Quantum Evolutionary Algorithm for Quantum Circuit Synthesis

by

Georgiy Krylov

B.S., Nazarbayev University (2016)

Submitted to the Department of Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at the

NAZARBAYEV UNIVERSITY

June 2018

Author .....
Department of Computer Science
May 8, 2018

Certified by .....
Martin Lukac
Associate Professor
Thesis Supervisor

Accepted by .....
Vassilios D. Tourassis
Dean, School of Science and Technology

Quantum Evolutionary Algorithm for Quantum Circuit Synthesis

by

Georgiy Krylov

Submitted to the Department of Computer Science
on May 8, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

Abstract

Quantum computing area has a lot research attention due to opportunities that possessing such device could provide. For example, quantum computers could deliver new insights to previously unsolvable problems. The reason for that is higher parallel capabilities of such devices. In addition, since quantum computers are naturally reversible, no heat dissipation occurs during computation [21]. This property could serve as a viable solution to the problem that computer chip production industry faces. Moreover, since the chip manufacturing industry reaches nanometer scale of size of elements, the effects that could cause unexpected information behavior in classical paradigm are part of the technology of quantum devices [31, 14].

Considering possible benefits that could be achieved by quantum computing devices, the new areas of Quantum Information Theory, Quantum Cryptography, Quantum Algorithms and Logic Design and many others emerged at the end of the twentieth century [31]. These areas are concentrating their efforts on solving problems of designing communication protocols, ensuring the security of the new systems, constructing appropriate algorithms. Computers that could be advancing in finding solutions in problems listed above require quantum circuits that have optimal structure and could implement error correction. This is the main motivation for this thesis work to explore the problem of circuit design. The approach that we investigate is circuit construction by the means of Quantum Evolutionary Algorithms. We propose a version of an algorithm that accounts with specificity and constraints of quantum paradigm. We use its Graphic Processing Unit (GPU) accelerated classical implementation to evaluate the behavior and performance of the proposed algorithm. Later we discuss additional complexity introduced by accounting with these constraints. We support our ideas with results of synthesis of small circuits and compare the performance with classical genetic algorithm on similar task.

Thesis Supervisor: Martin Lukac
Title: Associate Professor

Acknowledgments

Thanks to Professor Martin Lukac, Professor Benjamin Tyler and Professor Bernd Steinbach for help with the work. Special thanks to Dr Martin Lukac, Dr Mona Rizvi and Dr Benjamin Tyler for being great mentors. I would also like to say thank you to my family members for all their support.

Contents

List of Figures

List of Tables

List of Abbreviations

$CNOT$	**C**ontrolled **NOT** gate
$CCNOT, C^2NOT$	Three qubit **C**ontrolled **NOT** gate
CH/CZ	Model (gate set) for circuit design, named after gates it uses: Controlled Hadamard gate and Controlled Z gate
$CNOT/CV/CV^\dagger$	Model (gate set) for circuit design, named after gates it uses: CNOT, controlled V (square root of NOT gate) and its conjugate transpose $V^\dagger$
GA	**G**enetic **A**lgorithm
GPU	**G**eneral **P**urpose computing for **G**raphic **P**rocessing **U**nits
GPGPU	**G**raphics **P**rocessing **U**nit
GPUGA	**G**raphics **P**rocessing **U**nit accelerated **G**enetic **A**lgorithm
NMR	**N**uclear **M**agnetic **R**esonance
SU(3)	**S**pecial **U**nitary group of degree **3**
QEA	**Q**uantum **E**volutionary **A**lgorithm
QEQEA	**Q**uantum **E**ncoded **Q**uantum **E**volutionary **A**lgorithm

Chapter 1

Problem Description and Motivation

1.1 Introduction

The rapidly growing amount of information and the variety of tasks creating the need to process it demands an increase in available computing power. The traditional approach achieving increase in computational power of a computer is to increase the number of transistors laid out on the integrated circuit. This approach was prophesied by Gordon Moore in 1965 [28], and his ideas, informally were called Moore's law. In its essence, the Moore's law states that the number of elements on a chip doubles once per 18 months, and its components get cheaper proportionally within the same period of time [28]. Considering the trends of making devices more mobile or at least making them the same size, the increase of chip size is not desirable. Thus, the preferred way of achieving an increase in computing power is to shrink transistors in size. However, this approach has several drawbacks arising from fundamental principles of physics and computation theory. The first issue encountered is the heat generation linked with performing classical computation. This problem can be dealt with by implementing reversible computing - a model of computing which does not generate heat during the process of computation due to specific logic design. Another problem is the hard limit on the size of a transistor, depending on the chemical elements used to produce it [14]. If the size of integrated circuit element reaches the electron wavelength scale, the effect of electron energy quantization will appear, resulting in

information distortion [14]. Overcoming these obstacles requires persistent innovations to stay cost effective. One of the viable approaches emerged during the past decade is to create task-specific devices instead of general purpose computers. For example, using Graphic Processing Units for solving problems of machine learning and artificial intelligence became a widely applied practice. Because of the relative success of this approach, there is growing interest in considering different paradigms and insights on solving various computational tasks. Very promising computational paradigm to consider is quantum computing, because it could serve as a solution to all the problems described above. This is possible because quantum circuits are reversible in their nature, possessing the property of not generating heat similar to any other reversible computer. Quantum effects are part of the technology and thus do not lead to information distortion in same way they do for classical computers. Additionally, quantum computers outperform classical ones in tasks that benefit from higher parallel execution capabilities. Moreover, the area of quantum cryptography introduces new perspectives on dealing with security problems.

1.2 Reversible Computers as a Solution of Problem of Heat Generation

The classical hardware architecture relies mostly on use of binary devices. Such devices are prone to heat dissipation which on a large scale of integration becomes a significant problem. The problem arises from the design of these devices that are constructed to be in either ZERO or ONE state and support operations of changing the state. The essence of the problem is an inability to determine which operation should be applied based on the original state of the device. Thus, the change of state procedure is always performed regardless of the original state and damping the excess energy from the device [15]. For example, if the state ONE of the device is desired, there is no possibility to choose whether the change from ZERO to ONE should happen or the device was already at the ONE state and no operation should be

performed. In other words, for each bit erased or otherwise thrown away, the energy amount of $kT\ln 2$ joules must be dissipated, where k is the Boltzmann constant, and T is an absolute temperature of the system.

However, if the computer could be designed to implement reversible logic computations, this problem would get solved. According to Benett [4], irreversible functions could be translated to their reversible analogues by storing all the information that otherwise would be thrown away. Moreover, general purpose reversible computers are claimed to exist and they do not require significantly more complex design than their classical analogs.

1.3 Quantum Computers

1.3.1 Quantum Computers as an Implementation of a Reversible Computer

The idea of a logically reversible computer that does not require heat dissipation as a part of technology resulted in research for possible ways to construct such a device. Clearly, such system had to be fully isolated, and there were several attempts to build such computers. There are models of reversible computers such as an adiabatic computation [18] or billiard balls computer [9]. However, these architectures were unsatisfying either in terms of speed of performing computations or the system was prone to losing energy due to friction. Another model was inspired by findings in area of quantum physics. The nature of quantum mechanics was complying with proposed constraints. The transformations in quantum mechanics are unitary, moreover, each operation could be mathematically undone by applying an inverse matrix operation on the system. This means the reversibility on each step in such systems is achievable for most operations except input and output.

The theories about the quantum world and their projection in the area of computer science led to proposal of a model of a quantum computing device. In this model, the state of the computer was evolving with time and each change of the system was

proposed to be an abstraction of computation applied to the system of input quantum bits [33]. Moreover, unitary evolution of quantum process is able to simulate Turing machine behavior [3]. Such systems would be free from dissipating heat requirement, in fact, it has to be a fully isolated from interaction with outer world. The only two allowed interactions with outer world would be a process of preparing the inputs and reading the outputs (quantum measurement). The ability of building such device depends on satisfying two conditions: constructing logically reversible functions and implementing physical components.

1.3.2 Physical Realization of Quantum Computer

There are many problems impairing physical realization of quantum computers: construction of the machinery, error correction of the logic circuits, the monetary cost of the process as a whole and many more. This includes cooling, circuitry design and base material cost.

To implement a quantum computer that could efficiently perform computation, five main requirements should be satisfied. We list them exactly as they were put in original source [8] for clarity:

- *A scalable physical system with well characterized qubits;*
- *The ability to initialize the state of the qubits to a simple fiducial state, such as $|000...\rangle$;*
- *Long relevant decoherence times, much longer than the gate operation time;*
- *An 'universal' set of quantum gates;*
- *A qubit-specific measurement capability.*

These requirements, also known as DiVincenzo Criteria, are describing the difficulty of construction of a real computer. There are several possible approaches for physical realization; for example, optical quantum computers [32] or quantum computers utilizing NMR technology, supercooled Josephson junctions [10] or Trapped

Ions [35]. The proposed technologies are believed to be computationally equivalent. However, none of proposed methods solves the problem of physical realization in its entirety. All of the proposed realizations are currently expensive and only big national agencies or corporations like Microsoft or IBM could afford to possess them. This high monetary cost implies that quantum devices should have the most efficient and optimized circuitry.

