

# Evolution of Quantum Algorithms Using Genetic Programming

Dissertation  
zur Erlangung des Grades eines  
Doktors der Naturwissenschaften  
der Universität Dortmund  
am Fachbereich Informatik  
von

**André Leier**

Dortmund

2004



# Contents

<b>1</b>	<b>Introduction and Motivation</b>	<b>1</b>
<b>2</b>	<b>Basic Concepts of Quantum Computing</b>	<b>5</b>
2.1	Short History of Quantum Computing . . . . .	7
2.2	The State Space of Quantum Mechanical Systems . . . . .	8
2.3	Quantum Information . . . . .	9
2.3.1	A Single Qubit . . . . .	9
2.3.2	Multiple Qubits . . . . .	10
2.4	Quantum Gates . . . . .	12
2.4.1	Single Qubit Gates . . . . .	13
2.4.2	Controlled Operations . . . . .	16
2.4.3	Sets of Universal Quantum Gates . . . . .	18
2.4.4	Decomposition of unitary transformations . . . . .	20
2.4.5	Oracle gates . . . . .	22
2.5	Projective Measurements . . . . .	23
2.6	Quantum Circuits . . . . .	25
2.6.1	Intermediate Measurements . . . . .	27
2.6.2	Circuit Complexity Measures . . . . .	28
2.7	Quantum Computational Complexity . . . . .	29
2.8	Basic Programming Techniques and Simple Quantum Algorithms . . . . .	31
2.8.1	The Deutsch-Jozsa Problem . . . . .	31
2.8.2	Quantum teleportation . . . . .	35
<b>3</b>	<b>Genetic Programming Fundamentals</b>	<b>39</b>
3.1	GP's Scientific Roots . . . . .	39
3.2	Fundamental Program Structures . . . . .	42
3.2.1	Tree GP . . . . .	42
3.2.2	Linear and Linear-Tree GP . . . . .	44
3.2.3	Graph and Linear-Graph GP . . . . .	46
3.3	Genetic Operators . . . . .	47
3.3.1	Mutation . . . . .	47
3.3.2	Recombination . . . . .	47

3.4	Fitness and Selection . . . . .	49
3.4.1	Fitness Functions . . . . .	49
3.4.2	Selection Algorithms . . . . .	49
3.5	Basic GP Algorithms: Generational vs. Steady-State . . . . .	50
3.6	Introns and Neutrality . . . . .	51
<b>4</b>	<b>Quantum Algorithms and their Classification</b>	<b>53</b>
4.1	The Quantum Fourier Transform and Algorithms Based on It . . . . .	55
4.1.1	From DFT to QFT . . . . .	55
4.1.2	Phase Estimation . . . . .	56
4.1.3	Order-Finding and Other Applications . . . . .	58
4.1.4	Fourier Transform on Arbitrary Groups . . . . .	60
4.1.5	The Hidden Subgroup Problem . . . . .	61
4.1.6	A coarse outline on QFT-based algorithms . . . . .	62
4.2	Quantum Search Algorithms . . . . .	64
4.2.1	Grover's Algorithm . . . . .	64
4.2.2	Quantum Counting: Combining Grover Operator and Phase Estimation . . . . .	68
4.2.3	Applications of Grover's Algorithm . . . . .	69
4.2.4	Quantum Search and NP Problems . . . . .	70
4.2.5	Hogg's Algorithm . . . . .	71
4.3	Quantum Simulation . . . . .	74
4.4	Speedup Limits for Quantum Algorithms . . . . .	76
<b>5</b>	<b>Evolution of Quantum Algorithms</b>	<b>79</b>
5.1	Quantum Circuit Simulation . . . . .	80
5.1.1	Representation of Quantum Circuits . . . . .	81
5.1.2	Matrix-Vector vs. Matrix-Matrix Multiplications . . . . .	82
5.1.3	Implementation and Time Measurements . . . . .	87
5.1.4	Circuit Reduction . . . . .	89
5.1.5	Intermediate Measurements . . . . .	92
5.2	Searching the Space of Quantum Circuits . . . . .	92
5.2.1	Suitable Problems for GP-based Quantum Circuit Design . . . . .	94
5.2.2	Time Consumption for Individual-Evaluations in GP-Search . . . . .	96
5.3	Previous Work . . . . .	99
5.3.1	Automated Circuit Design by Williams & Gray . . . . .	99
5.3.2	Quantum Circuit Evolution by Spector et al. . . . .	100
5.3.3	Rubinstein's GP Scheme . . . . .	106
5.3.4	GAs for Quantum Circuit Design according to Yabuki & Iba . . . . .	108
5.4	Implemented GP Systems for Quantum Circuit Evolution . . . . .	110
5.4.1	Genome Structure - linear vs. linear-tree GP . . . . .	110

5.4.2	Genetic Operators . . . . .	111
5.4.3	Fitness Cases and Fitness Functions . . . . .	113
<b>6</b>	<b>Results and Analyses</b>	<b>121</b>
6.1	Evolution and Scalability . . . . .	122
6.1.1	Evolving Quantum Circuits for 1-SAT . . . . .	123
6.1.2	Evolving Quantum Circuits for Deutsch-Jozsa . . . . .	126
6.1.3	Pre-Evolved Initial Populations . . . . .	130
6.1.4	Disruptive Crossover . . . . .	135
6.2	Search Space Analysis . . . . .	136
6.2.1	Fitness Landscapes and Analysis Methods . . . . .	138
6.2.2	Landscape Analysis . . . . .	139
6.3	Comparison of Selection Strategies . . . . .	150
6.3.1	Selection Strategies . . . . .	150
6.3.2	Experiments and Empirical Results . . . . .	151
<b>7</b>	<b>Discussion and Outlook</b>	<b>165</b>
	<b>About the Author</b>	<b>169</b>
	<b>Bibliography</b>	<b>171</b>
	<b>Index</b>	<b>185</b>



# Acknowledgments

First I would like to express my gratitude to Prof. Dr. Wolfgang Banzhaf for his trusting cooperation, support, and guidance on the way towards this thesis.

Furthermore, I thank the members of the Chair of Systems Analysis at the Dept. of Computer Science at the University of Dortmund for creating a pleasant working atmosphere. Especially, I am greatly indebted to Dipl.-Inform. Michael Emmerich, Dipl.-Inform. Christoph Richter and Dipl. Phys. Ralf Stadelhofer for numerous discussions and helpful comments on quantum computing, and for proofreading various versions of this document.

Parts of this thesis were completed while I was a visiting student at the Memorial University Dept. of Computer Science. I am very grateful to its members for their hospitality and some prolific discussions about the topic. Special thanks go to Dwight Kuo for proofreading most parts of this thesis and never tiring of improving my English.

This work was supported by a grant from the Deutsche Forschungsgemeinschaft (DFG) within the Ph.D. program GK 726 “Materials and Concepts for Quantum Information Processing”.

In this context I would also like to thank the supervisors, scholarship holders and all other participants in the GK 726, ensuring that I got deep insights in one of the most exciting fields of research: quantum computing.





# 1 Introduction and Motivation

*The full range of physically computable functions is now within the scope of GP, which is beginning to find interesting new programs that humans had not previously discovered.*

---

Lee Spector [137]

In the last decade *non-standard computing*<sup>1</sup> concepts have received more and more attention since they raise the hope of computational power far beyond that of conventional computers. These innovative computational architectures were based on the interdisciplinary combination of biology, chemistry or physics on the one hand with computer science on the other. The trend towards alternative models of computation results in several new research fields including DNA or molecular computing, computational chemistry, optical computing, and last but not least quantum computing [58].

The pursuit of new, unexploited computational power inevitably leads to the physical limits of computing. Assuming that the miniaturization of microelectronics will approximately continue as predicted by Moore's law<sup>2</sup>, this shrinkage will reach atomic dimensions within the next 10 to 15 years, or even earlier [78, 130]. As a consequence, physical limits will prevent the development of smaller conventional computer architectures. Thus, quantum effects must be taken into consideration with subsequent adaptation of computational concepts.

Roughly speaking, quantum computation is computation based on the principles of quantum mechanics. That is, quantum computers use coherent atomic-scale dynamics to store and process information. As impressively demonstrated by Peter Shor's polynomial time quantum algorithm for integer factorization [132] (a problem of unknown classical complexity, classically solvable only in exponential time up to now) certain computational problems can be solved on a quantum computer with lower complexity than possible on

---

<sup>1</sup>In this context, the notion "non-standard" relates to computation concepts which differ from the classical von-Neumann concept.

<sup>2</sup>Moore's law, first identified by Gordon Moore in 1965 but reformulated several times in the last decades is nowadays associated with the prediction that computing power is doubled every 18 months at fixed cost.

classical computers.

Yet, quantum computing would not have any practical relevance if quantum computers were merely abstract machines. Different physical systems are tested and used as fundamental quantum computing models, such as optical photons, cavity quantum electrodynamical systems, ion traps, and nuclear spin systems. However, all of these approaches run into fundamental obstacles making a scalable, general-purpose quantum computer seem attainable only in the distant future, if at all. The best experimental realization so far is a seven-qubit NMR (nuclear magnetic resonance) quantum computer developed by a group of scientists from the IBM Almaden Research Center and the Solid State and Photonics Laboratory of Stanford University. They implemented Shor's factoring algorithm and proved that the quantum computer is able to compute the prime factors of a small integer [154].

In view of the demonstrable potential of quantum computing, finding further "killer" applications and the design of new quantum algorithms is desirable. Of course, the ultimate ambition is to develop quantum algorithms which are better than any classical algorithm, since for all other problems conventional computers will be the first choice. Unfortunately, the development of quantum algorithms is extremely difficult since they are highly non-intuitive and their simulation on conventional computers is very computationally intensive. Williams and Clearwater remarked in 1997 [166]:

*"Of course, computer scientists would like to develop a repertoire of quantum algorithms that can, in principle, solve significant computational problems faster than any classical algorithm. Unfortunately, the discovery of Shor's algorithm for factoring large composite integers was not followed by a wave of new quantum algorithms for lots of other problems. To date, there are only about seven quantum algorithms known."*

Currently, the situation is basically the same even though the nature of quantum computing is understood more deeply in general. There is still only a very narrow class of problems known to be sped up by quantum algorithms. Unfortunately, progress is still slow, yet aside from the development of quantum hardware the development of quantum algorithms seems to be crucial for future prospects of quantum computing.

Difficulties in manual quantum circuit design motivate the search for computer-aided or even automatic design techniques. In computer science automatic programming methods (making computers generate program code) were well studied over the last decades. Rooted in the field of machine learning (automatic improvement of algorithms through experience) and inspired by natural evolution, Genetic Programming (GP) emerged in the early 1990s [87]. Originally focused on evolving tree structures, GP is nowadays an umbrella term for all forms of automatic program induction by means of evolution [8]. Due to its large number of applications, e.g., in classical circuit design, data mining, image processing, pattern recognition, and robotics, GP turned out to be a promising approach for automatic generation of computer programs.

---

This thesis deals with the evolution of quantum algorithms using genetic programming. The idea of using GP for quantum circuit design is not novel; rather, it was inspired first by the results of Williams and Gray [167] and Spector et al. [140]. Strictly speaking, they deal with different problems in quantum circuit design: Williams and Gray use GP for *circuit analysis*, that is, the GP system already has knowledge about the quantum circuit (given in the form of a unitary matrix) and tries to find a decomposition of the circuit into elementary gates. Spector et al. use GP for *circuit synthesis*, that is, the GP system evolves quantum circuits only from knowledge about the desired input-output behavior. Using GP for both analysis and synthesis is possible at least for simple problems. In general, because of the missing information about the circuit to be evolved, synthesis has to be considered the much harder problem. In the following, evolving quantum circuits relates to the circuit synthesis, if not stated otherwise.

