier transform circuit decomposition

| | $R(2)$ | $R(3)$ |
|---|---|---|
| Old technique outcome accuracy | 0.001243 | 0.00182 |
| Old technique outcome sequence length | 8858 | 9012 |
| SSE technique outcome accuracy | $2.175 \times 10^{-13}$ | $5.306 \times 10^{-13}$ |
| SSE technique outcome sequence length | 1302 | 1266 |
| recursive SSE technique outcome accuracy | $1.4161 \times 10^{-4}$ | $4.8444 \times 10^{-4}$ |
| Recursive SSE technique outcome sequence length | 3146 | 2582 |

We also conducted an experiment plotting the relation between the Solovay-Kitaev decomposition outcome gate sequence length and approximation accuracy with different value of parameter $n$ in the function 6.3.4 for the new techniques and the old technique (figure 8.2) on another set of test cases, in order to visualize the efficiency of each technique. The plot also visually supports the conclusion that our proposed techniques is significantly better than the old one.

### 8.3.2 Evaluation of Geometric Near-neighbour Access Tree in matrix point looking up

(a) Comparison objective:
Linear look up procedure for matrices having distance to the

query matrix no larger than a certain value $\epsilon$

(b) Comparison value:
The time to look up an answer matrix in the same database is compared

(c) Evaluation method:
Generate a sufficient number of random matrices. Conduct the looking up procedure by the linear method and by the Geometric Near-neighbour Access Tree method, collecting the average time required for each technique. Statistically analyse the collected data.

(d) Experiment and Evaluation results:
The collected data is presented in the chart below. The chart compares the average time required to complete nearest neighbour search for 120 query points in different sizes of search space.

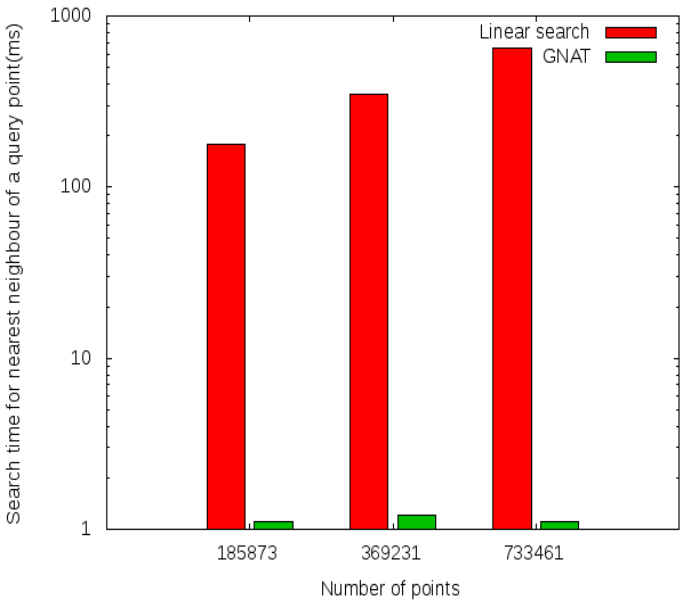
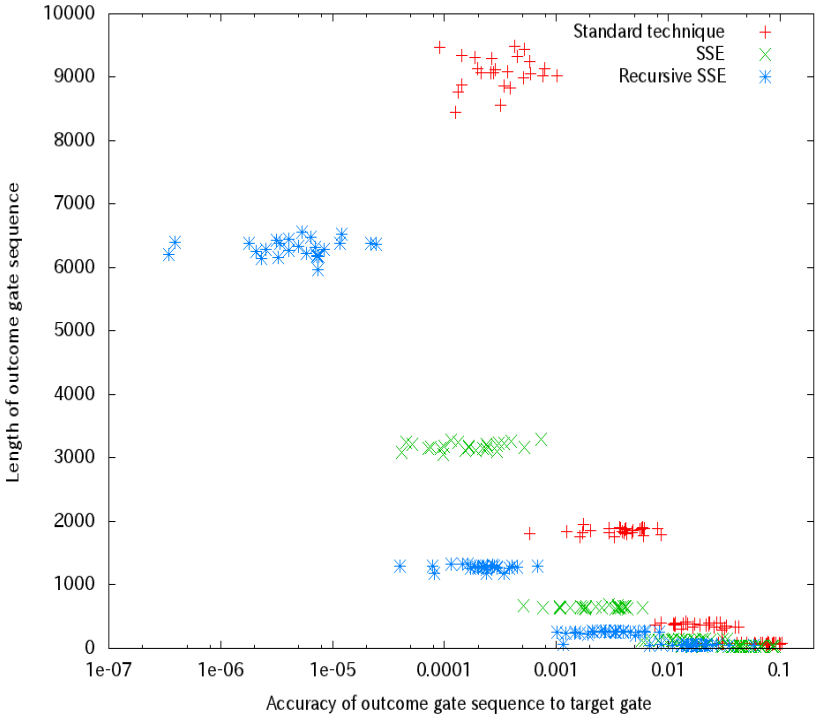



Figure 8.1: Time required: GNAT vs Linear search

From the chart, the large difference between the 2 techniques can be clearly recognized.