1.4 Motivation for using Quantum Computers

Research in area of quantum computers and algorithm construction resulted in quantum algorithms more suitable for problems requiring parallel computations. These algorithms executed on quantum computers can show exponentially better results with respect to time of execution of an algorithm, compared to previously known algorithms applied to solve similar problems. An example of such problem is integer factoring problem. There is no known polynomial time algorithms for performing that task on classical computers, however the quantum Shor's algorithm was theorized to reduce computational complexity of factoring problem to polynomial. This algorithm was tested and implemented on physical quantum computer for small cases [38]. The Figure 1-1 displays the extracts from comparing Shor's algorithm and state of the art (by year 2003) classical algorithm [26]. The work additionally makes prediction about the the scenario for 2018, following Moore's law principles. The prediction was supported as valid by year 2014 with benchmark of new classical algorithm for factoring task [11]. The Grover's algorithm for performing the search for an element of unordered database could serve as an additional example of possible benefits. The application of the quantum integer factoring algorithm can compromise current encryption algorithms in a scale of minutes instead of days, relative to clusters of classical computers [5]. Area of security and cryptography could also benefit from using quantum computers. The inability to copy quantum information while being an obstacle for other tasks can ensure a security specialist that the data was not lost. Moreover, an attempt of compromising the data could also be detected during use

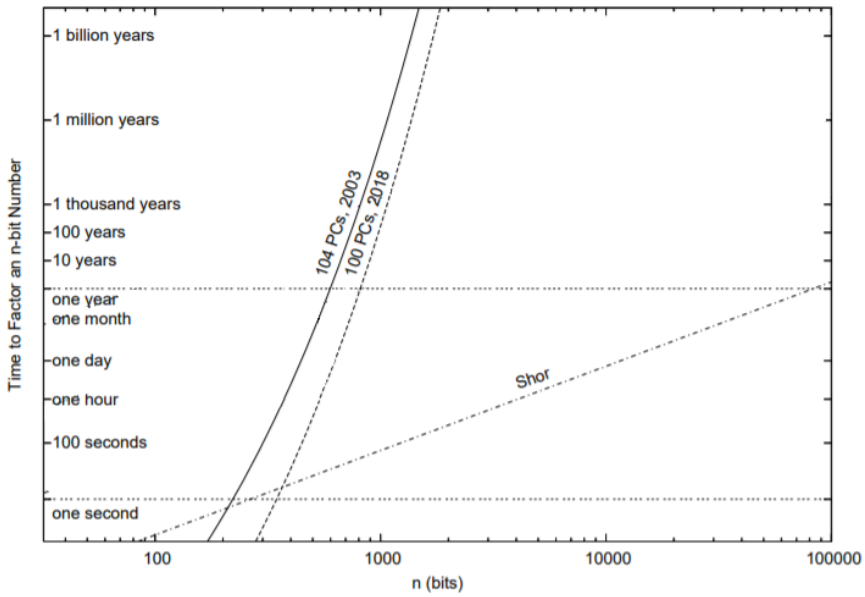


Figure 1-1: The complexity comparison of best performing Shor's algorithm vs. best known classical algorithm, adapted from [26]

of quantum encrypted communication. This could be used to create a pair of quantum encryption keys unique to channel. The state of the art research in quantum computing investigates the possibility and feasibility of applying quantum computing algorithms and devices to another active research area: machine learning. The study [5] shows that reasonable quantum speedup may be attained from quantum computers' efficiency in performing Basic Linear Algebra Subroutines. Moreover, it considers the reasonable possibilities of exploring quantum machine learning algorithms, for instance: Quantum Bolzman Machines, Quantum Principal Component Analysis, Quantum Least Squares Fitting and some other.

The combination of problems that quantum computers can overcome due nature of quantum computation and the possible gains in areas listed above outmatch the difficulties related to its construction. This makes research in area of quantum computing and related problems attractive and important.

1.5 The Proposed Quantum Evolutionary Algorithm

This research was inspired by the success of evolutionary approach to circuit synthesis and optimization [22, 34]. Moreover, since evolutionary algorithms were proven to benefit from Graphic Processing Unit accelerations [20], we planned to apply General Purpose GPU programming practices in our work. Furthermore, we plan to construct our algorithm in a way its parts could be executed on quantum computer, making several constraints and limitations that would affect our design choices. The previous study in the field outlines several possible improvements with respect to defining population in a more quantum compliant way, complexity of computation and difficulty in implementation of evolutionary operators. This algorithm follows general directions from existing quantum algorithms [7], however we differ from previous work by making emphasis on building a more quantum-compatible algorithm.

This thesis work covers:

- Quantum encoding for the evolutionary algorithm;
- Construction of evolutionary algorithm by selecting evolutionary operators that are more quantum compliant than classical evolutionary algorithm;
- Determine if the quantum evolutionary algorithms implemented on classical hardware are a viable approach to the problem of quantum logic design.

Chapter 2

Background

2.1 Quantum Information Theory

2.1.1 Definitions in Vector Form Notation

From a data representation point of view, the main distinguishing characteristic of quantum computers is the use of quantum bits, or *qubits*, instead of classical bits. While classical bits are limited to 0 or 1, the qubit represents the *superposition* of two fundamental states $|0\rangle$ and $|1\rangle$. Equation (2.1) demonstrates a single qubit quantum system state in vector notation:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.1}$$

for all possible assignments of complex coefficients α and β, such that the normalization condition (2.2) should always be satisfied:

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2.2}$$

A system of one or more qubits is called *quantum register*. The process of determining the value of a *quantum register* is called *measurement* [31]. Oversimplified, the *measurement* is a process of making a quantum system to become fixed at one of its states. The probability of finding a qubit in one or the other state is equal to

square of absolute value of *probability amplitude* - a scalar multiple of a state in the state equation. For instance, $|\alpha|^2$ is the probability of getting state $|0\rangle$ after measurement applied to $|\psi\rangle$ from Equation (2.1). The measurement process is not reversible and it is not possible to restore full information about measured qubit. One of the types of information that can be lost upon measurement is the *phase*. The quantum *phase* term meaning strongly depends on context [31]. For example, for θ being a real number and $|\psi\rangle$ being a quantum state, the probability amplitude $e^{i\theta}$ of a state $e^{i\theta}|\psi\rangle$ is called *global phase factor* [31]. Another phase-related keyword is notion of *relative phase*. For example the states $|0\rangle$ and $-|0\rangle$ are said "to differ by a *relative phase* if there is a real θ such that $a = exp(i\theta b)$" [31].

Quantum mechanics enables implementation of multiple valued logic. In a form of adding extra states. The quantum unit of information that can have three possible values is called qutrit. Qutrits are defined by superposition of three states, $|0\rangle$, $|1\rangle$ and, for instance $|2\rangle$:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle + \gamma|2\rangle. \tag{2.3}$$

By analogy, the normalization condition for qutrits (Equation (2.4)) constrains all possible assignments of α and β and γ as well

$$|\alpha|^2 + |\beta|^2 + |\gamma|^2 = 1. \tag{2.4}$$

The logic operations applied upon qubits are specified by unitary matrices. Equation (2.5) demonstrates an application of a NOT gate to a single qubit state (note the coefficients difference between Equation (2.1) and Equation (2.5)):

$$NOT|\psi\rangle = \beta|0\rangle + \alpha|1\rangle. \tag{2.5}$$

Furthermore, the logic in quantum circuits can be treated as rotations of qubit state. In order to explain and demonstrate such possibility, several operations should be performed. First, the Equation (2.1) should be transformed into polar form and thus,

rewritten to

$$|\psi\rangle = e^{i\gamma}(cos\frac{\theta}{2}|0\rangle + e^{i\phi}sin\frac{\theta}{2}|1\rangle) \tag{2.6}$$

where γ, θ and ϕ are real numbers [31]. However, the factor $e^{i\gamma}$ in this form could be omitted, because it has no *observable* effects [31] and thus, the equation can be finally transformed to:

$$|\psi\rangle = cos\frac{\theta}{2}|0\rangle + e^{i\phi}sin\frac{\theta}{2}|1\rangle. \tag{2.7}$$

This form provides the possibility of representing the single qubit in three dimensional space. This equation describes a unit three dimensional sphere that is known as Bloch sphere [31], demonstrated in the Figure 2-1. The Bloch sphere provides means for

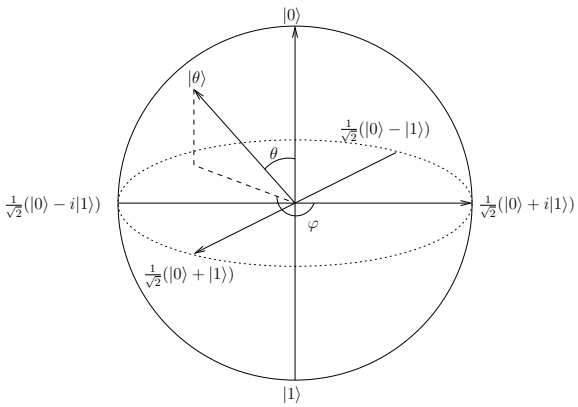


Figure 2-1: The Bloch sphere

visualizing single qubit rotations. Unfortunately, this model is not powerful enough to represent operations on multi qubit systems [31].

Considering the size of computational problems, having a single qubit systems would be insufficient for any meaningful task, multiple qubit systems should be used instead. Major advantage from exploiting *superposition* in multi qubit systems is that n qubits can represent 2^n values simultaneously. In addition to compact representation, this property enables the highly parallel nature of quantum computation [31]. Building a *quantum register* of two or more qubits can be done by applying the Kronecker product to the qubit states. Equation (2.8) displays an example of application of Kronecker product to combine qubits $|a\rangle$ and $|b\rangle$ to a two-qubit *quantum register*:

$$|\psi\rangle = |a\rangle \otimes |b\rangle = \alpha_a\alpha_b|00\rangle + \alpha_a\beta_b|01\rangle + \beta_a\alpha_b|10\rangle + \beta_a\beta_b|11\rangle. \tag{2.8}$$

The multi-qubit states that can be constructed by sequence of Kronecker products of the individual qubits are called *separable* states [6]. There are multi-qubit systems that are not in a *separable* state, but are rather in *entangled* state [31], the state that can not be split to a product of single qubet states. The *entanglement* effect may appear when logic gates affecting two or more qubits are applied in the circuit. This effect enables storing more information in quantum circuit. The *entanglement* phenomena is purely quantum mechanical and does not exist in classical logic, so any extra information carried in *entangled* state is also lost upon *measurement* [31]. It is possible to affect the states of single qubits in the multi-qubit system as long as it is in *separable* state. For instance, to negate the qubit $|a\rangle$ the X operator can be applied (note the coefficients reordered):

$$X|\psi\rangle = \beta_a\alpha_b|00\rangle + \beta_a\beta_b|01\rangle + \alpha_a\alpha_b|10\rangle + \alpha_a\beta_b|11\rangle. \tag{2.9}$$

2.1.2 Matrix Form

Since it is more convenient to represent logic functions as matrices for design purposes, the matrix notation for qubit representation is used in this research. The Equation (2.10) defines basic states of a single qubit:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{2.10}$$

This definition allows to rewrite the Equation (2.1) to become

$$|\psi\rangle = \alpha \times \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}. \tag{2.11}$$

Note that normalization condition from Equation (2.2) should also be satisfied in this case. Consequently, this means that all matrices are unitary - that is sum of squares

of absolute values of matrix terms on each row and column must be equal to one.