In the course of this thesis a linear and a linear-tree GP system with an integrated quantum computer simulator were implemented. Their practicality in evolving quantum circuits will be shown in different experiments. These experiments will also reveal that the evolution of quantum circuits is practically feasible only for sufficiently small problem instances. In this context, scalability and the detection of scalability becomes very important. Roughly speaking, scalability means that from quantum circuits solving smaller problem instances quantum circuits solving larger problem instances can be inferred. It is shown that scalable quantum circuits are also evolvable, that is, the general quantum algorithm circuit can easily be inferred from the evolved solutions for small instances of the given problem. Furthermore, investigations of search spaces, fitness landscapes and selection strategies are made, with the aim of improving the efficiency of evolutionary search. Three publications resulted from these examinations [97, 98, 99].

This thesis is organized as follows: After this introduction, Chapter 2 outlines the fundamentals of quantum computing, especially those essential to understand the mathematical principles on which the idealized, decoherence-free simulation of pure-state quantum algorithms depends. However, this chapter cannot compensate for a complete and profound introduction to quantum computing as is provided in [65, 107]. Chapter 3 explains the basics of genetic programming. This comprises the underlying theory of evolution, fundamental program structures in GP, genetic operators, the principle of fitness and selection mechanisms and basic GP algorithms. Furthermore, it briefly discusses the occurrence and possible effects of introns and neutrality. Chapter 4 summarizes most of the quantum algorithms developed thus far and the problems they may be applied to. It deals with possible characteristics of quantum algorithms and explains in detail quantum algorithms based on the quantum Fourier transform and quantum search algorithms. Furthermore, it touches briefly on quantum simulation algorithms and the known limits of quantum computing. The linear and linear-tree GP systems and the quantum computer simulator are explained in Chapter 5 as they are used for quantum circuit evolution. In addition, the difficulties of quantum circuit evolution are addressed and analyzed. Results and analysis concerning the evolution of quantum circuits for the

Deutsch-Jozsa problem and the one-satisfiability problem are explained in Chapter 6. It comprises the evolvability of scalable quantum algorithms, the analysis of search spaces or fitness landscapes respectively, and a comparison of selection strategies for faster evolutionary search. Finally, Chapter 7 summarizes results, draws conclusions and gives an outlook on the common future of quantum computing and genetic programming.

Present research on quantum computing is still dealing with fundamental, theoretical and experimental issues of the computation concept and not yet with programmable desktop quantum machines.

Julio Gea-Banacloche, Editor of Physical Review A, on a plenary debate session on quantum computing, June 2003 [1]:

*“We are not building this so that we can run Microsoft Office on it.”*

## 2 Basic Concepts of Quantum Computing

*All science is either physics or stamp collecting.*

---

E. Rutherford (1871–1937)

*Quantum computation is a qualitatively new way of harnessing nature.*

---

D. Deutsch [45]

*Quantum computation* results from the link between quantum mechanics, computer science and classical information theory [144]. It uses quantum mechanical effects, especially superposition, interference and entanglement, to perform new types of computation which show promise to be more efficient than classical computations. It is the essential trait of the theory of quantum mechanics to make (exclusively) probabilistic predictions, i. e. for a quantum mechanical experiment the theory predicts possible results and their probabilities to occur. This is what makes quantum computing probabilistic<sup>1</sup>.

This chapter describes the fundamental principles of quantum computing. It introduces the quantum circuit model of computation, which provides a “language” to describe quantum algorithms, and explains its basic building blocks: quantum bits, quantum operations (gates) and quantum measurements.

The mathematical framework describing the concepts and principles of quantum mechanics is of course also the theoretical basis of quantum computing. Therefore, it is necessary to deal with the basic postulates of quantum mechanics which connect the physical world with its mathematical model. These postulates directly relate to the modeling of the key elements of quantum computation:

- quantum mechanical systems which inherently contain quantum information,

---

<sup>1</sup>That does not mean that deterministic quantum computations are impossible, but that the nature of quantum computing is based on probabilities.

- compositions of such systems,
- operations on quantum systems for the purpose of information processing, and
- the readout (measurement) of information from quantum systems.

Within this chapter the postulates of quantum mechanics are specified in the subsections dealing with the corresponding topics.

In more detail this chapter is organized as follows: In Section 2.1 background and history of (theoretical) quantum computing are briefly outlined. Most of the information is borrowed from [107, 121]. A good historical review on quantum computing can also be found in the theory component of [146]. Section 2.2 begins the introduction to the mathematical concepts of quantum computing. It establishes the “working space” of quantum computing which is based on ideal, noiseless and therefore only theoretical quantum systems. However, in principle this is sufficient for quantum circuit design, since for practical operations of quantum algorithms, error-correcting mechanisms are already known. Section 2.3 describes the quantum analogous concept of a classical bit, the quantum bit or qubit for short. It can be roughly defined as the simplest (smallest) quantum mechanical system. Furthermore, this chapter deals with multiple qubits and composite quantum systems which enable the full range of quantum computations and allow the effect of entanglement. Section 2.4 shows how to perform quantum operations on quantum bits. It starts with a detailed study of single qubit gates and continues with the description of controlled gates and so-called black-box or oracle gates. This section concludes with a discussion of universal quantum gate sets. Section 2.5 describes measurements and their effects on quantum mechanical systems and Section 2.6 explains the most important differences between classical and quantum circuits. Classically, there are two important models for computation, the Turing machine model and the circuit model. Both models have quantum analogies which are mathematically equivalent. Since the quantum circuit model of computation is simple and more intuitive, it is used throughout this thesis. Like a classical circuit a quantum circuit consists of wires and gates, which have in quantum circuits (only) roughly the same meaning as in classical circuits. Temporarily, this intuitive understanding of the term “quantum circuit” is still sufficient. A rather brief introduction into quantum computational complexity is presented in Section 2.7. Two essential quantum complexity classes are explained and their relations to classical complexity classes described. Finally, Section 2.8 discusses elementary quantum programming techniques and demonstrates their use exemplarily for the Deutsch-Jozsa problem and quantum teleportation.

Basic knowledge of linear algebra and the tensor product is assumed, but the necessary mathematics can also be found in [107]. The postulates of quantum mechanics can also be found, too. To get a deeper insight into quantum computing and quantum algorithms the following broad-ranging references might be of interest: [65, 71, 107, 121]. They also form the basis of this chapter.

## 2.1 Short History of Quantum Computing

The theory of quantum mechanics was established in the mid-1920s. Main contributions were made by M. Born, P. Dirac, W. Heisenberg, E. Schrödinger and others. With quantum mechanics it was possible to explain unknown phenomena raised from various experiments and to resolve inconsistencies in the theories of physics, now designated as classical physics (classical mechanics and classical electrodynamics).

Information can be regarded as not abstract no matter whether it is in someone's mind, written in a book or stored on a magnetic layer of a hard disc – in the words of Rolf Landauer: “Information is physical” [93]. It is physics, which sets the main limitations to process and to manipulate information. Since the early beginnings of analog and digital computers classical physics provided the laws for computing devices. The idea of using the laws of quantum physics for information processing did not emerge until the early 1980s. Important influences on the development of the new computation concept are ascribed to Bell (1964), who demonstrated non-local correlations between different parts of a quantum system, as well as Landauer and Bennett, who both dealt with the connection between energy consumption and *irreversibility*<sup>2</sup> of computation. In 1961 Landauer showed that erasure of information, which is peculiar to irreversible operations, requires the dissipation of energy (*Landauer's principle*). Based on Landauer's work Charles Bennett [13] proved in 1973 that all computation can be performed in principle in a logically reversible manner and therefore does not require dissipation. This result led in 1980 to Paul Benioff's discovery that quantum systems could perform computations in a coherent manner and to his model of a *Quantum Turing Machine* (QTM) [12]. With this proposal the field of quantum computation was born.

In 1982 Richard Feynman pointed to the difficulties of classical computers to efficiently simulate quantum physical systems and suggested using computers based on quantum mechanical principles to handle these difficulties. Benioff's QTM was further developed by Deutsch [43], who also invented the *quantum circuit model of computation* [44]. It can be shown that both models are (nearly) equivalent [169, 108]. Furthermore, Deutsch formulated an *oracle* problem (cf. Section 2.4.5), today known as *Deutsch's problem*, for which he demonstrated the first (randomized) quantum algorithm that performs better than any comparable classical algorithm. The *Deutsch-Jozsa problem* [46], a generalization of Deutsch's problem, was the first one that was found to need only linear time on a quantum computer but exponential time on a deterministic Turing machine (although it needs only polynomial time on a probabilistic Turing machine) [65].

A major breakthrough in quantum computing happened in 1994. First, Simon [134] proposed a quantum algorithm solving an oracle problem in polynomial time on a quantum computer but exponential time on a classical, even probabilistic computer. Simon's

---

<sup>2</sup>A logical gate or function is *reversible*, if the input is uniquely determined by the output, i.e. an inverse function exists mapping the output to its unequivocal input. Otherwise it is *irreversible*.

work was based on a quantum algorithm introduced by Bernstein and Vazirani [16]. Inspired by Simon's results Shor published his polynomial time quantum algorithms for integer factorization and discrete logarithm [132]. These quantum algorithms were the first to solve problems of great practical relevance. Both problems are considered to be hard on classical computers. This difficulty is the basis of many modern public-key cryptography systems such as RSA. Another quantum algorithm attracted attention in 1996 when Grover presented a quadratic speed-up quantum search algorithm [61]. In the period following, newly discovered quantum algorithms were mainly based on the work of Shor and Grover. It turned out that both computational approaches could be applied to classes of similar problems. Unfortunately, no other conceptually new quantum algorithms were presented which had such a deep and pioneering impact like Shor's. A summary of most quantum algorithms is given in Chapter 4.

## 2.2 The State Space of Quantum Mechanical Systems

The model of a quantum computer used here is based on a *closed* or *isolated* quantum mechanical system. This is an ideal system without perturbations and noisy interactions with its surrounding, which are referred to as *decoherence*. Systems in the real world are never absolutely closed. There is always a coupling with the environmental system resulting in a decay of information in the quantum computing device. However, decoherence can be corrected in principle by using *error correcting codes* which also protect against defective quantum operations. Both kinds of *quantum errors*, imperfect operations and quantum noise, are left out of account here. For a detailed introduction on quantum error-correction and fault-tolerant quantum computing see Chapter 7 of [120].

The mathematical formalism of quantum mechanics models a closed quantum mechanical system as follows:

**Postulate 1.** Associated with any closed quantum mechanical system is a *Hilbert space*  $\mathcal{H}$  which is a complete (complex) inner-product space.<sup>3</sup> This vector space is also known as the *state space* of the system. Its unit norm vectors are called (*pure*) *states*.

In quantum computation the state space is limited to finite dimensions. Each state can be regarded as a complete description of the physical system.

**Notation (Bra/Ket).** The standard quantum mechanical notation for (column) vectors in a Hilbert space  $\mathcal{H}$  is  $|\psi\rangle$ . Here,  $\psi$  is just a label of the vector. The notation  $\langle\psi|$  is used for the vector *dual* to  $|\psi\rangle$ . This is a row vector, which corresponds to the complex conjugated and transposed column vector  $|\psi\rangle$ . A column vector  $|\psi\rangle$  is sometimes referred

---

<sup>3</sup>A vector space is complete if every Cauchy sequence in the space converges, concerning a given norm, to an element in the space. In Hilbert spaces the norm is induced by the inner product. Complex inner product spaces are also called *unitary* vector spaces.



to as a *ket*, its dual vector  $\langle\psi|$  is referred to as a *bra*. This notation, also called bra-/ket-notation, was invented by Paul Dirac [47]. The inner product of two vectors  $|\psi\rangle$  and  $|\rho\rangle$  is defined by  $\langle\psi|(|\rho\rangle) \equiv \langle\psi|\rho\rangle$ .