Figure 8.2: Approximation accuracy vs length of outcome gates sequence



# Chapter 9

# Conclusion and future work

In this work, we have successfully modelled a general quantum circuit model in a programming language, and constructed a compiler which has the ability to translate this language into several output formats. We also proposed an improvement to M. Dawson and A. Nielsen's algorithm for the Solovay-Kitaev theorem by reducing the number of gates used to approximate an arbitrary single qubit gate to acceptable accuracy by at least a factor of about 3 or even 7 by extending the search space for the basic approximation stage of this recursive algorithm to get higher accuracy at this step, which remarkably increases the accuracy of the whole algorithm. This improvement is especially seen in the decomposition of gates in the quantum Fourier transform circuit, making it more significant. The run time required for the approximation procedure was also taken into account. This work found a very efficient method to reduce the procedure by improving the basic step of the matrix looking up subroutine by solving it as a geometric search problem and taking advantage of the previous work on that field like [16].

In the future, besides giving more features to the language in order to meeting user requirements (which are getting larger), we will concentrate on composing a more efficient algorithm for the gate decomposition problem to come up with a lower bound for the Solovay-Kitaev theorem, toward multi-qubit gate efficient decomposition. The phase effect removal on gate decomposition is also one of the target. It is a big remaining problem. We will also work on analysing the current results to direct future work.

# Acknowledgement

First and foremost, I would like to express my gratitude to my supervisors: Professor Hideyuki Tokuda, Professor Jun Murai, Associate Professor Hiroyuki Kusumoto, Professor Osamu Nakamura, Associate Professor Kazunori Takashio, Associate Professor Rodney D. Van Meter III, Associate Professor Keisuke Uehara, Associate Professor Jin Mitsugi, Lecturer Jin Nakazawa.
Especially, I would like to express my deepest gratitude to Associate Professor Rodney D. Van Meter III and Doctor Clare Horsman, who gave me fundamental knowledge in the field of Quantum Computation and Quantum Information, as well as enthusiastically supported me in this project from the very beginning to the end. Without their help, I could not finish this thesis.
I thanks all my fellow in AQUA lab: master student Shota Nagayama, bachelor students Kaori Ishizaki, Yasuharu Miyata, Shoichiro Fukuyama, Iori Mizutani, Masakazu Fujimori, Koji Murata, and Tomoki Sugiyama for their help during my semesters in AQUA. In particular, I would like to thank AQUA members who established and maintained server machines. Their work really made my project much easier.

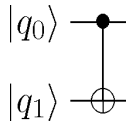
# Appendix A

# Frequently used quantum gates

In this appendix, we present frequently used quantum gates, which are mentioned in the thesis as useful building blocks in the design of quantum circuits. These gates are presented with their name, denoting symbol and the characteristic unitary matrix in the table A.1. For multi-qubit gates such as CNOT and Toffoli gates which may make readers confused, a comprehensive explanation is as follows:

- Controlled-NOT (CNOT) gate
  Diagram explanation: If and only if the control qubit $|q0\rangle$ is set to $|1\rangle$, the target qubit $|q1\rangle$ is flipped (from $|0\rangle$ to $|1\rangle$ and vice-versa)

Figure A.1: CNOT gate




- Toffoli gate
  Diagram explanation: If and only if the 2 control qubits $|q0\rangle$ and $|q1\rangle$ are set to $|1\rangle$, the target qubit $|q1\rangle$ is flipped (from $|0\rangle$ to $|1\rangle$ and vice-versa)

Figure A.2: Toffoli gate

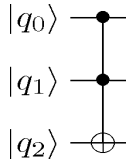


Table A.1: Frequently used quantum gate

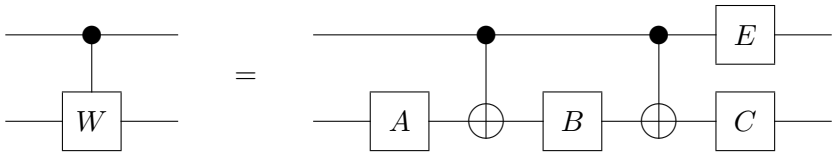
| Name | Denoting symbol | Characteristic unitary matrix |
|---|---|---|
| Single qubit gate | | |
| Identity gate | I | $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ |
| Pauli-X gate | X | $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ |
| Pauli-Y gate | Y | $\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$ |
| Pauli-Z gate | Z | $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ |
| Hadamard gate | H | $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ |
| $\frac{\pi}{4}$ gate | T | $\begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$ |
| $\alpha$ phase shift gate | $S_\alpha$ | $\begin{pmatrix} e^{i\alpha} & 0 \\ 0 & e^{i\alpha} \end{pmatrix}$ |
| 2 qubits gate | | |
| Controlled-NOT | CNOT | $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$ |
| 3 qubit gate | | |
| Controlled-Controlled-NOT | Toffoli | $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$ |

# Appendix B

# Barenco decomposition method

In this appendix, we summarize the decomposition technique applied for one and two controlled quantum gate, presented in [11]. This decomposition technique provides a way to decompose a controlled quantum gate into the sequence of elementary gates drawn from the library of Controlled-Not gate, and arbitrary single qubit gates upto global phase.