In matrix form, applying a function to a quantum state means multiplying the desired function represented in a matrix form by the state vector. For example, the Equation (2.5) in matrix form is

$$NOT|\phi\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix} \tag{2.12}$$

The Kronecker product can for matrices be defined as follows:
If $\boldsymbol{A}$ *is m x n matrix and* $\boldsymbol{B}$ *is* $p \times q$ *matrix, then the result of Kronecker product* $\boldsymbol{A} \otimes \boldsymbol{B}$ *is of size* $mp \times nq$ *calculated as it is displayed in the Equation* (2.13)

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \dots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \dots & a_{mn}\mathbf{B} \end{bmatrix}. \tag{2.13}$$

To demonstrate a multi-qubit system construction, the Equation (2.8) should be rewritten to:

$$|\psi\rangle = |a\rangle \otimes |b\rangle = \begin{bmatrix} \alpha_a \\ \beta_a \end{bmatrix} \otimes \begin{bmatrix} \alpha_b \\ \beta_b \end{bmatrix} = \begin{bmatrix} \alpha_a\alpha_b \\ \alpha_a\beta_b \\ \beta_a\alpha_b \\ \beta_a\beta_b \end{bmatrix}. \tag{2.14}$$

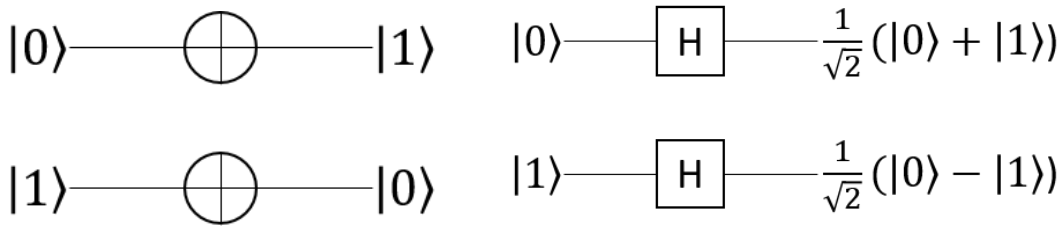
2.2 Quantum Circuits

2.2.1 Introduction to Quantum Circuits and Logic Design

Single-Qubit Gates

The logic construction for quantum computers is based on applying unitary operations and interactions of elementary particles used to build a quantum computer in the specified architecture [31, 20]. Regardless of architecture, quantum computers are convenient to implement classical reversible logic functions. A sequence of single and two-qubit operators (*gates*) applied to a quantum register is called a *quantum*

circuit. From circuit design perspective, qubits composing a quantum register can be alternatively called *wires*. The Figure 2-2 shows an example of a single qubit gates applied to *a*) $|0\rangle$ and *b*) $|1\rangle$ states. The Hadamard gate may be often used to introduce *entanglement* to the quantum circuit [31]. In this work, the entanglement might appear in case of two qubit interaction used in the circuit.



(a) (b)

Figure 2-2: Example of single qubit gates: a) quantum NOT gate, b) Hadamard gate

Alternatively, these single qubit gates could be represented in matrix form as it is described in Equation (2.15)

$$NOT = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \text{ and } H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.15}$$

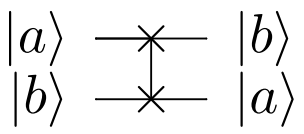
Multi-Qubit Gates

The quantum circuit design from quantum primitives is limited to use single and two-qubit gates for implementing logic functions. In order to represent logic of higher order the Kronecker product can be used. To build a multi qubit gate applied to a multi qubit system, the Kronecker product should be applied to all single qubit logic gates. If there is no logic applied to some wire, identity matrix of size two by two should be used instead. The Equation (2.16) displays procedure of construction of

NOT gate matrix applied to second wire of three qubit quantum register.

$$NOT_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.16}$$

One of remarkable multi-qubit gates is the two-qubit $SWAP$ gate. Basically, this quantum gate swaps the information contained in these two wires. The Figure 2-3 demonstrates its function and matrix:



(a)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.17}$$

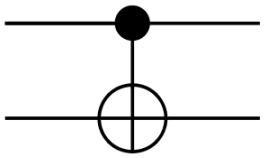
(b)

Figure 2-3: SWAP gate: a) the SWAP gate, b) the matrix of SWAP gate

Control Gates

In the multi-qubit systems, control gates can be defined as a special class of gates. This type of gates implements a function that is applied to one wire only if the other wires are set to some value, in case of reversible logic, the value of the control wires should be equal 1. The Figure 2-4 displays one of the simplest possible control gates, the Feynman gate and its function in matrix form. Feynman gate is alternatively called Controlled Not gate, in short, $CNOT$.

The wire marked with dot is called *control* wire, the wire with NOT gate displayed on it is called *target* wire. The general behavior of this logic gate can be described the following way: If the value on the control wire is one, then perform the operation



(a)

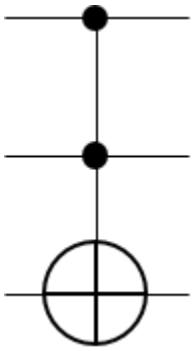
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.18}$$

(b)

Figure 2-4: Feynman Gate: a) the CNOT gate, b) the matrix of CNOT gate

specified on the target wire. There can be any arbitrary unitary gate in place of the NOT gate.

One of the most important controlled gates is the Toffoli function, $CCNOT$ or C^2NOT, which is Turing universal gate by itself. The Toffoli gate has AND and NOT in its truth table, which allows to implement any classical circuit by building cascades of Toffoli gates. The Figure 2.19 demonstrates its matrix representation.



(a)

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.19}$$

(b)

Figure 2-5: : The Toffoli Gate: a)the quantum gate, b) the function matrix

However the Toffoli gate itself is not sufficient for realization of an arbitrary quantum logic gate. The properties of quantum computational space, the quantum *phase* and *entanglement* require specific consideration because these properties are crucial for quantum speedup and utilizing the full information hidden in a quantum register. The Toffoli gate, realized on a quantum computer and thus, able to preserve quantum information inside of circuit is displayed on the Figure 2-6

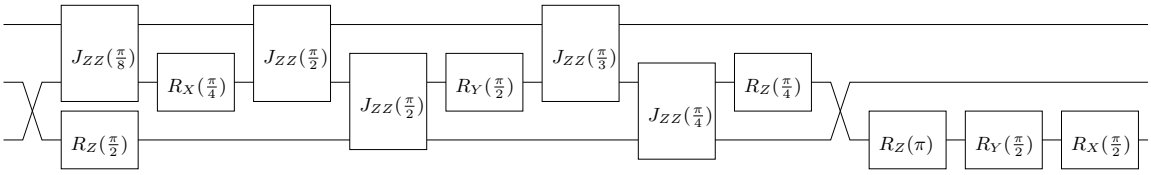


Figure 2-6: Example of efficient realization quantum Toffoli gate extracted from [20]

2.2.2 Models for Quantum Circuit Design

The fourth of the above presented Di Vincenzo criteria implies that the bigger a quantum circuit is, the more difficult it is to keep the system in the state appropriate for performing a computation. This reinforces the need for designing *optimal* circuits. In this context *optimal* means a circuit having the smaller *quantum cost* capable of performing the desired computation. The *quantum cost* of a circuit can be defined as a number of primitive gates required to build the desired function.

There are several possible models for quantum logic circuits design. The Figure 2-7 demonstrates realization of a reversible logic gate, called Peres gate, in different models varying by the gate set used:

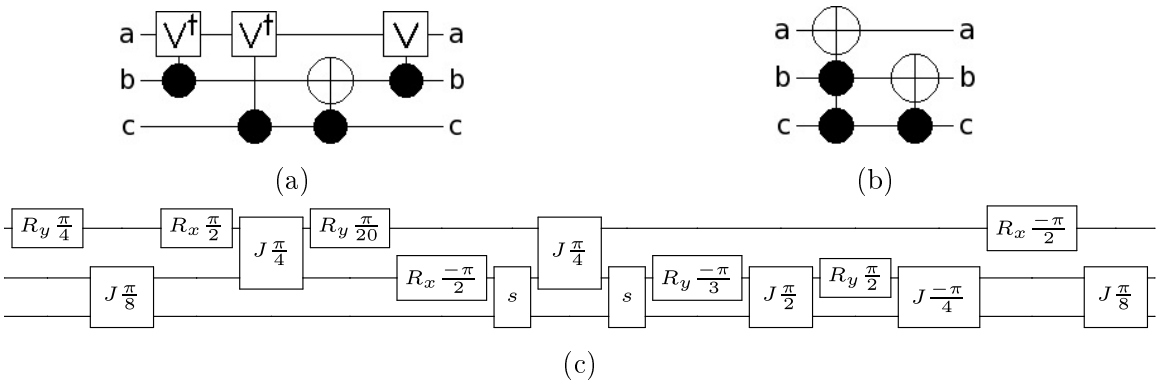


Figure 2-7: Realization of Peres gate in different models: a)Elementary Quantum Gates [40], b) Multiple Controlled Toffoli [40], c) Ising model from [20]

Any model consists of *primitive gates* sufficiently complex such that the gate set could be used to implement the *target* circuit. The *target* circuit is the circuit function the synthesis is aiming to build. Some of important models are the $CNOT/CV/CV^\dagger$, the Clifford-T or the CH/CZ or the Ising model. This research uses the Ising model for circuit synthesis.