An alternative formalism, especially necessary to deal with open and composite quantum systems, uses the *density operator* or *density matrix* notion respectively and the concept of *mixed states*. Both approaches are mathematically equivalent and lead to the same results. The postulates of quantum mechanics can be formulated using both formalisms. As pointed out by Gruska [65] (in Theorem 2.3.46), “the model of quantum circuits with mixed states is polynomially equivalent, in computational power, to the standard model of quantum circuits over pure states.”

Note, within this thesis that a quantum computer is regarded just as an abstract, mathematical object without reference to a specific implementation. Its physical realization is irrelevant; the same applies to possible sources of error.

## 2.3 Quantum Information

The simplest possible two-level quantum system and therefore the basic information unit in quantum computing is the *quantum bit*, or *qubit* for short.

### 2.3.1 A Single Qubit

Like its classical counterpart a qubit has two basic states denoted  $|0\rangle$  and  $|1\rangle$  by analogy with the two values 0 and 1 of a classical bit. But unlike the classical bit a qubit can also be in a *superposition* of its two basic states. Only after the qubit is read out it is with a certain probability in one or the other basic state.

According to the state vector formalism (Postulate 1), a qubit is a unit vector in a two-dimensional complex vector space  $\mathcal{H} = \mathbb{C}^2$  with inner product. The states or vectors respectively,  $|0\rangle$  and  $|1\rangle$ , also known as the *computational basis states*, form an orthonormal basis (ON-basis) of this space. Usually  $|0\rangle$  and  $|1\rangle$  are identified with the standard basis vectors in  $\mathbb{C}^2$ ,  $(1, 0)$  and  $(0, 1)$ . A qubit in  $\mathcal{H}$  can be any arbitrary state formed by *linear combination* of  $|0\rangle$  and  $|1\rangle$ :

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle, \quad (2.1)$$

with  $\alpha_0, \alpha_1 \in \mathbb{C}$  and  $|\alpha_0|^2 + |\alpha_1|^2 = 1$ . Here, with  $\alpha_0, \alpha_1 \neq 0$  the qubit labeled  $\psi$  is in a *superposition state*.

The normalization condition relates to equivalence classes of vectors that differ only by a nonzero complex factor. They always describe the same physical state and it is therefore useful to choose unit vectors as representatives of the states. Moreover, the additional condition  $|\alpha_0|^2 + |\alpha_1|^2 = 1$  relates to the readout or *measurement* of qubits: Classical bits have to be read to determine their values or states 0 or 1 — the same

applies to qubits. However, in quantum computing the outcome of a (single qubit) measurement, ‘0’ or ‘1’, is not deterministic but probabilistic. Measurement of qubit  $|\phi\rangle$  gives either the result ‘0’ with probability  $|\alpha_0|^2$  or the result ‘1’ with probability  $|\alpha_1|^2$ . From the normalization condition of probability measures it follows  $|\alpha_0|^2 + |\alpha_1|^2 = 1$ . By measurement any superposition state collapses to the computational basis state  $|k\rangle$  according to the measurement result ‘ $k$ ’. But it also means that, although a qubit is very different from a classical bit, it is not possible to gain more information from a qubit than from a classical bit.<sup>4</sup> Especially, the values of the amplitudes  $\alpha_0, \alpha_1$  are not accessible by measurement. Measurements are discussed in detail in Section 2.5.

A geometrical representation of the state of a single qubit is provided by the *Bloch sphere*. It is often used to illustrate the effect of single qubit operations, which are elementary in quantum computing. Unfortunately, there is no equivalent representation for multiple qubits. The Bloch sphere representation of a qubit reads as follows:

$$|\psi\rangle = e^{i\gamma} \left( \cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle \right).$$

It is obtained from Equation 2.1 by rewriting the complex numbers  $\alpha_0, \alpha_1$  in polar coordinates,  $\alpha_0 = r e^{i\gamma}$  and  $\alpha_1 = s e^{i\delta}$  with  $r, s, \gamma, \delta \in \mathbb{R}$  and  $r, s \geq 0$ , and a suitable choice of the parameters  $\varphi = \delta - \gamma$  and  $\theta = 2 \arccos r$ . It is a property of measurement that *global phase factors* like  $e^{i\gamma}$  can be ignored. Then, a single qubit state  $|\psi\rangle$  can be visualized as a point  $(\cos \varphi \sin \theta, \sin \varphi \sin \theta, \cos \theta)$  on the unit sphere in  $\mathbb{R}^3$  as it is illustrated in Figure 2.1. The  $x$ -,  $y$ - and  $z$ -axis are defined by the states  $1/\sqrt{2}(|0\rangle + |1\rangle)$ ,  $1/\sqrt{2}(|0\rangle + i|1\rangle)$  and  $|0\rangle$ .

### 2.3.2 Multiple Qubits

A *quantum register* is a quantum mechanical system composed of several quantum bits. Considering a system of  $n$  qubits, its state space is the  $2^n$ -dimensional Hilbert space  $\mathcal{H}^{(n)} := \mathbb{C}^{2^n}$ . Similar to the 1-qubit case, the computational basis states of  $\mathcal{H}^{(n)}$ , labeled

$$|k\rangle \equiv |k_{n-1} \dots k_0\rangle, \text{ with } k_i \in \{0, 1\},$$

compare to the  $2^n$  possible states of a classical  $n$ -bit register. Here,  $k_{n-1} \dots k_0$  is the binary representation of  $k$ , where  $k_i$  is associated with the  $i$ -th qubit. In the case where the system is in a computational basis state each qubit has a definite value, either  $|0\rangle$  or  $|1\rangle$ . Note that within this thesis the qubits are counted starting with 0 from the rightmost position in the ket vector (the least significant qubit), as is usual in computer science.

---

<sup>4</sup>*Superdense coding* allows to communicate two classical bits by transmitting a single qubit of a pair of entangled qubits [107]. At first sight, this might contradict the above statement. However, one needs two qubits to perform superdense coding and both qubits must be measured (in the Bell basis).

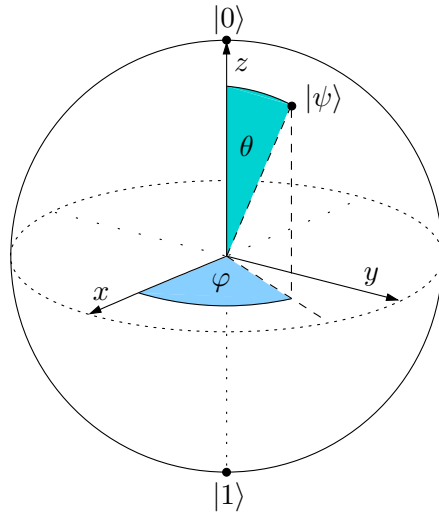


Figure 2.1: Bloch sphere.

Any linear combination or superposition of the basis vectors is an allowed state of the system (*superposition principle*). Thus, the general (superposition) state of an  $n$ -qubit register can be written as

$$|\psi\rangle = \sum_{k=0}^{2^n-1} \alpha_k |k\rangle, \quad \alpha_k \in \mathbb{C}, \quad 0 \leq k \leq 2^n - 1.$$

Because of the normalization condition of state vectors it is  $\sum_{k=0}^{2^n-1} |\alpha_k|^2 = 1$ . The probability for the quantum register being in state  $|k\rangle$  (measurement result ‘ $k$ ’) is  $|\alpha_k|^2$ . By convention the  $2^n$  basis states are identified with the standard basis vectors:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \dots, \quad |2^n - 1\rangle = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

Mathematically, the extension from one to many qubits or the union of two or more quantum registers to a larger register is made by means of the tensor product  $\otimes$ :

**Postulate 2.** The state space of a composite system is the tensor product of the state spaces of the component systems. Let  $|\psi_A\rangle$  be the state of system  $A$  with state space  $\mathcal{H}_A$  and  $|\psi_B\rangle$  the state of system  $B$  with state space  $\mathcal{H}_B$ . Then, the Hilbert space of the bipartite system  $AB$  is  $\mathcal{H}_{AB} = \mathcal{H}_A \otimes \mathcal{H}_B$  and the joint state of the total system is  $|\psi_A\rangle \otimes |\psi_B\rangle$ .

Moreover, if  $\{|\nu\rangle_A\}$  is an ON-basis for  $\mathcal{H}_A$  and  $\{|\mu\rangle_B\}$  is an ON-basis for  $\mathcal{H}_B$ , then  $\{|\nu\rangle_A \otimes |\mu\rangle_B\}$  is an ON-basis for  $\mathcal{H}_{AB}$ . In accordance with the tensor product of vector spaces, the dimension of  $\mathcal{H}_{AB}$  is the product of the dimensions of  $\mathcal{H}_A$  and  $\mathcal{H}_B$ . Therefore, the state space of a quantum register increases exponentially with the number of qubits. Abbreviated notions for the tensor product  $|\psi\rangle \otimes |\phi\rangle$  of two arbitrary states  $|\psi\rangle$  and  $|\phi\rangle$  are  $|\psi\rangle|\phi\rangle \equiv |\psi, \phi\rangle \equiv |\psi\phi\rangle$ .

As an example, consider a system of two qubits  $A$  and  $B$ . The computational basis states  $(|00\rangle, |01\rangle, |10\rangle, |11\rangle)$  for system  $AB$  result from the basis states of  $A$  and  $B$   $(|0\rangle, |1\rangle)$  by tensor multiplication:  $|x_1x_0\rangle = |x_1\rangle \otimes |x_0\rangle, \forall (x_0, x_1) \in \{0, 1\}^2$ .

In multiple qubit systems there are states which cannot be expressed as a tensor product of states of its single qubit components. This property is referred to as *entanglement* or *nonseparability*. Let  $|\psi_{AB}\rangle$  be a bipartite state. If there are any two states  $|\phi_A\rangle$  in  $\mathcal{H}_A$  and  $|\phi_B\rangle$  in  $\mathcal{H}_B$ , such that  $|\psi_{AB}\rangle = |\phi_A\rangle \otimes |\phi_B\rangle$ , the state is called *separable* (or unentangled); otherwise it is *entangled* (or unseparable). The following examples for entangled states of a two qubit system are known as the *Bell* or *EPR states* (due to Einstein, Podolsky and Rosen):

$$\begin{aligned} |\phi^+\rangle &:= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), & |\phi^-\rangle &:= \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle), \\ |\psi^+\rangle &:= \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle), & |\psi^-\rangle &:= \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle). \end{aligned}$$

When performing a measurement on a subsystem of a composite system with entangled state, another way of thinking about entanglement becomes noticeable. How entanglement is characterized by measurement is described in detail in Section 2.5.

## 2.4 Quantum Gates

Roughly speaking, quantum computation means just transforming a state of a given quantum system into another state, usually followed by a measurement. Physicists call this *state transformation* a (time) *evolution* of the quantum system, which can be mathematically represented by a unitary operator<sup>5</sup>:

**Postulate 3.** The evolution of a *closed* physical system in a time interval  $[t_0, t_1], t_0 < t_1$ , is described by a unitary operator  $U(t_0, t_1)$  which depends only on  $t_0$  and  $t_1$ . Let  $|\psi_t\rangle$  denote the state of the system at time  $t$ , then it is

$$|\psi_{t_1}\rangle = U(t_0, t_1)|\psi_{t_0}\rangle.$$

---

<sup>5</sup>The matrix representation  $U$  of a linear operator is unitary (and therefore the operator itself), if  $U^\dagger U = I$ , where  $U^\dagger = (U^T)^*$  is the complex conjugate transpose of the  $U$  matrix and  $I$  is the identity operator.