1. One-Controlled gate
   For a unitary $2 \times 2$ matrix $W$, a $\wedge_1(W)$ (one-controlled gate of $W$) gate can be simulated by a network of the following form.




   where $A$, $B$, and $C \in SU(2)$; $E$ is unitary.

   The formulae to calculate unitary matrices $A$, $B$, $C$ and $E$, assuming that $W = \mathrm{R_z}(\alpha) \cdot \mathrm{R_y}(\theta) \cdot \mathrm{R_z}(\beta) \cdot \mathrm{Phase}(\delta)$ are:

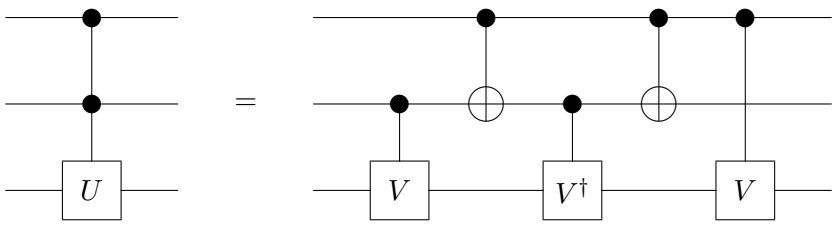
$$A = \mathrm{R_z}(\alpha) \cdot \mathrm{R_y}(\frac{\theta}{2}) \tag{B.1}$$

$$B = \mathrm{R_y}(-\frac{\theta}{2}) \cdot \mathrm{R_z}(-\frac{\alpha+\beta}{2}) \tag{B.2}$$

$$C = \mathrm{R_z}(\frac{\beta-\alpha}{2}) \tag{B.3}$$

$$E = \mathrm{R_z}(-\delta) \cdot \mathrm{Phase}(\tfrac{\delta}{2}) \tag{B.4}$$

2. Two-Controlled gate
   For any unitary $2 \times 2$ matrix $U$, a $\wedge_2(U)$ (two-controlled gate of $U$) gate can be simulated by a network of the following form




   Where $V$ is unitary and $V^2 = U$. As the result for two-controlled gates only includes CNOT gates and one-controlled gates, we can continue the decomposing process by applying the one-controlled gate decomposition (summarized above) for them to obtain the final results consisting of single qubit gates, CNOT gates and phase gates.

# Bibliography


[1] Peter Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing Issue 5*, 26, October 1997.

[2] Piotr Gawron. `http://www.quantiki.org/wiki/Bloch_sphere`, 2005. Online; accessed 19-January-2012.

[3] W. K. Wooters and W. H. Zurek. A single quantum can not be. *Nature*, 299:802, 1982.

[4] Michael A.Nielsen and Isaac L.Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2005.

[5] David P. DiVincenzo. The physical implementation of quantum computation. *Fortschritte der Physik*, 48:771–784, 2000.

[6] Tatsuya Hagino. `http://www.tom.sfc.keio.ac.jp/~hagino/web10/`, 2010. Web Text Processing lectures. Accessed 19 January 2012.

[7] Rich Maclin. `http://www.d.umn.edu/~rmaclin/cs5641/Notes/`, 2009. CS5641 Compiler Design at University of Minnesota Duluth. Accessed 19 January 2012.

[8] D. Orleans and K. Lieberherr. Dj: Dynamic adaptive programming in java. *Metalevel Architectures and Separation of Crosscutting Concerns*, pages 73–80, 2001.

[9] Rodney D. Van Meter III. `http://web.sfc.keio.ac.jp/~rdv/quantum/aqua-tools/AAAREADME.txt`, 2006. Aqua Tool online README. Accessed 19 January 2012.

[10] Alexander Slepoy. Quantum gate decomposition algorithms. *SANDIA REPORT*, page 7, 2006.

[11] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457, 1995.

[12] C. Williams and A. Gray. Automated design of quantum circuits. *Quantum Computing and Quantum Communications*, pages 113–125, 1999.

[13] C.M. Dawson and M.A. Nielsen. The Solovay-Kitaev algorithm. *Arxiv preprint quant-ph/0505030*, 2005.

[14] Aram Harrow, 2001. Bachelor Thesis of Science in Physics, Massachusetts Institute of Technology.

[15] T.K. Sellis, N. Roussopoulos, and C. Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of the 13th*

*International Conference on Very Large Data Bases*, pages 507–518. Morgan Kaufmann Publishers Inc., 1987.

[16] S. Brin. Near neighbor search in large metric spaces. *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 574–584, 1995.

[17] `http://mlsc.lboro.ac.uk/resources/statistics/Pairedttest.pdf`. Student's t-Tests.