Ising Model

The Ising model had success at physical implementation of quantum computers [38] performing Shor's factoring and inverse Quantum Fourier Transform. The elemental gate set for the Ising model consists of three single qubit gates representing rotations around the X,Y,Z axes of the Bloch sphere (see Figure 2-1) and two qubit Z interaction gate [17]. The next equations demonstrate the matrices of these gates:

- X direction

$$R_x(\theta) = e^{\left(\frac{-i\theta X}{2}\right)} = cos(\frac{\theta}{2})I_2 - isin(\frac{\theta}{2})X = \begin{bmatrix} cos\frac{\theta}{2} & -isin\frac{\theta}{2} \\ -isin\frac{\theta}{2} & cos\frac{\theta}{2} \end{bmatrix}. \tag{2.20}$$

- Y direction:

$$R_y(\theta) = e^{\left(\frac{-i\theta Y}{2}\right)} = cos(\frac{\theta}{2})I_2 - isin(\frac{\theta}{2})Y = \begin{bmatrix} cos\frac{\theta}{2} & -sin\frac{\theta}{2} \\ sin\frac{\theta}{2} & cos\frac{\theta}{2} \end{bmatrix}. \tag{2.21}$$

- Z direction:

$$R_z(\theta) = e^{\left(\frac{-i\theta Z}{2}\right)} = cos(\frac{\theta}{2})I_2 - isin(\frac{\theta}{2})Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}. \tag{2.22}$$

The template for the two-qubit interaction is:

$$J_{ij}(\theta) = e^{\frac{-i\theta}{2}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{\theta} & 0 & 0 \\ 0 & 0 & e^{\theta} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.23}$$

2.3 Logic Circuits Design

2.3.1 Problem of Logic Design

There are two main problems hindering the task of quantum or reversible logic circuits design directly from quantum primitives. The first problem is: the optimal solution for this task does not exist yet. The main cause is the complexity of the search for circuits having a number of qubits bigger than two. The complexity growth is due the nature imposed constraint for primitive logic gates: at most two-qubit gates can be used to accomplish the task. This limitation enforces utilization of composite gates with previously discovered physical realizations for logic implementation, which has the second problem enclosed in it. This approach makes designing bigger quantum and reversible logic susceptible to finding non-optimal solutions using quantum primitives. In other words, the available methods of discovering quantum and reversible logic circuits cannot guarantee finding of an optimal solution. The synthesis of reversible quantum gates such as gates from the C^nU family with U being NOT, or $SWAP$ or unitary operations, has been solved in general for some sets of Turing universal quantum gates and small number of qubits. For instance, the minimal realization of C^2NOT gate is known in the $CNOT/CV/CV^\dagger$, Clifford-T or CH/CZ set of quantum gates. However, in the Ising model the Toffoli gate is not known with certainty as the original specification was found by a stochastic algorithm [16] while in [20] an improved realization was found. Additionally, this situation only gets worse with larger logic gates, where synthesis is done by LUT (Look Up Tables) [36] or replacement of large gates by a group of smaller gates already known [37]. Thus, a synthesis method that designs larger quantum circuits directly using quantum gates would benefit from better minimal cost but also would require faster computers.

2.3.2 Application of Evolutionary Computation to Circuit Synthesis Problem

Evolutionary Computation is well suitable for problems having complicated search space. The evolutionary algorithms is type of algorithms inspired by observing natural processes [13]. Genetic algorithms are one kind of evolutionary algorithms. To perform evolutionary computation by means of genetic algorithm, the problem must be represented as a population of *genes* and *chromosomes*. For circuit design, *genes* may encode components of a circuit.

The circuit, constructed from multiple *genes* is used as a candidate for the solution. In terms of evolutionary computation, the composite circuit can be called an *individual*, alternatively a *chromosome*. Multiple *chromosomes* form a *population*. The population undergoes an *evolution* implementing the "survival of the fittest" strategy. The *evolution* is a meta heuristic based on constantly reducing the search space towards one of the optimal solutions, depending on the initial population assignment. The general flow of a genetic algorithm is described on Figure 2-8

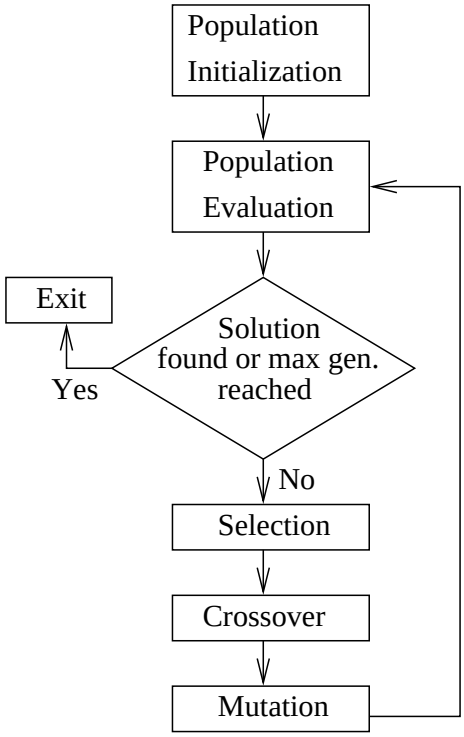


Figure 2-8: General evolutionary algorithm procedure

The Figure 2-8 describes the general flow of a classical evolutionary algorithm. The more detailed explanation follows below:

1. The step of population initialization consists of mapping a problem to the population suitable for evolution. Next, the individuals are randomly initialized following the constraints of the specified problem.

2. The population undergoes evaluation to calculate each candidates' similarity to the target solution. At that point, the *fitness value* is assigned to each of the individuals. If the solution meeting the desired tolerance was obtained or the maximum number of iterations is reached, the algorithm halts.

3. The next evolutionary operator is the selection. The purpose of this stage is to identify the best and the worst individuals to be processed during the next stage. The most famous approaches to the selection are Stochastic Universal Sampling, Roulette Wheel Selection or Tournament Selection [1].

4. The crossover operation is an exchange of the individual's genes. There exist several possible strategies to perform crossover operation. One of them is to combine and reorder the genes of two best candidates for the solution to construct a new individual. Later, this new individual replaces the ones identified during the selection stage.

5. The mutation operation prevents the population convergence to local maximum. It is a stochastic process introducing random noise to the population

6. Steps 2 to 5 are repeated.

2.3.3 Quantum Evolutionary Computation

A lot of work has been done for solving the problem in classical paradigm using different approaches and hardware, however the execution time is a limiting factor even for the most optimal evolutionary and general algorithms [22] directly designing quantum circuits. Thus Quantum and Quantum Inspired Algorithms were introduced

in order to reduce the computation time using principles of quantum mechanics. One of the first evolutionary algorithms inspired by quantum computing was developed in [29]. The most original idea was the extension of quantum inference crossover [29]. In [30] the first definition and requirements for evolutionary quantum algorithms have been introduced. The most important and challenging requirements are listed below for the clarity of understanding:

- A reasonable method of splitting the problem to sub-problems;
- "The number of universes required should be identified" [30], that is the number of quantum registers should be well described;
- The computations should occur in parallel;
- "There must be some form of interaction between all of the universes. The interference must either yield a solution, or new information for the universes to utilize in locating a solution" [30].

Several further studies described the Quantum Genetic Algorithms for general purposes [27] [24] such as for the knapsack problem. The problem of quantum circuits synthesis was studied using Quantum Evolutionary Algorithm (QEA) in [7]. The study [7] used integer representation of population, and demonstrated synthesis with multiple controlled NOT gates. In [23] the design of quantum circuits used qutrits for individual encoding. This allowed for more advantageous usage of mutation and ternary operators. In order to run these algorithms, most of the studies design special quantum encoding and mapping of evolutionary operators that could potentially allow to execute their algorithm on quantum computers.

Chapter 3

Quantum Evolutionary Algorithm for Design of Quantum Circuits

3.1 Quantum Encoded Quantum Evolutionary Algorithm

Our Quantum Evolutionary Algorithm for Quantum Circuit Synthesis was named Quantum Encoded Quantum Evolutionary Algorithm because it does not fit directly to any classification of existing algorithms and is not fully quantum per se. There are some parts that require classical control over quantum encoded population and operators. This chapter is dedicated to the description of the proposed algorithm, selected restrictions and optimization strategies used to overcome the raising difficulty of the search. The proposed approach features the following characteristics:

- population of solution candidates encoded using qubits and qutrits;
- adaptive mutation as the main driving force of the evolution;
- templates for building interaction gates;
- use of position in the memory to encode circuit information;
- synthesis on a level of single qubit rotations and interaction gates;

- predefined templates of interaction matrices for simplification of the search;
- ensemble-quantum computer inspired set of evolutionary operators;
- measurement based quantum gate and quantum circuit creation.

The general flow of the proposed algorithm is depicted in Figure 3-1. The QE-QEA does not evolve circuits directly; instead a set of quantum gates (segments) are evolved as a population. The circuits are obtained by random selection of gates from the population. Each gate is encoded by several quantum parameters and uses measurement procedure for circuit construction.

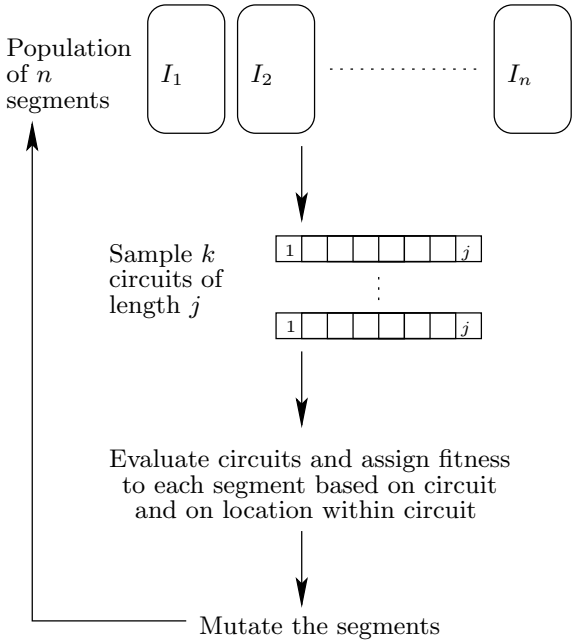


Figure 3-1: High level flow of the quantum evolutionary algorithm

3.2 Quantum Gates Representation

The QEQEA is constructed to synthesize gates in Ising model. Despite the fact this model is considered impractical due to difficulties growing with the quantum register size. The synthesis of logic gates using the Ising model is considered one of the most complex tasks in quantum logic synthesis. Thus, generating results in this model is

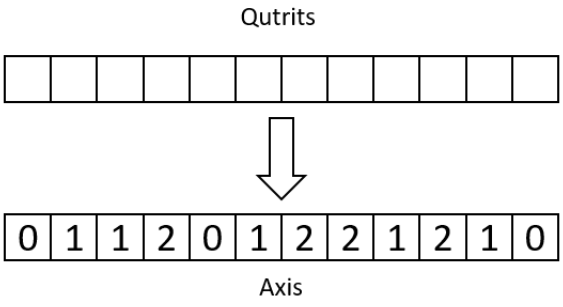
an indicator of the performance of the applied algorithm and this is the main cause for chosing this model to be explored in this research. The primitive gate set is single-qubit rotation gates and two qubit gates. Construction of each circuit segment requires a qubit and, for rotation gate qutrit with our proposed encoding.