A unitary transformation on  $n$  qubits, and thus a vector in  $\mathcal{H}^{(n)}$ , is a unitary  $2^n \times 2^n$  matrix. The set of all unitary matrices of same size is a group in the algebraic sense with the matrix multiplication as group operation. In particular, it follows:

**Theorem 1.** If  $U$  and  $V$  are two unitary matrices (of suitable dimension), then  $UV$  is unitary as well.

Note that there are infinitely many unitary matrices of a fixed size.

Geometrically, unitary transformations preserve inner products between vectors and with it the length of vectors and the angles between vectors. They are imaginable as rotations of the vector space. Moreover, unitary operators are bijective and therefore reversible.

Following the nomenclature of electrical circuits which consist of wires and logic gates, unitary transformations are called *quantum gates*. This analogy is further discussed in Section 2.6. Like many classical gates the most important quantum gates have a certain graphical representation. A general quantum gate  $U$  operating on  $n$  qubits is illustrated in Figure 2.2(a). The short wires correspond to the incoming (left) and outgoing qubits (right). Usually, the bottom-most wire corresponds to the least significant qubit (qubit 0). Suppose  $U$  is a product of unitary transformations  $U_1$  and  $U_2$ ,  $U = U_2U_1$ , then an equivalent schematic symbol notation, or quantum circuit respectively, is depicted in Figure 2.2(b). Note the different order of  $U_1$  and  $U_2$  in the product notation and in the graphical representation.

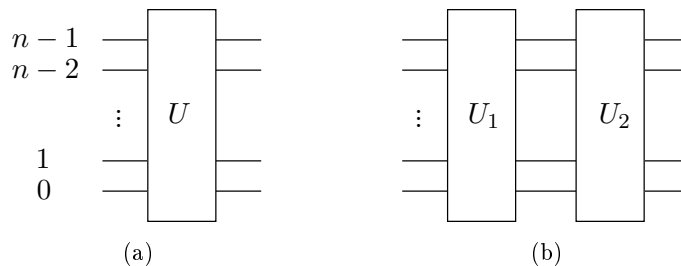


Figure 2.2: a) A general  $n$ -qubit gate, denoted  $U$ . The qubits are counted from bottom (0) to top ( $n-1$ ). b) Quantum circuit implementing  $U = U_2U_1$  by means of  $U_1$  and  $U_2$ . Time and control flow in the quantum circuit model goes from left to right.

### 2.4.1 Single Qubit Gates

A single qubit gate is identified by a unitary  $2 \times 2$  matrix. Some important quantum gates are defined by the so-called *Pauli matrices*, denoted  $X$ ,  $Y$  and  $Z$ :

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

Their graphical representation, including the action on an arbitrary single qubit state, is shown in Figure 2.3. The  $X$ -gate is equivalent to quantum  $NOT$ . Alternative symbols are a box labeled with  $NOT$  and the  $\oplus$  symbol.

$$\begin{array}{ccc}
 \alpha|0\rangle + \beta|1\rangle & \text{---} \boxed{X} \text{---} & \beta|0\rangle + \alpha|1\rangle \\
 \alpha|0\rangle + \beta|1\rangle & \text{---} \boxed{Y} \text{---} & -i\beta|0\rangle + i\alpha|1\rangle \\
 \alpha|0\rangle + \beta|1\rangle & \text{---} \boxed{Z} \text{---} & \alpha|0\rangle - \beta|1\rangle
 \end{array}$$

Figure 2.3: The Pauli matrices.

By exponentiation of the Pauli matrices further unitary matrices emerge: calculating the matrix exponentials  $e^{-i\psi U}$ , for  $U \in \{X, Y, Z\}$  and  $0 \leq \psi < 2\pi$  results in rotations about the  $x$ -,  $y$ - and  $z$ -axis of the Bloch sphere. From the resulting matrices the following rotation operators  $R_x$ ,  $R_y$  and  $R_z$  can be easily derived (using  $\cos(\psi) = \cos(-\psi)$  and  $\sin(-\psi) = -\sin(\psi)$ ):

$$\begin{aligned}
 R_x(\phi) &= \begin{pmatrix} \cos \phi & i \sin \phi \\ i \sin \phi & \cos \phi \end{pmatrix}, \\
 R_y(\phi) &= \begin{pmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{pmatrix}, \\
 R_z(\phi) &= \begin{pmatrix} e^{-i\phi} & 0 \\ 0 & e^{i\phi} \end{pmatrix}.
 \end{aligned}$$

According to this definition a gate  $R_u[\phi]$ ,  $u \in \{x, y, z\}$ , is a rotation by  $2\phi$  about the  $u$ -axis. In literature,  $R_x$ ,  $R_y$  and  $R_z$  are usually defined in a way that they perform rotations by  $\phi$  (by inserting a factor of  $1/2$ ).

The corresponding schematic gate symbols are shown in Figure 2.4.

$$\text{---} \boxed{R_x[\phi]} \text{---} \quad \text{---} \boxed{R_y[\phi]} \text{---} \quad \text{---} \boxed{R_z[\phi]} \text{---}$$

Figure 2.4: Symbols of the single-qubit gates  $R_x$ ,  $R_y$  and  $R_z$ .

An arbitrary unitary operator on a single qubit can be written in different ways as a product of rotation matrices together with an overall phase shift factor. One way is provided by the following theorem [107]:

**Theorem 2.** (*X-Y decomposition of rotations*) Let  $U$  be a unitary single qubit operator. Then  $\alpha, \theta, \phi, \psi \in \mathbb{R}$  exist such that

$$U = e^{i\alpha} R_x(\phi) R_y(\theta) R_x(\psi).$$

In particular, each  $Rz$  operator can be written as a product of  $Rx$  and  $Ry$  rotations. Moreover, there is also an analogous  $Z$ - $Y$  decomposition, where in Theorem 2  $Rx$  is substituted by  $Rz$ . Occasionally, such a general one-qubit unitary operator is denoted  $U_2(\alpha, \theta, \phi, \psi)$ .

$Rz$ -gates can also be designated as *phase gates*. This becomes clear when writing them in the form

$$Rz(\phi) = e^{-i\phi} \begin{pmatrix} 1 & 0 \\ 0 & e^{i2\phi} \end{pmatrix}.$$

Applied to a single qubit state the diagonal coefficient  $e^{i2\phi}$  becomes a relative phase factor regarding the  $|1\rangle$  amplitude of the state. Because of a peculiarity of quantum measurements the global phase factor is unimportant and can be ignored. The factor 2 in the exponent can be eliminated by defining

$$PH(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}.$$

Up to a global phase factor this class of gates is equivalent to  $Rz$ .

Besides this definition, the following gate is sometimes referred to as the phase gate:

$$S = PH(\pi/2) = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}.$$

The square root of gate  $S$  is gate  $T$ , the so-called  $\pi/8$  gate<sup>6</sup>:

$$T = PH(\pi/4) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}.$$

Another important single qubit gate is the *Hadamard* gate ( $H$ ):

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

It maps  $|0\rangle$  to  $1/\sqrt{2}(|0\rangle + |1\rangle)$  and  $|1\rangle$  to  $1/\sqrt{2}(|0\rangle - |1\rangle)$ . Furthermore, it is  $H^2 = I$ .

In order to be applicable to an  $n$ -qubit quantum register with a  $2^n$ -dimensional state vector, quantum gates operating on less than  $n$  qubits have to be adapted to higher dimensions. For example, let

$$U = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

be an arbitrary single qubit gate applied to qubit  $q$  ( $0 \leq q < n$ ) of an  $n$ -qubit register. Then the entire  $n$ -qubit transformation, here denoted with  $\hat{U}$ , can be written as a tensor product in the form:

$$\hat{U} \equiv \underbrace{I \otimes \dots \otimes I}_{n-(q+1)} \otimes U \otimes \underbrace{I \otimes \dots \otimes I}_q.$$

---

<sup>6</sup>The equivalent  $Rz$ -gate has diagonal coefficients  $e^{\pm i\pi/8}$ .

What was intuitively clear is now confirmed: an empty wire in the graphical representation of quantum gates can be identified with the identity matrix. The matrix  $\widehat{U}$  consists of  $2^{n-(q+1)}$  major “blocks” on the diagonal, where each block contains  $2^q$  matrices  $U$  which are diagonally arranged and shifted by one position. The structure of  $\widehat{U}$  with  $n = 4$  and  $q = 1$  is illustrated in Figure 2.5.

Calculating the new quantum state requires  $2^{n-1}$  matrix-vector-multiplications (each block, each submatrix) of the  $2 \times 2$  matrix  $U$  (cf. Section 5.1.2). It is easy to see that the costs of simulating quantum circuits on conventional computers grow exponentially with the number of qubits.

In the same way as  $\widehat{U}$  is a composition of  $I$  gates and  $U$ , other transformations can be built by tensor multiplication from single qubit gates which operate in parallel on different qubits. More generally:

**Theorem 3.** Let  $U$  be a unitary operator on  $\mathcal{H}^{(m)}$  and  $V$  a unitary operator on  $\mathcal{H}^{(n)}$ . Then,  $U \otimes V$  is a unitary operator on  $\mathcal{H}^{(mn)}$ .

For example, applying the Hadamard gate on each qubit of an  $n$  qubit quantum register realizes the unitary transformation

$$\underbrace{H \otimes \cdots \otimes H}_{n \text{ times}} =: H^{\otimes n}.$$

But there are also multiple qubit gates which cannot be decomposed into a tensor product of single qubit transformations, i.e., they are *unseparable*. This, of course, relates to entanglement of quantum states, as entangled states can only be generated by using unseparable gates.

### 2.4.2 Controlled Operations

The controlled-*NOT* gate, also referred to as *CNOT*, operates on two qubits, a *control qubit* and a *target qubit*. The action of the *CNOT* is as follows: it flips the target qubit if the control qubit is set to  $|1\rangle$  and leaves it unchanged otherwise. Suppose two qubits  $|ct\rangle$  are given where the first qubit is the target qubit and the second is the control qubit. Then, the effect of *CNOT* on the computational basis states is given by  $|c\rangle|t\rangle \rightarrow |c\rangle|t \oplus c\rangle$ , with  $c, t \in \{0, 1\}$ . Its matrix representation and gate symbol is shown in Figure 2.6. It is easy to prove that the *CNOT* cannot be decomposed into a tensor product of two single qubit transformations.

In a similar way the ( $C^k$ *NOT*) gate is defined on  $k + 1$  qubits. It flips the target-qubit if the  $k$  control-qubits are 1. For  $k = 2$  this gate is called a *Toffoli* gate or *CCNOT*. It acts on the computational basis states as follows:  $|a, b, c\rangle \rightarrow |a, b, c \oplus ab\rangle$  for  $a, b, c \in \{0, 1\}$ , where  $a$  and  $b$  denote the two control qubits and  $c$  the target qubit. Essentially, by preparing qubit  $c$  to 1 the outgoing target qubit becomes  $\neg(ab)$ . Another



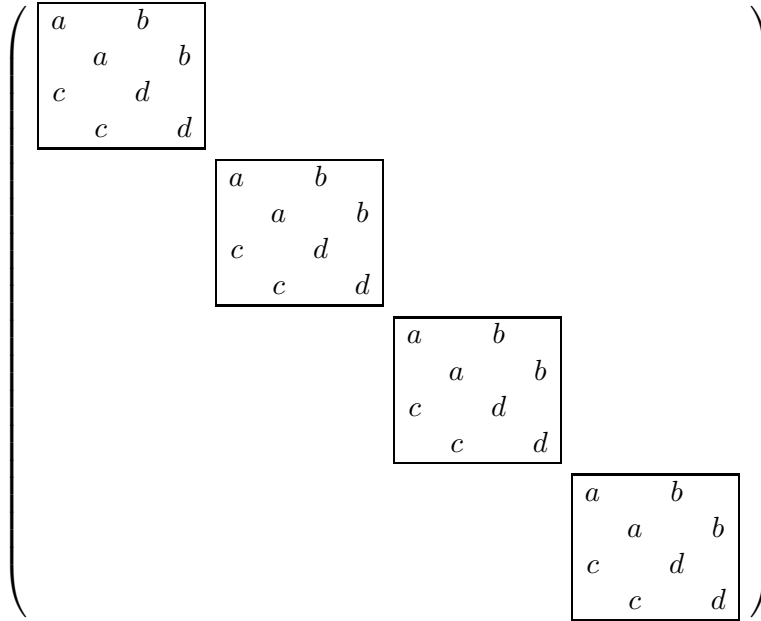


Figure 2.5: With  $n = 4$  and  $q = 1$ ,  $\widehat{U}$  consists of four major blocks on the diagonal, where each block contains the matrix  $U$  twice. Within one block the structure of the  $2 \times 2$  matrices is “broken open” and the matrices overlap each other.

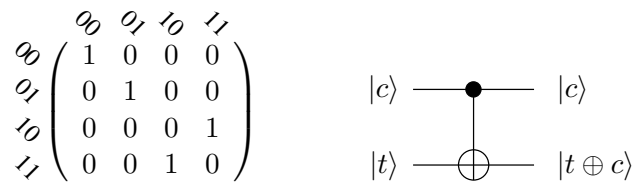


Figure 2.6: The *CNOT* gate operates on two qubits: the solid circle indicates the control qubit and the  $\oplus$  indicates the target qubit. The bit tuples labeling the matrix rows and columns indicate the order of basis states.

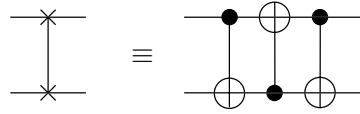


Figure 2.7: The *SWAP* gate and the equivalent quantum circuit using *CNOT* gates. The matrix of *SWAP* is obtained by multiplication of the corresponding *CNOT* matrices.

useful 2-qubit operation is *SWAP* which interchanges the states of the two input qubits:  $|a, b\rangle = |b, a\rangle$ . It can be implemented as a sequence of three *CNOT*s. Its schematic symbol and decomposition in *CNOT* gates is shown in Figure 2.7.

More generally, let  $U$  be an  $m$ -qubit unitary operator. Then, a controlled operation  $C^k(U)$  on  $k+m$  qubits acts on the  $m$  target qubits like the  $U$ -gate, provided all  $k$  control qubits are 1. Otherwise it has no effect. For example, controlled phase gates with control qubit 1 and target qubit 0 are given by

$$CPH(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{2i\phi} \end{pmatrix}.$$

Similar to single-qubit gates, controlled quantum gates have to be adapted to higher dimensions, if required by the Hilbert-space. Regarding a controlled operation  $C^k(U)$  with single-qubit gate  $U$ , the number of matrix-vector-multiplications of  $U$  for calculating the new quantum state is reduced to  $2^{n-1-k}$ . In that case, all diagonal coefficients, assigned to basis states which do not meet the control conditions, are set to 1, as exemplified by Figure 2.8.

### 2.4.3 Sets of Universal Quantum Gates

A gate or a set of gates is defined to be universal for classical computing, if any arbitrary gate or function can be computed requiring only those gates. Since the *NAND* gate is universal in classical computing and has a quantum equivalent provided by the Toffoli or *CCNOT* gate, the set of all quantum circuits comprises all classical circuits. However, this is only a small subset. In contrast to the discrete space of all classical operations the set of quantum operations is continuous. Therefore, the concept of universality for a *discrete* set of quantum gates rests on “good approximations” of arbitrary unitary operations.

In this context it is quite helpful that the sufficiently strong causality principle - similar causes have similar effects - applies also to quantum circuits, that is, small changes or errors in a single unitary operation or a gate sequence cause only small changes in the outcome of the circuit. Specifically, considering a computation where several quantum

$$\begin{pmatrix} \boxed{\begin{matrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{matrix}} & & & \\ & \boxed{\begin{matrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{matrix}} & & & \\ & & \boxed{\begin{matrix} 1 & & & \\ & a & b & \\ & & 1 & \\ & c & & d \end{matrix}} & & & \\ & & & \boxed{\begin{matrix} 1 & & & \\ & a & b & \\ & & 1 & \\ & c & & d \end{matrix}} & & & \end{pmatrix}$$

Figure 2.8: Controlled- $U$  transformation with control qubits  $\{0, 3\}$  and target qubit 1 on a 4-qubit quantum computer. To calculate the new quantum state, only two multiplications of  $U$  with the corresponding subvector of the current state vector are required.

gates with a common bounded error are applied to an initial state  $|\psi\rangle$ , the accumulated error in the resulting state grows *linearly* with the length of the circuit. This motivates the following definition:

Let  $U$  and  $V$  be unitary operators acting on a Hilbert Space  $\mathcal{H}^n$ .  $V$  is called an *approximation of  $U$  with error  $\epsilon$* , if

$$\epsilon = \max_{|\psi\rangle} \|(U - V)|\psi\rangle\|,$$

with  $|\psi\rangle \in \mathcal{H}^n, \|\psi\rangle\| = 1$ .

A set of quantum gates is called *universal*, if any unitary transformation can be approximated to arbitrary accuracy by a quantum circuit consisting of the gates from that set.

The following two theorems [107] comprise the most important results about the universality of quantum gates and the approximation of quantum circuits.

**Theorem 4.** (*Universality of single qubit and CNOT gates*) An arbitrary unitary operation  $U$  on  $n$  qubits ( $U : \mathcal{H}^n \rightarrow \mathcal{H}^n$ ) can be realized *exactly* by a quantum circuit requiring  $O(n^2 2^{2n})$  gates from the set of *CNOT* and single qubit gates.

**Theorem 5.** (*Universality with a discrete set*) The discrete gate set  $\{H, CNOT, T\}$  forms a universal basis for quantum computation. Moreover, let  $U$  be an arbitrary unitary operation which can be realized exactly by a quantum circuit containing  $m$  gates from the set of *CNOT* and single qubit gates. Then, this circuit can be approximated to an accuracy  $\epsilon$  using  $O(m \log^c(m/\epsilon))$  gates from the discrete gate set, where  $c$  is a constant approximately equal to 2. By adding gate  $S$ , the approximations can be done fault-tolerantly.

Unfortunately, most unitary transformations cannot be *efficiently* implemented from a small set of elementary gates, i. e. given a unitary transformation  $U$  on  $n$  qubits, there is no circuit of size polynomial in  $n$  approximating  $U$  [107].

#### 2.4.4 Decomposition of unitary transformations

Theorem 4 is proven by explicitly constructing a decomposition of an arbitrary unitary matrix into single qubit and *CNOT* gates. In the following, this construction is outlined briefly. In this context, an improvement of this construction is mentioned which was introduced by Aho and Svore [5]. Here, only the idea behind their approach is described.

The construction of a decomposition can be done in two steps: first, expressing the general unitary matrix of dimension  $d$  as a product of at most  $d(d - 1)/2$  *two-level unitary operators*, that is, matrices which act only non-trivially on two or fewer vector components, as shown in Figure 2.9, and second, implementing an arbitrary two-level matrix by single qubit and *CNOT* gates.



In their recent paper [5] Aho and Svore present a decomposition algorithm<sup>7</sup> with an improved two-level decomposition phase using a technique, which they call *Palindrome Transformation*. The idea behind this technique is, to find an optimal ordering of two-level operations in the first phase, such that the ordering of the *palindromic subcircuits* of self-inverting gates<sup>8</sup>, resulting from the second phase, leads to a maximal amount of cancellations of the self-inverting gates: A palindromic subcircuit  $A$  is a gate sequence of the form

$$A_1 A_2 \dots A_k V A_k \dots A_2 A_1 .$$

Two successive palindromic subsequences  $A$  and  $B$  can have a subsequence of self-inverting gates in common, like

$$\dots A_{j+1} A_j \dots A_2 A_1 A_1 A_2 \dots A_j B_{j+1} \dots ,$$

with  $B_1 = A_1 \dots B_j = A_j$  which can be reduced to  $\dots A_{j+1} B_{j+1} \dots$

It is shown that the *Palindromic Optimization Algorithm* (POA) achieves a large benefit, resulting in significantly smaller decompositions than those obtained by the conventional method. However, even for small numbers of qubits, the resulting quantum circuits are still large. For more details see [5].

It is unknown, whether there exist more efficient decomposition algorithms. Therefore, other approaches might be necessary to find even shorter decompositions. Such a different approach to find optimal quantum circuits is provided by evolutionary algorithms (cf. Section 5.3.1) [167].

### 2.4.5 Oracle gates

In computer science an *oracle* is a black-box function, that is, a function whose internal working is unknown. An input for the oracle is directly processed into an output. It is said that the oracle responds on the query immediately. This implies that the costs of operating a black-box are irrelevant for complexity analysis.

An oracle gate in quantum computing is usually a “variable” gate. It enables the encoding of problem instances and represents in this way the input of a quantum algorithm. Oracle gates may change from instance to instance of a given problem, while the “surrounding” quantum circuit remains unchanged. Consequently, a proper quantum circuit solving a given problem has to achieve the correct outputs (after measurement) for all oracles representing problem instances.

In certain quantum algorithms, like Grover’s (cf. Chapter 4.2) or Deutsch’s (cf. Section 2.8.1), oracle gates are permutation matrices computing Boolean functions  $f : \{0, 1\}^n \rightarrow$

---

<sup>7</sup>They call the process that generates for an arbitrary unitary matrix an exact decomposition a *quantum circuit compilation*.

<sup>8</sup>Of course, a gate  $A$  is called *self-inverting*, if  $AA = I$ . Examples are the Hadamard gate  $H$  and the *NOT* and *CNOT* gates.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 2.10: Example for an oracle matrix implementing the *OR* function of two inputs. The right-most qubit is flipped, if at least one of the two other qubits is ‘1’.

$\{0, 1\}$ . The transformation can be defined by the map  $U_f : |x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$ , where  $|x\rangle$  is an  $n - \text{qubit}$  state,  $|y\rangle$  is a single qubit state and  $\oplus$  indicates the addition modulo 2. For  $y = 0$  the final state of the single qubit becomes  $|f(x)\rangle$ . The oracle matrix for  $f$  inverts the output qubit, iff  $f$  yields “1” on the input qubits. In this case, the matrix swaps the amplitudes between the states differing only with respect to the output bit. As an example, Figure 2.10 shows a reversible matrix implementing *OR* on two input bits. The symbol representation of an oracle gate is usually just an appropriately labeled box, covering all the qubits the oracle is operating on. In cases where the oracle gate corresponds to a Boolean function the target qubit with the function value may be marked by the  $\oplus$  symbol.

Behind every oracle gate is a particular quantum circuit calculating the output. Such a quantum circuit usually gets additional inputs and needs also additional ancillary qubits which are left out in the symbol representation of the oracle. This can be done, because the additional qubits are not needed for the remainder of the computation and using a technique called *uncomputing*, one can get rid of the “garbage” assigned to the ancillary qubits and put them back to the initial base state. For instance, a quantum circuit for  $U_f$  might have additional input qubits encoding a Boolean function  $f$  which is then calculated at the output. Since any classical circuit can be simulated efficiently (in linear time) by a quantum circuit, usually one does not need to deal with the exact implementation.