3.2.1 Rotation gates

The single qubit gates ($R_X(\theta)$, $R_Y(\theta)$ and $R_Z(\theta)$) are encoded using one qubit and one qutrit. The angle of rotation θ is represented by the qubit parameter specifying its complex amplitudes: $e^{-i\pi\theta}$. The axis of rotation is obtained by measuring the state of the qutrit. We repeat the measurement process multiple times to approximate the state of the qutrit, without eliminating uncertainty. For qutrit state $\psi\rangle = \alpha|0\rangle + \beta|1\rangle + \gamma|2\rangle$ the qutrit states: $\{|0\rangle, |1\rangle, |2\rangle\}$ correspond to rotations around $\{x, y, z\}$ axis, respectively. The Figure 3-2 demonstrates the pseudo code for the measurement procedure:

$r \leftarrow random[0, 1]$
$axis \leftarrow z$
if $r \leq |\alpha|^2$ **then**
    $axis \leftarrow y$
else if $r \leq |\alpha|^2 + |\beta|^2$ **then**
    $axis \leftarrow x$
end if

(a)



(b)

Figure 3-2: Measurement simulation: a) pseudo code for measurement simulation, b) parallel axis decoding from qutrits

3.2.2 Interaction gates and templates

The second type of quantum gate we use is the two-qubit interaction. The interaction gate is equivalent to two parameterized single-qubit Z gates applied simultaneously to two qubits [16]. In the GPGPU version of an algorithm, the number of parameters required to encode interaction gate was three: two indexes on which the gate operates

and the θ parameter. By introducing interaction matrices *templates*, we reduced the number of parameters required to construct the interaction gate to one. The parameter θ obtained by copying qubit value similar to the case of single qubit gates construction. The other two parameters, the indexes, are encoded in position within the memory. There are $\binom{numberOfWires}{2}$ distinct templates possible for circuit with input size $numberOfWires$. Utilization of the templates allows to exclude $SWAP$ gates that were used to simulate interaction gates between non-neighboring qubits in GPGPU version [20] during process of evolution.

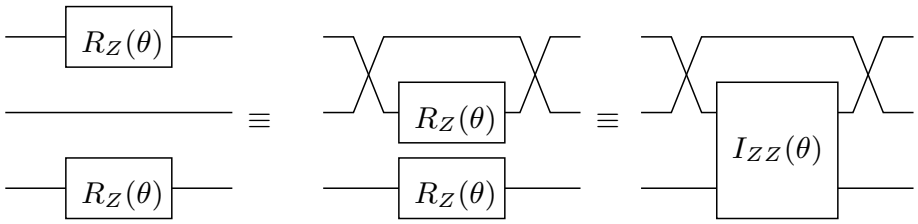


Figure 3-3: Construction of templates for interactions between non-neighboring qubits

The interaction gate can be expressed in form of term-wise exponent of scalar multiple of gate parameter value and special diagonal matrix, *template*. The *template* is a diagonal matrix which diagonal entries are are $+1$ and -1 depending on the wires on which the interaction is applied. Notice that the number of possible interaction gates with respect to wires of their application grows slowly considering the size of the problem. The algorithm for creation of the templates and construction of gates from them can be implemented in three stages:

1. During the preparation stage of the algorithm, the template matrices for non-neighboring qubits are constructed as it follows from the Figure 3-3 .

2. When an interaction gate is to be inserted in a quantum circuit, the template should be multiplied by qubit value.

3. Parallel exponent of elements of diagonal matrix should be calculated.

This optimization potentially allows synthesize more optimal circuits by removing redundant swap gates.

3.3 Population Initialization

The qubits and qutrits are undergoing evolutionary process in the proposed Quantum Encoded Quantum Evolutionary algorithm. These qubits and qutrits are used to encode quantum gates that in turn are sampled to form multiple quantum circuits.

The GPGPU version of the algorithm has the wire on which the logic gate should operate as a parameter. To eliminate this parameter in QEQEA, the position of the segment in the population fulfills the role. In essence, the population size is increased proportionally to the problem size to account for gate on each possible wire.

The qubits population can be divided into two parts: the qubits encoding rotation gates and the qubits encoding the two-qubit interaction gates. The following parameters define the population:

- $sizeOfIndividual$ defines the length of the circuit in terms of number of gates (segments)
- $sizeOfPopulation$ sets the number of individuals in the population. This parameter increases the number of segments in population to:

$$sizeOfPopulation * sizeOfIndividual \tag{3.1}$$

- $numberOfWires$ affects multiple aspects of the algorithm. As it was described before in Section 3.2.2, it defines the $interactionTemplatesNumber$. Thus, the $numberOfWires$ affects the segments count in the population to

$$(interactiontemplatesChose + numberOfWires) * sizeOfPopulation * sizeOfIndividual \tag{3.2}$$

Since the qutrits are required only to encode axes of rotation for single-qubit gates, the number of qutrits is fixed at

$$numberOfWires * sizeOfPopulation * sizeOfIndividual \tag{3.3}$$

The number of individuals in the population raises the amount of initial information to explore. It also significantly increases the computational complexity. However, the tasks required to synthesize one individual may be executed in parallel and we aim to get most benefit of highly effective parallel capabilities of quantum computers at that stage of the algorithm.

Note that the implementation of the encoding is aimed to on one hand exploit quantum parallelism and direct encoding of quantum circuits directly on qubits and on the other hand is targeted to be accelerated with current classical parallel technologies. The proposed encoding is intended for ensemble quantum computers such as using the NMR approach or one-way quantum computer where many of the same qubits exists and thus, many samples can be obtained.

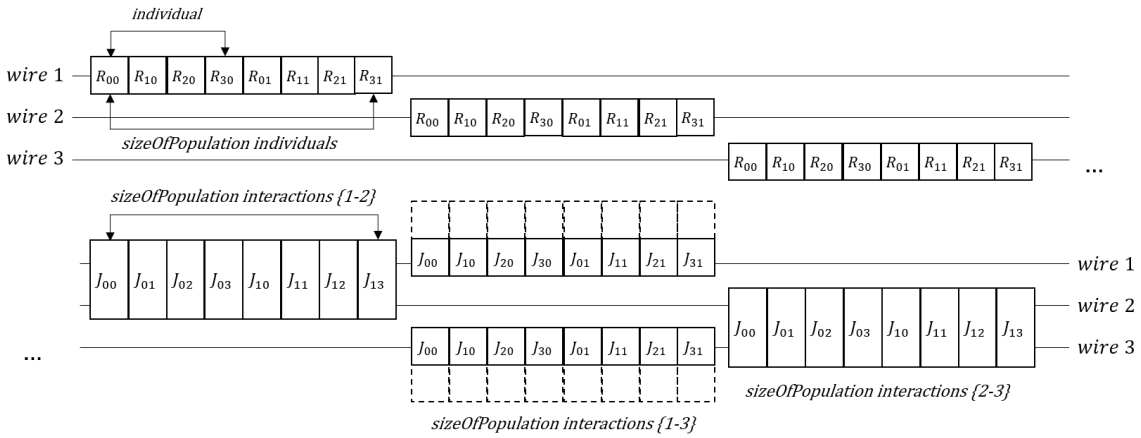


Figure 3-4: Segments layout with $numberOfWires = 3$, $sizeOfPopulation = 2$ and $sizeOfIndividual = 4$

Figure 3-4 describes an example population that would have two individuals, targeting to synthesize the circuit consisting of four gates applied to three input qubits (wires). The first eight qubits encoding the circuit segments correspond to rotation applied on the first input wire (labeled "wire 1" in Figure 3-4). There are exactly eight qubits in this particular case because the population consists of two individuals of size four. Similarly, the next eight qubits correspond to rotation on the second wire (labeled "wire 2" in Figure 3-4). Same rules apply to the third set of eight qubits. The remaining twenty-four qubits do not have qutrits allocated for them because they

belong to interaction region and use precalculated templates instead of measured axis (labeled "Interactions" in Figure 3-4). The Figure does not contain qutrits in it, the qutrits are described on the Figure 3-7.

3.4 Circuit Construction

3.4.1 Circuit Segments

Previous research in design of circuits using the Ising model [19] suggests using *circuit segments* for simplification of synthesis task. Listed without changes to preserve the original meaning, the circuit segment was defined in [19] as:

"A Segment of Quantum Circuit is another Quantum Circuit or a Quantum Gate of width n, such that it is built only by using Kronecker product between its component gates".