## 2.5 Projective Measurements

Quantum information processing is useless without readout or measurement respectively. It is the final step in quantum algorithms, since there is no other way to gain information about the quantum system than by measurement. This section deals only with the *projective* or *von Neumann measurement* in the computational basis, which is a special case

of a more general quantum measurement described by measurement operators. However, all other kinds of measurements proved to be equivalent to unitary transformations, using auxiliary qubits, so-called *ancillae*, if necessary, followed by projective measurements.

Before explaining the effects of projective measurements on quantum systems the concept of *observables* is introduced. An observable  $M$  is a property of a physical system that can be measured. Mathematically  $M$  is a Hermitian (self-adjoint) operator in a Hilbert space with a spectral decomposition  $M = \sum_i \lambda_i |i\rangle\langle i|$ , where  $\lambda_i$  are the eigenvalues of  $M$ . The corresponding eigenstates  $|i\rangle$  of  $M$  form an ON-basis in the vector space. Here,  $P_i := |i\rangle\langle i|$  is the projector onto the eigenstate  $|i\rangle$ .

**Postulate 4.** A projective measurement on a quantum system is described by an observable  $M$ . The possible outcome of a measurement of  $M$  is an eigenvalue  $\lambda_i$  of  $M$ . After the measurement, the quantum state is an eigenstate of  $M$  corresponding to the measured eigenvalue. If the quantum state of the system just before the measurement is  $|\psi\rangle$ , then the probability of getting result  $\lambda_i$  is given by

$$Pr(i) = \|P_i|\psi\rangle\|^2 = \langle\psi|P_i|\psi\rangle.$$

If the outcome is  $\lambda_i$ , the normalized post-measurement state becomes

$$\frac{P_i|\psi\rangle}{\sqrt{Pr(i)}}.$$

A projective measurement *in the computational basis*  $\{|k\rangle\}$  implies the use of projectors  $P_k = |k\rangle\langle k|$  to perform the projective measurement. For measurements in the standard basis the numerical outcomes  $\lambda_i$  are identified with ‘ $i$ ’. In this case, a superposition state prior to the measurement collapses with the measurement to  $|i\rangle$ . Furthermore, it is not important to know the eigenvalues  $\lambda_i$  (and thus  $M$ ), since probabilities  $Pr(i)$  and post-measurement states do not depend on their numerical values. In the following, the term “measurement” always refers to projective measurements in the computational basis.

A *partial* measurement of a single qubit  $q$  in an  $n$ -qubit register with outcome ‘ $i$ ’ is a projection into the subspace, spanned by all computational basis vectors with  $q = i$ . The probability  $Pr_q(i)$  of measuring a single qubit with result ‘ $i$ ’ is the sum of the probabilities for all basis states with  $q = i$  and the post-measurement state is just the superposition of these basis states, re-normalized by the factor  $1/\sqrt{Pr_q(i)}$ . For example, measuring the first (right-most) qubit of  $|\psi\rangle = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle$ , with  $\alpha_0, \alpha_1, \alpha_2, \alpha_3 \in \mathbb{C}$  and  $|\alpha_0|^2 + |\alpha_1|^2 + |\alpha_2|^2 + |\alpha_3|^2 = 1$ , gives ‘1’ with probability  $|\alpha_1|^2 + |\alpha_3|^2$ , leaving the post-measurement state  $|\psi'\rangle = 1/\sqrt{|\alpha_1|^2 + |\alpha_3|^2}(\alpha_1|01\rangle + \alpha_3|11\rangle)$ . The projectors are just  $P_i = I \otimes |i\rangle\langle i|$ . It can be proved that multiple qubit measurements can be treated as a series of single qubit measurements.

Note that quantum measurements are irreversible operators, though it is usual to call these operators measurement *gates*. In this thesis a single qubit measurement is assigned the schematic symbol illustrated in Figure 2.11.



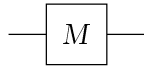


Figure 2.11: Circuit symbol for a single qubit measurement.

The quantum effect of entangled states was already discussed in Section 2.3.2. Measurements provide another equivalent way to define entanglement. A multiple qubit quantum state is not entangled if the measurement of one single qubit has no effect on any other single qubit measurement. Consider the entangled Bell state  $|\phi^+\rangle = 1/\sqrt{2}(|00\rangle + |11\rangle)$ . Provided the second qubit has not been measured before, the probability of measuring the first qubit to be  $|0\rangle$  is  $1/2$ , and vice versa. If a measurement is performed either on the first or the second qubit, the measurement of the other qubit in each case gives the same result. That is, the measurement of one qubit has an effect on the measurement of the other – the measurement outcomes are correlated.

An important principle about measurements in the context of quantum circuits is discussed in the following section.

## 2.6 Quantum Circuits

The quantum circuit model of computation [44] is analogous to the classical circuit model. A classical circuit is made of gates computing Boolean functions and wires that connect gates.<sup>9</sup> A quantum circuit has nearly the same structure, but with some restrictions.

Of course, gates in quantum circuits are quantum gates. Their graphical representation is already described in Section 2.4. Wires, symbolized by horizontal lines, connect quantum gates and indicate input and output of a quantum circuit. Each wire represents a certain qubit. However, wires generally do not correspond to physical wires, but refer instead to a “time line”. The quantum circuit fixes the chronological order of unitary transformations (plus some measurements) which are applied successively to an initialized quantum state. If not stated otherwise the input state of an  $n$ -qubit quantum circuit is the computational basis state  $|0\rangle^{\otimes n} = |0\rangle \otimes \dots \otimes |0\rangle$ . By convention a quantum circuit has to be read from left to right. The simplest quantum circuit is a single quantum gate. However, to implement this gate, it usually has to be decomposed into a sequence of elementary gates.

The following rules specify the restrictions of quantum circuits compared with classical circuits and define allowed connections of quantum gates in the common quantum circuit model:

---

<sup>9</sup>Formally, it can be described by a (acyclic) directed graph whose vertices represent the gates and whose directed arcs represent wires.

- Only acyclic circuits are valid quantum circuits, i.e. loops are not allowed. A quantum gate which is to be applied repeatedly has to be wired as many times in the quantum circuit.
- Also, wire crossings are not allowed since arbitrary qubit permutations can be realized by *SWAP* operations.
- As a result of the reversibility of operations in quantum circuits it is not allowed to perform the *FANIN* operation, which joins several wires to a single wire containing the bitwise *OR* of the inputs.
- Moreover, the number of input and output wires or qubits respectively is exactly the same.
- In contrast to classical circuits, it is not possible to split a wire into two or more identical wires, which means, the *FANOUT* operation is not allowed. This corresponds to the following theorem.

**Theorem 6.** (*No-cloning theorem*) It is not possible to make a copy of an unknown quantum state!

A proof of this theorem is given e.g. in [107]. Even though (universal) cloning is not possible, classical information can be copied with perfect fidelity, as any particular pair of orthogonal states can be cloned perfectly.

In the following, the main aspects of quantum circuits are summarized, beginning with a working definition of a quantum circuit and continued with explanations on inputs and outputs of quantum circuits.

A *quantum circuit* is a quantum computational model operating on a finite number of qubits. A *quantum circuit on  $n$  qubits* is a unitary operation on  $\mathcal{H}^{(n)}$  which can be represented by a finite concatenation of quantum gates. In practice, the unitary operation is to be composed of elements from a (small) finite set of quantum gates which form a universal basis for quantum computation. Since discrete, universal gate sets realize any unitary operation with arbitrary accuracy (Theorem 4 and 5), this restriction (also important in the context of circuit evolution) does not limit the set of computable functions.

A quantum circuit can get its *input* in two ways: (i) by the initial quantum state (the state of the qubits) or (ii) by input gates (or oracle gates), that is, unitary operations which depend on the input. The first approach is more intuitive and corresponds to the way classical circuits obtain their input. However, encoding inputs may lead to quantum systems with several qubits. Since the costs for circuit evaluations on conventional computers increase exponentially with the number of qubits, large numbers must be avoided (cf. Chapter 5). This is possible using the second approach. Oracle or input

gates usually substitute much larger quantum circuits and hide qubits necessary to encode further problem inputs and ancillary qubits. Yet, the use of oracle gates leads to a different complexity measure, if it is assumed, that the oracle performs its operation in a single time step (cf. Section 2.6.2). Using input gates does not contradict to the definition of quantum circuits given above. The input gate is merely integrated in the unitary operation and can be seen as an element of the elementary gate set.

Applying the quantum circuit means multiplying the unitary operation with the initial quantum state. The resulting vector provides the probabilities for all measurement outcomes. A measurement is necessary to obtain any information. From this the *output* of the quantum circuit can be inferred. How this is done is essentially a convention. For instance, for a decision problem one can define a particular qubit to carry the answer. In a different way the output of the Deutsch-Jozsa algorithm (described at the end of this chapter) is obtained. Here, the measurement result of the entire quantum system is decisive for the outcome: One basis state encodes the answer “f is constant”, all others the answer “f is balanced”. Moreover, for optimization problems every quantum state may encode a certain solution. So, there are usually many ways to define the output modalities and the decision in favor of either way will affect the quantum circuit solving a given problem.

Note, it is still an open issue whether there exist other models of computation which are more powerful than the quantum circuit model.

### 2.6.1 Intermediate Measurements

The final element in quantum circuits is the measurement. In circuits without explicit measurements at the end they are implicitly assumed. However, measurements can also be performed as an intermediate step in the circuit. Moreover, they allow conditional branchings in quantum circuits. The advantage of intermediate measurements is that they tend to make quantum circuits more “readable” and interpretable, when they are used in a clever and effective way.

In case of a single-qubit intermediate measurement, dependent on the measurement result ‘0’ or ‘1’ one of two quantum subcircuits is applied and describes now the continued evolution of the quantum system. Multiple-qubit intermediate measurements are composed of single-qubit intermediate measurements. Therefore, it is sufficient to focus on the latter. The possibility to use intermediate measurements extends the quantum circuit model described above and requires a new definition: A *quantum circuit on  $n$  qubits with intermediate measurements* is a binary tree with quantum (sub)circuits on  $n$  qubits as nodes. In addition, all inner nodes of the tree are labelled with a single-qubit measurement (the qubit it acts on). By definition the left subtree corresponds to measurement outcome ‘1’, the right to measurement outcome ‘0’.

Applying such a circuit means evaluating a path from the root to a leaf: The quantum circuit in the root is applied to an initial quantum state. Each application of a quantum

circuit belonging to an inner node is followed by a single qubit measurement which determines the next subtree and the new quantum state the subcircuit is applied to.

According to the *quantum principle of deferred measurement*, “measurements can always be moved from an intermediate stage of a quantum circuit to the end of the circuit” [107]. Of course, such a shift has to be compensated by some other changes in the quantum circuit. The transfer of a quantum circuit with intermediate measurements to a quantum circuit without them might lead to a much larger quantum circuit (in the number of elementary quantum gates).

Note that this circuit model with intermediate measurements seems to be different to the circuit model (with intermediate measurements) roughly described by Nielsen and Chuang [107]. Moreover, the quantum circuit models with and without intermediate measurements are computationally equivalent, but it is not quite clear how the circuit size of a quantum circuit with intermediate measurements is related to the circuit size of its equivalent circuit without intermediate measurements.