In our research, we do synthesis using circuit segments. Additionally, we restrict circuit segment having only one logic gate expanded to full circuit width. This restriction makes size of the circuit become sort of measure of quantum cost (special account should be done for two-qubit logic gates). Figure 3-5 helps illustrating the definition of a circuit segment:

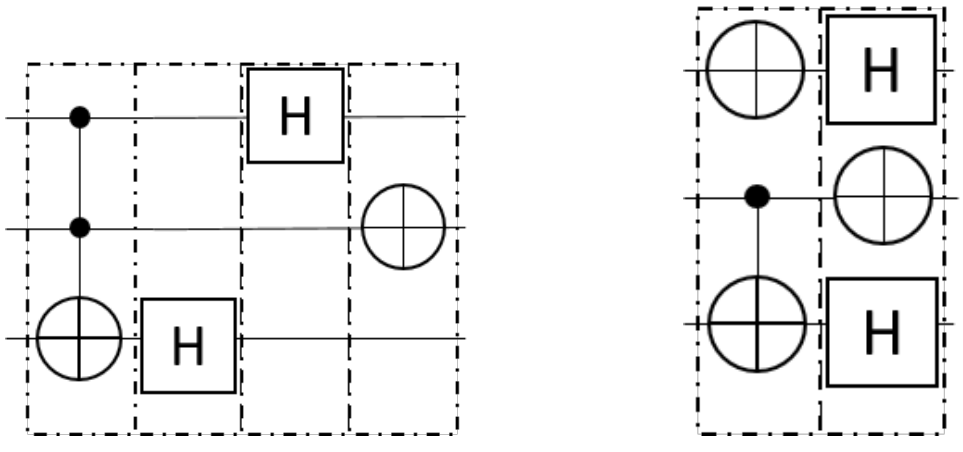


Figure 3-5: Possible logic layout within a quantum circuit: a) circuit split to segments, b) segments are not enforced

3.4.2 Segments Construction

Since the "one logic gate per circuit segment" approach was successful in the GPGPU algorithm [20], our algorithm performs same operation. To build such segments from the population of qubits and qutrits, it has to undergo multiple steps, including expansion by Kronecker product. The procedure for parallel Kronecker product expansion was first described in [20], however it was not utilized to apply the procedure to all individuals simultaneously. In the QEQEA, the logic gates were laid out in memory in a way to fully exploit parallel acceleration of this complex task. The Figure 3-6 demonstrates difference between classical single threaded approach with parallel version of algorithm, the arrows represent simultaneous threads.

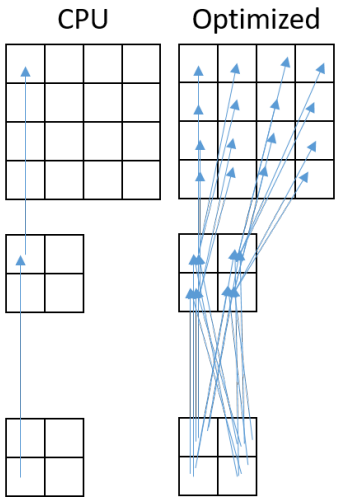


Figure 3-6: Classical and Parallel procedure of Kronecker product

The full segment construction requires performing four steps which are illustrated in Figure 3-7.

- Step 1: Obtain array of axes by performing measurement of qutrits
- Step 2: Insert qubit values to corresponding templates (rotations or interactions)
- Step 3: Prepare the memory for application of Kronecker product and apply the procedure

- Step 4: Construct interaction matrices in a way it was described in Section 3.2.2 and put next to segments obtained from rotation matrices

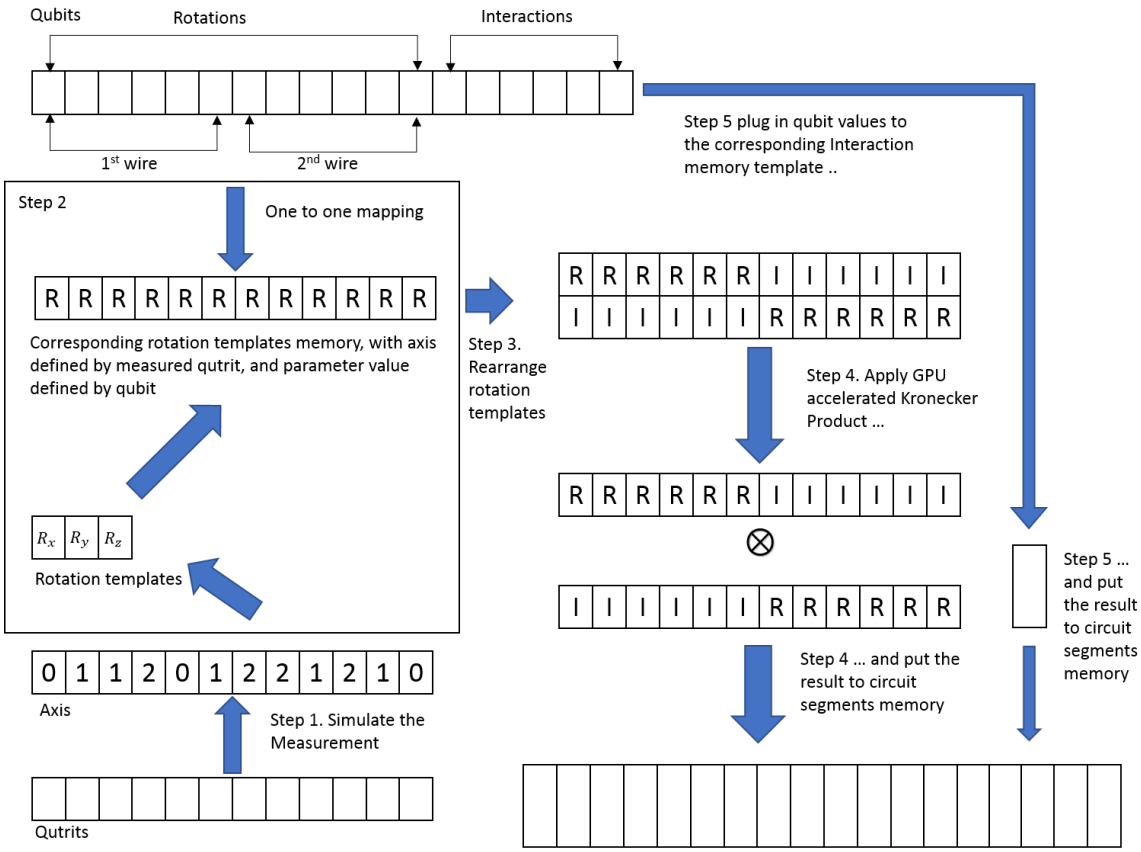


Figure 3-7: The procedure of segments construction prior circuit building stage

3.4.3 Circuit construction

The proposed quantum circuit design method is a form of evolutionary algorithm heavily altered in order to allow some of its components to be directly mapped into a quantum computer. Additionally, the proposed algorithm is also intended to be efficiently implementable on a highly parallel device such as GPGPU.

For the circuit of length of $sizeOfIndividual$ we launch $sizeOfIndividual$ parallel threads each indexed by $index_{thread}$. Each thread generates two random numbers:

- $whichIndividual$ from range $0..sizeOfPopulation$

- $whichRotationOrInteraction$ from the range
  $0..numberOfWires + interactionTemplatesNumber$

These values are later used to calculate the index of segment to be plugged in the circuit:

$$\begin{aligned} segmentIndex = {} & whichRotationOrInteraction \\ & * sizeOfIndividual * sizeOfPopulation \\ & + whichIndividual * sizeOfIndividual + index_{thread} \end{aligned} \quad (3.4)$$

The result of calculation is stored as reference to a segment in population for $index_{thread}$ position in the circuit. An example of process is shown in the Figure 3-8. We repeat

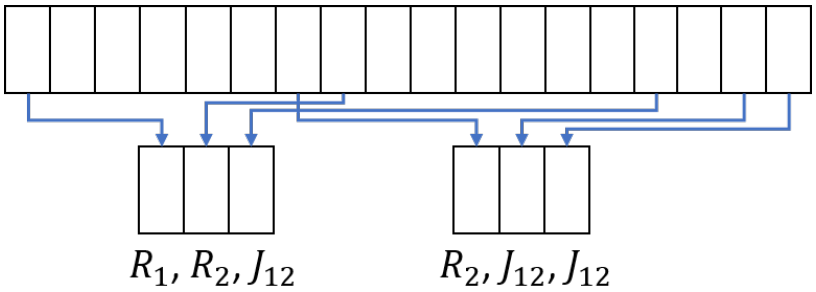


Figure 3-8: Building a circuit of length three affecting two qubits having two individuals in the population

this process $sizeOfPopulation$ times to generate $sizeOfPopulation$ circuits each iteration.

3.5 Fitness Evaluation

The fitness value reflects the proximity of the synthesized circuit matrix S, to the target circuit matrix T. The possible values of selected function are ranging from 0 to 1, where 1 represents identical matrices and 0 being the opposite. The fitness function used in [20] was initially used for driving the evolution. The error was accumulated by the sum of term-wise squared difference between circuit matrix and

target function matrix, shown on Equation (3.6):

$$error = \sum_{j=0}^{4^{numberOfWires}-1} [abs(|S|)^2 - abs(|T|)^2] \tag{3.5}$$

$$fitnessValue(S) = \frac{1}{1 + error} \tag{3.6}$$

Another approach that is more suitable for quantum computers is based on property of unitary matrices that $U^{\dagger} * U = I$. The Equation 3.7 displays the Quantum fitness function:

$$fitnessValue(S) = 1 - \sqrt{\frac{size - |tr(S^{\dagger}T)|}{size}} \tag{3.7}$$

In this expression, the || operator denotes absolute value. The tr operation represents calculation of the sum of diagonal elements of the matrix. The $size$ is normalization constant and is taken to be equal to $2^{numberOfWires}$.

The classical fitness function from Equation (3.6) is targeted to reduce the error between the terms of the matrix, however it does not differentiate between elements of the matrix having complex terms in it. Thus, Classical fitness function should be used only for reversible and classical logic synthesis.

3.5.1 Segment Fitness

Each segment used during circuit construction stage (Section 3.4.3) is assigned with a fitness value. The fitness value assigned to each segment is the same as the fitness value of the circuit it was used to construct.

Additionally, an elitist approach was implemented: if the new fitness value of a segment is better than the previous best value, the states of the qubits and qutrits are preserved, otherwise they get discarded.

Finally, each segments fitness is tied to a particular position in a given circuit. That is, the same segment will be represented by various fitness values depending on the position where it was located within the synthesized circuit.

3.6 Evolutionary Search

The main distinguishing characteristic of QEQEA compared to GPUGA is the crossover operation was left out. Because of that, the main source of changes in QEQEA is the mutation operation. We use adaptive mutation inspired from the evolutionary strategies approach [2]. The adaptive mutation approach sets the changes in individuals (qubits in qutrits) to account with the fitness value. The fitness value assigned to segments is responsible to changes applied to population to qubits and qutrits. For example, $1 - segmentFitness$ coefficient may serve the purpose of implementing adaptive mutation approach. In other words, better individuals undergo less significant changes [13]. This approach is argued to be more effective than the mutation with constant probability and mutation range [25]. To strengthen the control over the population, the elitism strategy was implemented in the process of evolution. The elitism implies copying the segments that were best in the iteration and save it to the next generation. The application of this strategy is determined by parameter.

Each individual undergoes change per iteration of our algorithm with $probabilityOfMutation$. Every time the mutation is to be performed, there are two equiprobable operations that may happen: qubits or qutrits mutation.

- The qubits mutation is a process of modifying the parameter representing qubit inversely proportional to the segment fitness value. Similar to our previous research, we use the $mutationRange$ parameter that determines the maximum possible change to parameter. In our algorithm, it is taken to be fraction of π. The final formula to calculate the mutation value is shown in Equation (3.8)

$$\pm(1 - segmentFitness) * mutationRange. \tag{3.8}$$

  The qubit parameters are assumed to stay within $[0, 2 * \pi]$ range, so after the mutation the resulting parameter is readjusted modulo $2 * \pi$

- The qutrits mutation is performed by applying the arbitrary SU(3) rotations on a qutrit [39]. Such matrix can be generated using eight parameters: three

$$\begin{bmatrix} e^{i\phi_1}c_1c_2 & e^{i\phi_3}s_1 & e^{i\phi_4}c_1s_2 \\ e^{-i\phi_4-i\phi_5}s_2s_3 - e^{i\phi_1+i\phi_2-i\phi_3}s_1c_2c_3 & e^{i\phi_2}c_1c_3 & -e^{-i\phi_1-i\phi_5}c_2s_3 - e^{i\phi_2-i\phi_3+i\phi_4}s_1s_2c_3 \\ -e^{-i\phi_2-i\phi_4}s_2c_3 - e^{i\phi_1-i\phi_3+i\phi_5}s_1c_2s_3 & e^{i\phi_5}c_1s_3 & e^{-i\phi_1-i\phi_2}c_2c_3 - e^{-i\phi_3+i\phi_4+i\phi_5}s_1s_2s_3 \end{bmatrix} \tag{3.9}$$

Figure 3-9: SU3 matrix construction, where $c_k = cos\theta_k$ and $s_k = sin\theta_k$, from [39]

rotation angles $\theta_1, \theta_2, \theta_3$ from range $0 < \theta < \pi/2$ and five phases $\phi_1, \phi_2, \phi_3, \phi_4, \phi_5$ from range $0 < \phi < 2 * \pi$. We construct the matrix using template described in Figure 3-9.

During one step of mutation, one of these nine parameters is generated randomly from a domain of its possible values multiplied by $1 - segmentFitness$. After the rotation matrix is constructed, it is applied to vector representing qutrit to produce an updated qutrit value.

Chapter 4

Results and discussion

4.1 Results

4.1.1 Evaluation of QEQEA

To verify the QEQEA algorithm we tested it on several quantum gates: Worst Case (3_17) [41] , Miller gate [41], C^2NOT, Peres and $CNOT$. Table 4.1 shows the results of the search for the $CNOT$ gate.

The Table 4.1 presents the outputs from the algorithm obtained in the process of synthesizing a CNOT gate. This gate was the only type of gate we were able to find exact representation. Each row in the table from top to the bottom represent encoded circuit segments in the order they appear in the synthesized circuit. Each row of the table contains all information required to decode information about circuit segment. The first column contains the parameter value θ representing the rotation. The second column determines whether the parameter θ should be plugged to rotation or interaction template. The third column of the table contains the states of the qutrit, which after measurement indicate the direction of the rotation gate. The value of this column should be ignored if the segment is a two-qubit interaction. The fourth column indicates the axis of rotation obtained as a result of measurement.

Thus, Table 4.1 represents a CNOT circuit constructed using the following sequence of gates:$R_{1y}(\theta = 1.570796)J_{12}(\theta = 4.712389)R_{1x}(\theta = 4.712389)$.

Table 4.1: Result of CNOT gate synthesis ($sizeOfIndividual$=3, $sizeOfPopulation = 1$)

Parameter θ	Index in memory	Qutrit states	Axis
$\pi/2$	0	-0.43 - 0.16i; 0.85 + 0.08i; 0.03 - 0.24i	y
$3\pi/2$	7	Interaction template between 1 and 2	
$3\pi/2$	2	0.39 - 0.66i; -0.43 + 0.43i; 0.16 - 0.14i	x

Equation (4.1) shows the resulting matrix of the obtained C^2NOT gate with the length of 16 segments. Some terms of the matrix have differences from original Toffoli gate therefore the circuit obtained is not exact, however on average the error per term is ≈ 0.02.

$$\begin{bmatrix} 0.894 & 0.000 & 0.004 & 0.000 & 0.101 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.916 & 0.000 & 0.001 & 0.000 & 0.080 & 0.000 & 0.004 \\ 0.004 & 0.000 & 0.967 & 0.000 & 0.000 & 0.000 & 0.029 & 0.000 \\ 0.000 & 0.004 & 0.000 & 0.121 & 0.000 & 0.000 & 0.000 & 0.875 \\ 0.101 & 0.000 & 0.000 & 0.000 & 0.894 & 0.000 & 0.004 & 0.000 \\ 0.000 & 0.080 & 0.000 & 0.004 & 0.000 & 0.916 & 0.000 & 0.001 \\ 0.000 & 0.000 & 0.029 & 0.000 & 0.004 & 0.000 & 0.967 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.875 & 0.000 & 0.004 & 0.000 & 0.121 \end{bmatrix} \quad (4.1)$$

4.1.2 Experiments Description

The final version of described algorithm was tested by executing the software multiple times. Two types of results were gathered: one for fitness function described by Equation (3.6), denoted as "Classical Fitness function" in the Table 4.2. The other result is for fitness function labeled as "Quantum distance measure" and is described by Equation (3.7).

The contents of the table reflect multiple outcomes of QEQEA

1. **Fitness function**: The change from was enforced by the original idea of making the implementation more quantum compatible. The quantum fitness function seems to create less evolutionary pressure and loosened control over the popu-

Table 4.2: Experiments results

Algorithm	Classical Fitness function	Quantum distance measure
Quantum implementable	worse	better
Number of runs	21	8
Average fitness	0.697	0.5634
Exact three-qubits gates	0	0
Elitism enabled	10	0

lation, evaluating received results.

2. **Number of Runs**: From 116 experiment conducted, only 30 were selected. The other results are excluded because there were circumstances affecting the purity of experiments. Significant (≈ 70) number of samples was rejected because the used approach was significantly different from the one presented in this work. Either there were bugs in implementation or the optimization strategy was considered negatively affecting the algorithm. The dominance of experiments with Classical Fitness Function is because the change of function happened on relatively late stage of work.

3. **Average fitness**: The big difference in average fitness might be based on difference in sample size. Another possible reason is that there was no specific parameters tuning applied after Quantum Fitness function was introduced.

4. **Exact three-qubits gates synthesized**: Very disappointing aspect of results, but the QEQEA was never able to repeat success of GPGPU and find a gate with 100% accuracy. We assume, this is due to lack of tools of population control available.

5. **Elitism enabled**: The elitist approach is prone to convergence to local maxima which appears in all our experiments. After several experiments this option was disabled and never turned on again. Another reasoning, elitist approach would requires classical control added to quantum computer running the algorithm.

4.2 Comparing QEQEA and GPUGA

The QEQEA was built as an extension for the algorithm from [20]. The main purpose for construction of this algorithm was to test whether quantum operators could be successfully implemented on classical hardware and to evaluate usefulness of this approach. In order to do that, several optimizations outlined above were implemented on GPU. The non-quantum GPUGA algorithm was used for comparison because there is common algorithmic and acceleration basis. The main features of GPUGA are:

- Representation: same mapping from memory to individual was implemented. The representation of quantum gate (segment) was performed using a set of real and complex coefficients.
- The Evolutionary operators: two point crossover was used and the mutation was a random small alterations of the gate parameters.
- Selection was using the Stochastic Universal Sampling (SUS).
- Evolution occurred on the level of level of circuits, not on the individual gates (segments).
- In the GPUGA no qutrits were used; we introduced the qutrits in QEQEA in order to avoid allocating extra memory for each type of the rotation gates (x,y,z) direction. This evolution of qutrits could possible reduce computation time required for each population step.

The GPU acceleration and parallelism makes the two algorithms alike. The advantages and disadvantages of the new algorithm are outlined below:

- Improvements the QEQEA introduced:
  - Full utilization of Kronecker product;
  - The interaction matrices construction using templates.
- Drawbacks due to constraints and limitations:

  - Qutrits measurement introduced huge computation overhead compared to the GPGPU algorithml
  - Lack of crossover operation significantly affects the convergence of the algorithm in negative manner.

- Changes with complex effect:
  - Mutation operation for each individual became easier due to reduced number of parameters required to encode gate;
  - Procedure of circuit construction became simpler for each iteration and is just random numbers without control, however less control may be the cause of slower convergence rate to the solution;
  - Fitness function change affected convergence of algorithm, but due to insufficient number of results collected, the statistics can not be built to say which one.

The Table 4.3 shows the differences of speed in obtaining the various gates for which we tested both algorithms. First column, labeled "Accuracy" presents the similarity of obtained circuit matrix with target circuit matrix. Notice that in all cases the classical algorithm was faster than the QEQEA algorithm (iteration of QEQEA takes significantly more time). Thus, even if the iteration number is smaller in QEQEA, the GPUGA is faster in real time and was able to converge to better results. The reason is the fact that the QEQEA is evolving gates rather than whole circuits while the classical GA evolves whole circuits. Additionally, the QEQEA generates solutions from a single set of encoding qubits and qutrits. As such there is no crossover because there is only one individual of qubits and qutrits. Consequently, because the main evolution mechanisms are selection and mutation, the proposed QEQEA is more related to evolutionary strategies rather than to genetic algorithm.

The first and the third columns of Table 4.3 display the accuracy of best results achieved by each algorithm. The iterations number could also serve as a measure for performance comparison, however for the QEQEA this data is only partially available.