### 2.6.2 Circuit Complexity Measures

There are two important measures to quantify the “costs” of a quantum program: first, the total number of quantum gates concerning a given elementary gate set and, second, the number of qubits needed to implement the circuit. To find the optimal quantum circuit both parameters have to be optimized which might be a conflicting objective. However, suppose a sufficient number of qubits is available. Then, the efficiency can also be measured by the time passing from the initialization of the system state to the final measurement, which corresponds to the number of computational steps. Here, a computational step means the application of (maximally) parallel quantum gates. Figure 2.12 gives an example.

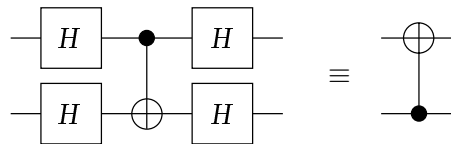


Figure 2.12: The left quantum circuit consists of five elementary quantum gates. Since two Hadamard gates before and after *CNOT* each can be applied parallel the quantum circuit consists of three computational steps. However, the right quantum circuit is equivalent to the left circuit but consists only of a single *CNOT* gate.

If choosing the number of quantum gates as a measure, the simple uniform valuation model, i. e., each gate contributes the same costs, may be replaced by a more “realistic” measure, depending on a certain physical realization, which e. g. weights the costs according to the gate type. Independent of the physical implementation it might make sense, for instance, to rate controlled gates higher than single qubit gates.

Comparisons between quantum algorithms which use oracle gates, so-called quantum black-box algorithms, and their corresponding classical algorithms are based on the number of accesses to input data, that is, the number of oracle calls, instead of the number of computational steps, or the number of elementary gates respectively (cf. also Section 4.4). This complexity measure is also denoted as *query complexity* or *decision tree complexity*. Of course, a comparison of quantum and classical algorithms on the basis of oracle calls is only reasonable, if both algorithms use the same oracle.

Behind this is another computation model, the *decision-tree* or *query model*. Strictly speaking, in the quantum version of this model (deterministic and randomized decision-trees are two other kinds) quantum circuits are studied which can be described by a unitary transformation  $A = U_T \cdot O \cdot U_{T-1} \cdot O \cdot U_{T-2} \cdots O \cdot U_1 \cdot O \cdot U_0$ , where the  $U_i$  denote fixed, input-independent unitary transformations and the gate  $O$  an oracle call (input-dependent unitary transformation). Relevant from the viewpoint of complexity is the number of queries  $T$ . A survey on the decision-tree model and its complexity is given in [26].

Why investigating such a restricted model? As it is not possible to make decisive complexity theoretical statements in more powerful computation models (this would comprise the answer to the question, whether quantum computing is more powerful than classical computing), simpler and more limited models of computation are analyzed. The hope is that understanding of such easier models will lead to a better understanding of the more complex models.

## 2.7 Quantum Computational Complexity

Like other promising non-standard computing approaches, quantum computing raised the hope that NP-Complete problems which seem to be intractable for classical computers could be solved efficiently. Due to the merely quadratic speedup and the optimality of Grover's quantum search algorithm (cf. Section 4.2), it is obvious that approaches primarily based upon (unstructured) quantum search cannot yield efficient solutions to the problems in NP, in particular to the NP-complete problems. This can be considered to be an indication that the class of NP-complete problems cannot be solved efficiently on a quantum computer. However, up to now this is neither proven nor disproven.

Apart from NP-complete problems, there are some problems, which seem to be intermediate in difficulty between P and NPC problems. This class of (decision) problems is denoted NPI (NP-incomplete) and it is  $NPI := NP - NPC - P$ . Ladner [92] has shown that NPI is not empty, iff  $NP \neq P$ . Thus, finding a problem in NPI would solve the most important and famous problem in computer science. For instance, the decision problem of factoring<sup>10</sup> is regarded as a candidate for NPI. Another problem which is still believed to

<sup>10</sup>Given a composite integer  $m$  and  $l < m$ , decide whether  $m$  has a non-trivial factor less than  $l$ .

be in NPI is *graph isomorphism*. Such problems appear to be hard classically, but they can perhaps be solved efficiently on a quantum computer, as it was shown for factoring.

Another example for a classically hard problem, but not proven to be in NP-complete, which perhaps can be efficiently solved on quantum computers is the *shortest lattice vector problem* (SVP)<sup>11</sup>. Like factoring, the difficulty of this problem ensures security in public key encryption systems (by using the inverse of this problem as a one-way function). However, the provability of Micciancio's number theoretical conjecture would lead to  $SVP \in NPC$  [150], and this would argue against an efficient quantum solution.

From these short and unfinished reflections, it becomes convincing that quantum computing needs a separate complexity theory to understand the potentials and limitations of quantum computing compared to classical computing.

Different complexity measures for quantum circuits were already explained at the end of the last section. Another computation model is the *Quantum Turing Machine* (QTM) which is not explained here. While quantum circuits (like Boolean circuits) are a non-uniform computation model, because they have only constant input length, QTMs (like classical TMs) are uniform, as they work on arbitrary input lengths. As in classical complexity theory, the uniform quantum complexity of computational problems is of particular interest. Therefore, analogous to uniform Boolean circuit families, *uniform quantum circuit families*<sup>12</sup> are considered. They allow a comparison of the computational power of the more abstract QTMs and the more practical (uniform) quantum circuits: It can be shown that QTMs with polynomial runtime can be simulated by uniform quantum circuits of polynomial size (with bounded error probability) and vice versa [169, 108]. See [108] for a more detailed investigation on the equivalence of the computational powers of uniform quantum circuit families and QTMs.

The following paragraph summarizes some results about quantum complexity classes (defined over QTMs) and their relation to classical complexity classes. In [17] Bernstein and Vazirani introduce quantum analogons to the classical complexity classes  $P$  and  $BPP$ <sup>13</sup>  $EQP$  corresponds to  $P$  and denotes the class of problems solved error-free on a QTM in polynomial time.  $BQP$ , the quantum version of  $BPP$ , denotes the complexity class of all computational decision problems that can be solved with bounded error probability on a QTM in polynomial time.

---

<sup>11</sup>The shortest lattice vector problem consists in finding the shortest non-trivial vector of a lattice  $L$  generated by  $d$  linear independent vectors in the vector space  $\mathbb{Q}^d$

<sup>12</sup>A uniform quantum circuit family is an infinite sequence of circuits  $C_n$  for each input length  $n$  such that  $C_n$  can be generated by a QTM on input  $n$  in polynomial time  $O(poly(c(n)))$  where  $c(n)$  is the size of quantum circuit  $C_n$  based on a given elementary gate set.

<sup>13</sup> $BPP$  (**p**robabilistic **p**olynomial with **b**ounded **e**rror) is the classical complexity class of decision problems that can be solved in polynomial time on a probabilistic Turing machine with bounded error probability.

How EQP and BQP exactly relate to the classical complexity classes P, NP, PP<sup>14</sup>, BPP and PSPACE<sup>15</sup> is unknown. What is known is that

$$BPP \subseteq BQP \subseteq PP \subseteq PSPACE \quad \text{and} \quad P \subseteq EQP \subseteq BQP.$$

It is still an open problem to determine which of the inclusions are proper inclusions and which are not. Especially it is not proven that  $BQP \neq BPP$ , that is, that quantum computers have capabilities beyond those of classical computers, although there is strong evidence suggesting this.

## 2.8 Basic Programming Techniques and Simple Quantum Algorithms

It is *assumed* that certain computational problems can be solved on a quantum computer with a lower complexity than possible on classical computers since there are problems which are efficiently solvable by quantum algorithms but not classically up to now. In doing so, quantum algorithms take advantage of basic computational techniques, namely *superpositioning*, *quantum parallelism* and *quantum interference*. Additionally, *entanglement* seems to be a source of computational power quantum algorithms can benefit from.

In this section fundamental programming techniques are discussed on the basis of the simple quantum algorithms for Deutsch's problem or its generalization, the Deutsch-Jozsa problem. Also, a quantum algorithm for quantum teleportation is explained, demonstrating the power of quantum entanglement. Both problems, quantum teleportation and Deutsch-Jozsa, were already used as test problems for quantum circuit evolution (cf. Chapter 5.3).

### 2.8.1 The Deutsch-Jozsa Problem

Given a Boolean function  $f : \{0,1\}^n \rightarrow \{0,1\}$  as a black-box, promised to be either *constant*,  $f(x) = c, \forall x \in \{0,1\}^n$  and  $c \in \{0,1\}$ , or *balanced*, i. e., as the result of  $f$ , 0 occurs as many times as 1, the *Deutsch-Jozsa problem* (DJ for short) is to determine which of the two properties  $f$  has. This problem is also referred to as the *early promise problem*. For  $n = 1$ , the task is also known as *Deutsch's problem*.

The number of Boolean functions being either constant or balanced amounts to  $2 + \binom{2^n}{2^{n-1}}$ . In classical (deterministic) computing  $f$ , has to be evaluated  $2^{n-1} + 1$  times in the worst case; in quantum computing a single application of the corresponding input

---

<sup>14</sup>PP is the classical complexity class of problems solved by randomized algorithms with unbounded error probability.

<sup>15</sup>PSPACE is the classical complexity class of all decision problems which can be solved on a deterministic Turing machine using polynomial space and arbitrary time.

$$\begin{pmatrix} (-1)^{f(00)} & 0 & 0 & 0 \\ 0 & (-1)^{f(01)} & 0 & 0 \\ 0 & 0 & (-1)^{f(10)} & 0 \\ 0 & 0 & 0 & (-1)^{f(11)} \end{pmatrix}$$

Figure 2.13: Example for an oracle matrix implementing a Boolean function  $f$  on two input qubits.

matrix is sufficient [31]. The description of the quantum circuits solving DJ is restricted to the case where the input matrix is a permutation matrix defining  $f$ . An example for such a unitary input matrix was already given in Figure 2.10. Using this matrix representation of a Boolean function,  $n + 1$  qubits are necessary to solve the problem on a quantum computer. Though, there is also a reduced  $n$  qubit version of this algorithm which does without an additional ancillary qubit, using a diagonal representation matrix with coefficients  $(-1)^{f(x)}$  (cf. Figure 2.13).

Mathematically expressed, the input gate  $U_f$  acts on the computational basis for any Boolean function  $f$  as follows:

$$|x\rangle|y\rangle \longrightarrow |x\rangle|y \oplus f(x)\rangle$$

where  $|x\rangle$  is the “data” register describing the input for  $f$  and  $|y\rangle$  is a single “target” qubit. Here,  $\oplus$  stands for addition modulo 2.

The quantum algorithm solving the Deutsch-Jozsa problem is discussed in detail in [65, 107]. In the following, a short summary is given with a view to the basic programming techniques.

### Superpositioning

The preparation of an equal superposition is a decisive step in most quantum algorithms. Since

$$H|x\rangle = \frac{1}{\sqrt{2}} \sum_{z=0,1} (-1)^{xz} |z\rangle,$$

for  $x = 0$  and  $x = 1$ , applying  $H^{\otimes n}$  on an arbitrary basis state  $|x\rangle \in \mathcal{H}^n$ ,  $x \in \{0, 1\}^n$  results in the state

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} (-1)^{x \cdot z} |z\rangle, \quad (2.2)$$

where  $x \cdot z \equiv x_1 z_1 + \dots + x_n z_n$ . The even superposition with amplitude  $\frac{1}{\sqrt{2^n}}$  for every base state is achieved by applying  $H^{\otimes n}$  to  $|0\rangle$ .



It may help to grasp the benefit of superpositioning by interpreting the resulting superposition state as  $2^n$  classical equally weighted computation paths that the quantum computer follows simultaneously. In this sense, superpositioning is the first step on the way to quantum parallelism.