Table 4.3: Comparison of Results and performance between the QEQEA and a classical GPGPU

Function	QEQEA		GPGPU	
	Accyracy	No. Generations	Accuracy	No. Generations
CNOT	1.0000	400	1.0000	200
Toffoli	0.7047	13000	0.9663	34500
CCCNOT	0.6464	limit	0.7539	650970
Peres	0.5693	limit	0.9443	2M

The reason for that is the search of CCCNOT and Peres gates reached the maximum iterations limit of ten million iterations. However, this fact also means the result could be possibly improved if the higher limit for iterations was set.

4.3 Discussion and Future Work Suggestion

The main problem with the proposed algorithm as we put it is lack of control over the population. This problem appears from the original design choice that was investigated. Nevertheless, the control could still be executed by means of adaptive mutation as we tried to explain.

- First point to improve, SU(3) rotations applied to modify qutrits encoding actions requires careful investigation. Alternatively, the population memory can be increased three times in size to disregard the mutation, measurement and the SU(3) mutations completely.

- Second point to investigate is to account more with the periodicity of change of the single-qubit gates to improve the rotation gates change. In particular, the limits for *mutationRange* parameters should be set depending on the problem size as it was in GPGPU algorithm.

- Introducing penalty for interaction gates gives room for improvement, because classical circuit built exclusively from interaction gates is a local maxima for big number of control gates. At the same time, it is just an identity gate.

- Since the proposed algorithm implements evolutionary strategies, it is said to converge to solution in limited amount of time. So another suggestion is to perform more rigorous testing after verifying algorithm once again.

- As an important extension of the algorithm may be restarting synthesis after convergence to local maxima achieved. The extension should target to find the $Target \oplus Result$ which later can be used to find better results

- The algorithm could significantly benefit from an extension that would allow preserving the sequences of segments. Now the segments are tied to position in the circuit but grouping them up and calculate fitness of group of segments might help.

4.4 Conclusion

An attempt to discover new algorithmic optimizations from quantum paradigm and feasibility of implementing of QEQEA was made by means of comparison with a version of classical GA. The QEQEA features certain components being a possible target for implementation in a quantum computer but in order to keep the implementation computationally tractable several design choices were applied that made it impossible to port directly to a quantum computer.

The work on the proposed problem led to obtaining two positive and two negative results. The results can be briefly summarized as:

- The algorithm encoded in terms of quantum units of information can find non-exact circuit realizations for known gates;

- The algorithm performance does not benefit from selecting more quantum compliant evolutionary operators;

- Several techniques described in this work can be adapted for improvements in other existing algorithms;

- The quantum evolutionary algorithm converges slower than the classical genetic algorithm.

In other words, even if all components of the algorithm were made quantum-implementation compatible, many components would remain classic. In particular this means, that even if QEQEA evolutionary components are mapped to a quantum computer, fitness function values, circuit information, algorithm flow control and other parameters require to be kept in a classical memory. The comparison with the classical GPUGA showed that the quantum evolutionary model shows worse performance than the classical evolution. The inferior performance is due to many constraints included in the QEQEA that resulted in strong simplification of the evolutionary process. Consequently the main result is that the evolutionary process for computation as originally proposed in [12] seems to be most efficient when implemented in classical computer. In a quantum computer, an efficient implementation requires exploitation of the entanglement property that would made the search much more efficient. However, simulating such system on classical computer requires high computational resources and cannot easily be compared to a classical GA.

Appendix A

Brief Software Package Description

The software package contains GPU accelerated version of QEQEA that can be used on any GPU device having compute capability 3.0 and above. The execution was performed on Tesla K40 GPU accelerator. The software can perform synthesis of gates specified in .pla format [40]. The synthesis can only be performed for gates without don't cares.

In addition to the QEQEA the package contains MATLAB scripts enabling verification of results achieved from compiled GPU code and means of verification of gates built in MATLAB.

The code repository is hosted on bitbucket.org website and can be sent out upon request.

Bibliography

[1] Firas Alabsi and Reyadh Naoum. Comparison of selection methods and crossover operations using steady state genetic based intrusion detection system. 2012.

[2] Anne Auger. Convergence results for the $(1, \lambda)$-sa-es-irreducible markov chains. *Theoretical Computer Science*, 334(1-3):35–69, apr 2005.

[3] P. Benioff. Quantum mechanical Hamiltonian models of Turing machines. *Journal of Statistical Physics*, 29(3):515–546, 1982.

[4] C. Bennet. Logical reversibility of computation. *IBM Journal of Research and Development*, -:525–532, 1973.

[5] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, sep 2017.

[6] Eli Biham, Gilles Brassard, Dan Kenigsberg, and Tal Mor. Quantum computing without entanglement. 2003.

[7] Shengchao Ding, Zhi Jin, and Qing Yang. Evolving quantum circuits at the gate level with a hybrid quantum-inspired evolutionary algorithm. *Soft Computing*, 12(11):1059–1072, Sep 2008.

[8] David P. DiVincenzo and IBM. The physical implementation of quantum computation. 2000.

[9] Jérôme Durand-Lose. Computing inside the billiard ball model. In *Collision-Based Computing*, pages 135–160. Springer London, 2002.

[10] M H. Devoret, Andreas Wallraff, and J.M. Martinis. Superconducting qubits: A short review. 12 2004.

[11] S. M. Hamdi, S. T. Zuhori, F. Mahmud, and B. Pal. A compare between shor's quantum factoring algorithm and general number field sieve. In *2014 International Conference on Electrical Engineering and Information Communication Technology*, pages 1–6, April 2014.

[12] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[13] Libin Hong, John H. Drake, and Ender Özcan. A step size based self-adaptive mutation operator for evolutionary programming. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO Comp '14, pages 1381–1388, New York, NY, USA, 2014. ACM.

[14] R. W. Keyes. Fundamental limits of silicon technology. *Proceedings of the IEEE*, 89(3):227–239, Mar 2001.

[15] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 1961.

[16] S. Lee, S. J. Lee, T. Kim, J. S. Lee, J. Biamonte, and M. Perkowski. The cost of quantum gate primitives. *Journal of Multiple-Valued Logic and Soft Computing*, 5-6(12), 2006.

[17] S. Lee, S.J. Lee, T. Kim, J.S. Lee, J. Biamonte, and M. Perkowski. The cost of quantum gate primitives. *Journal of Multiple-Valued Logic and Soft Computing*, 12(5-6):561–574, 2006.

[18] Ming Li and Paul VitÃąnyi. Reversibility and adiabatic computation: trading time and space for energy. *PROC ROYAL SOCIETY OF LONDON, SERIES A*, pages 769–789, 1997.

[19] M. Lukac. *Quantum Logic Synthesis and Inductive Machine Learning*. PhD thesis, Portland State University, 2009.

[20] M. Lukac and G. Krylov. Study of gpu acceleration in genetic algorithms for quantum circuit synthesis. In *2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 213–218, 2017.

[21] M. Lukac, M. Perkowski, H. Goi, M. Pivtoraiko, C. H. Yu, K. Chung, H. Jee, B-G. Kim, and Y-D. Kim. Evolutionary approach to quantum and reversible circuits synthesis. In *Artificial Intelligence in Logic Design*, pages 201 – 257. Kluwer Academic Publisher, 2004.

[22] M. Lukac, M. Perkowski, and M. Kameyama. Quantum finite state machines - a circuit based approach. *International Journal of Unconvetional Computing*, 9(3-4):267–301, 2013.

[23] Martin Lukac, Marek Perkowski, and Michitaka Kameyama. Evolutionary quantum logic synthesis of boolean reversible logic circuits embedded in ternary quantum space using heuristics, 2011.

[24] A. Malossini, E. Blanzieri, and T. Calarco. Quantum genetic optimization. *IEEE Transactions on Evolutionary Computation*, 12(2):231–241, April 2008.

[25] S. Marsili Libelli and P. Alba. Adaptive mutation in genetic algorithms. *Soft Computing*, 4(2):76–80, Jul 2000.

[26] Rodney Van Meter, Kohei M. Itoh, and Thaddeus D. Ladd. Architecture-dependent execution time of shor's algorithm. 2005.

[27] A. M. Mohammed, N. A. Elhefnawy, M. M. El-Sherbiny, and M. M. Hadhoud. Quantum crossover based quantum genetic algorithm for solving non-linear programming. In *2012 8th International Conference on Informatics and Systems (INFOS)*, pages BIO–145–BIO–153, May 2012.

[28] G.E. Moore. Cramming more components onto integrated circuits. In *Electronics, April 19*, 1965.

[29] Mark Moore and Ajit Narayanan. Quantum-inspired computing. 12 1995.

[30] A. Narayanan and M. Moore. Quantum-inspired genetic algorithms. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 61–66, May 1996.

[31] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[32] Jeremy L. O'Brien. Optical quantum computing. *Science*, 318(5856):1567–1570, 2007.

[33] A. Peres. Reversible logic and quantum computers. *Phys. Rev. A*, 32(6):3266–3276, 1985.

[34] Trailokya Nath Sasamal, Ashutosh Kumar Singh, and Anand Mohan. Reversible logic circuit synthesis and optimization using adaptive genetic algorithm. *Procedia Computer Science*, 70:407–413, 2015.

[35] Philipp Schindler, Daniel Nigg, Thomas Monz, Julio T. Barreiro, Esteban Martinez, Shannon X. Wang, Stephan Quint, Matthias F. Brandl, Volckmar Nebendahl, Christian F. Roos, Michael Chwalla, Markus Hennrich, and Rainer Blatt. A quantum information processor with trapped ions. 2013.

[36] Mathias Soeken, Martin Roetteler, Nathan Wiebe, and Giovanni De Micheli. Hierarchical reversible logic synthesis using luts. In *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, pages 78:1–78:6, New York, NY, USA, 2017. ACM.

[37] M. Szyprowski and P. Kerntopf. An approach to quantum cost optimization in reversible circuits. In *2011 11th IEEE International Conference on Nanotechnology*, pages 1521–1526, Aug 2011.

[38] Lieven M. K. Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S. Yannoni, Mark H. Sherwood, and Isaac L. Chuang. Experimental realization of shors quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414(6866):883–887, dec 2001.

[39] Nikolay V. Vitanov. Synthesis of arbitrary su(3) transformations of atomic qutrits. *Phys. Rev. A*, 85:032331, Mar 2012.

[40] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: An online resource for reversible functions and reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.

[41] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: An online resource for reversible functions and reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.