### Quantum Parallelism

This is another fundamental feature of quantum computing used in many quantum algorithms — usually in those where black-box or oracle gates are applied to compute a function  $g : \{0, 1\}^n \rightarrow \{0, 1\}^m$ . Since the mapping  $x \rightarrow (x, g(x))$ ,  $x \in \{0, 1\}^n$ , is reversible, there is a unitary transformation  $U_g$  simulating the classical circuit calculating  $(x, g(x))$ , such that  $|x, y\rangle \rightarrow |x, y \oplus g(x)\rangle$  for any  $y \in \{0, 1\}^m$ . Ancillary qubits, which are needed to implement the reversible circuit are omitted.

The “black-box” transformation  $U_f$ , described above, is such a unitary transformation related to the given Boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ . If  $|y\rangle$  is initially in state  $|0\rangle$ , it receives after applying  $U_f$  the output of  $f(x)$  and the resulting state is  $|x, f(x)\rangle$ . Quantum parallelism now refers to the effect resulting from the application of  $U_f$  on a superposition state, representing different values of  $x$ . For instance, applying  $U_f$  to  $|x, y\rangle = |\psi, 0\rangle$  with

$$|\psi\rangle = H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2^n}} \sum_{z \in \{0, 1\}^n} |z\rangle$$

results in

$$U_f|\psi, 0\rangle = \frac{1}{\sqrt{2^n}} \sum_{z \in \{0, 1\}^n} |z, f(z)\rangle, \quad (2.3)$$

a superposition of all possible function values. Thus, a single function or “black-box” evaluation<sup>16</sup> is sufficient to calculate  $f(x)$  for all its possible inputs in parallel. However, only one function value is actually accessible by measurement because readout always collapses superposed states to a single basis state.

### Computing a Function Into the Phase

Applying  $U_f$  to the superposition  $|\phi\rangle := 1/\sqrt{2}(|0\rangle - |1\rangle)$  in the single target qubit leads to

$$U_f|x, \phi\rangle = \frac{1}{\sqrt{2}}(|x, f(x)\rangle - |x, 1 \oplus f(x)\rangle) = (-1)^{f(x)}|x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = (-1)^{f(x)}|x, \phi\rangle.$$

Neglecting  $|\psi\rangle$ , this can be shortened by defining a new transformation

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

---

<sup>16</sup>A single application of the hidden quantum circuit.

Thus,  $V_f$  enables the *computation of the function values into the phase* or, in other words, it transfers the values of  $f$  from the basis states to the amplitudes relative to the basis states. Now, applying  $V_f$  on  $|\psi\rangle$  produces the state

$$|\psi'\rangle = V_f|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} (-1)^{f(z)} |z\rangle. \quad (2.4)$$

This corresponds to  $U_f H^{\otimes(n+1)}(|0\rangle^{\otimes n} |1\rangle)$  neglecting the target qubit.

### Quantum Interference

The effect of interference is best described (following [115]) considering the Hadamard transformation and its action on  $|0\rangle$ ,  $|1\rangle$ ,  $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ . It is quite clear that  $H|0\rangle = |+\rangle$  and  $H|1\rangle = |-\rangle$ , defining  $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ . So, both resulting states, Hadamard applied to  $|0\rangle$  and Hadamard applied to  $|1\rangle$ , have the same probability distribution. Furthermore,  $|+\rangle$  is only a superposition of both  $|0\rangle$  and  $|1\rangle$ . Therefore, classically one could assume that the resulting state of Hadamard applied to  $|+\rangle$  shows the same probability distribution, however,  $H|+\rangle = \frac{1}{\sqrt{2}}(|+\rangle + |-\rangle) = |0\rangle$ . That is, the probabilities interfere with each other; they add on one hand and cancel on the other. Interference allows computations on superposed states to interact[115].

Interference is also the source for solving DJ: Applying  $H^{\otimes n}$  on  $|\psi'\rangle$  (using Equations 2.4 and 2.2) results in

$$\frac{1}{2^n} \sum_{z \in \{0,1\}^n} \sum_{x \in \{0,1\}^n} (-1)^{x \cdot z + f(x)} |z\rangle. \quad (2.5)$$

This state is actually the result of the quantum circuit solving the Deutsch-Jozsa problem. Considering the property of  $f$ , the solution of the question, whether  $f$  is constant or balanced, is directly readable: The amplitude for  $|0\rangle^{\otimes n}$  is  $\sum_x (-1)^{f(x)} / 2^n$ . If  $f$  is constant, this amplitude becomes  $\pm 1$  and the measurement outcome is with certainty  $|0\rangle^{\otimes n}$ . If  $f$  is balanced, the contributions to the amplitude of  $|0\rangle^{\otimes n}$  cancel out and the amplitude becomes 0. A final measurement will measure anything but  $|0\rangle^{\otimes n}$ . Thus, the quantum algorithm determines clearly the property of  $f$ .

The entire quantum algorithm for the general Deutsch-Jozsa problem is summarized below, Figure 2.14 illustrates the quantum circuit.

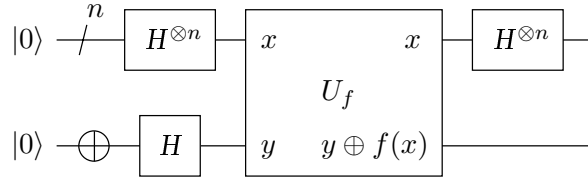


Figure 2.14: Quantum circuit implementing the general Deutsch-Jozsa algorithm. The top wire carries  $n$  qubits. The  $U_f$  gate represents a black-box quantum circuit which maps the input  $|x, y\rangle$  to  $|x, y \oplus f(x)\rangle$ .

---

**Quantum Algorithm 2.8.1:** *Deutsch-Jozsa*

---

1. Create the superposition:

$$|0\rangle^{\otimes n} |1\rangle \xrightarrow{H^{\otimes(n+1)}} \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} |z\rangle \left[ \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \right]$$

2. Calculate  $f$  in parallel:

$$\xrightarrow{U_f} \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} (-1)^{f(z)} |z\rangle \left[ \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \right]$$

3. Consider only the  $n$ -qubit register and cause interference by applying  $H^{\otimes n}$ :

$$\xrightarrow{H^{\otimes n}} \frac{1}{2^n} \sum_{z \in \{0,1\}^n} \sum_{x \in \{0,1\}^n} (-1)^{x \cdot z + f(x)} |z\rangle$$

4. Measure quantum state with output  $z$ :

$$\xrightarrow{M} \begin{cases} z = 0 : & f \text{ is constant} \\ z \neq 0 : & f \text{ is balanced} \end{cases}$$


---

## 2.8.2 Quantum teleportation

Quantum teleportation [15, 22] is described here for two reasons. First, as already mentioned, it is also used as a test problem for quantum circuit evolution [167, 168], and second, quantum teleportation is an important application of entanglement.

The task is to move an unknown quantum state  $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$  of a single qubit (source) to another remote qubit (target) by communicating just two classical bits, assuming that the target qubit is not directly accessible, but instead is entangled to a third qubit, which is accessible. This problem is usually explained in the ordinary “Alice (A) and Bob (B) scenario”: A wants to transmit a single qubit state to B. A cannot do this directly. However, she can send classical information to B. The idea of quantum teleportation depends on entangled states.

In short, the procedure solving the problem is:

1. A or B or a third party generates an EPR pair (cf. Section 2.3.2)

$$1/\sqrt{2}(|00\rangle + |11\rangle),$$

and both, A and B, receive one qubit of this entangled pair. Assume, qubit 0 is given to B, qubit 1 is given to A (Figure 2.15, left).

2. A entangles this qubit with the source qubit (Figure 2.15, middle), with the result:

$$1/2[(\alpha|0\rangle + \beta|1\rangle)|00\rangle + (\alpha|1\rangle + \beta|0\rangle)|01\rangle + (\alpha|0\rangle - \beta|1\rangle)|10\rangle + (\alpha|1\rangle - \beta|0\rangle)|11\rangle].$$

3. A measures the two qubits in her possession, influencing also B’s qubit, and sends the (classical) outcome to B (Figure 2.15, middle). For example, if A measures and transmits 11, B knows that his qubit state changed to  $\alpha|1\rangle - \beta|0\rangle$ .
4. Knowing the state  $xy$ , makes it easy for B to restore  $|\phi\rangle$  using two qubits, prepared to  $|xy\rangle$ , and two controlled gates, *CNOT* and *CZ* (Figure 2.15, right). For 11, he has to apply both, a bit flip (by applying *NOT*) and a phase flip (by applying *Z*):

$$(\alpha|1\rangle - \beta|0\rangle)|11\rangle \xrightarrow{\text{CNOT}} (\alpha|0\rangle - \beta|1\rangle)|11\rangle \xrightarrow{\text{CZ}} (\alpha|0\rangle + \beta|1\rangle)|11\rangle.$$

In the following some reflections on the difficult nature of entanglement are summarized.

### Entanglement

The mathematical definition of entanglement was already given in Section 2.3.2. A simple fact about entangled states (in the present mathematical model) is usually given to emphasize their impact on quantum computing: The number of parameters, needed to describe a (pure) unentangled state in  $\mathcal{H}^{(n)}$  which is just a tensor product of single qubits, increases only linearly with the number of qubits  $n$ . But describing a general state (unentangled and entangled) still requires exponentially many ( $2^n$ ) vector coefficients.

Yet, the question of whether entanglement is a key resource for computational power or not has no simple answer. In [81] Jozsa and Linden discuss this topic in more detail. They prove that for any quantum algorithm operating on pure states it is necessary

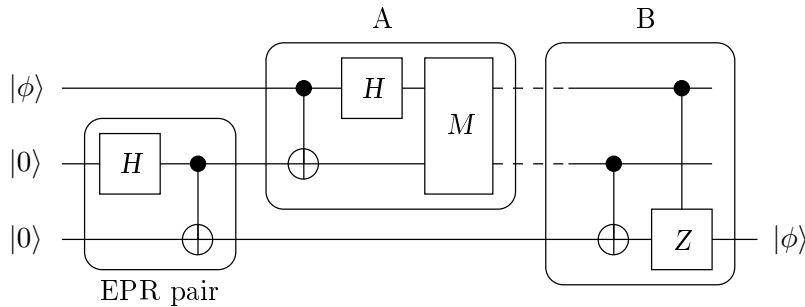


Figure 2.15: The three parts of the entanglement algorithm: generation of the EPR pair, A's "send"- and B's "receive"-part. The dotted part of the wires indicate the classical transmission. On receipt, B has to prepare two qubits according to the classical information.

that the number of entangled qubits increases unbounded with the input size to speed up classical computation. Furthermore, they show that only quantum algorithms with small amount of entanglement can be efficiently simulated by a classical algorithm within a certain tolerance. Independently, G. Vidal establishes in [159] *how* to simulate slightly entangled quantum computations efficiently on a classical computer with cost growing linearly in the number of qubits and exponentially in the entanglement.

These are strong arguments in favor of entanglement as a driving force (in pure state quantum dynamics). However, Jozsa and Linden also argue that, "It is nevertheless misleading to view entanglement as a key resource for quantum computational power", since the significance of entanglement is not independent of the underlying mathematical formalism, namely the amplitude description. Instead, for other mathematical formalisms entanglement seems to have no certain meaning [81]. Moreover, to rate the genuine computational power it is also important to consider that the information actually accessible by measurement is still linear despite the exponential growth of the Hilbert space [119]. Therefore, the above argument might be misleading.

Increasing multi-partite entanglement is exemplarily shown for Shor's factoring algorithm [81]. For quantum search, which has only a quadratic speed up over classical search, it is proven that there exists also a better-than-classical quantum algorithm, which does not require entanglement [100] but instead is based on interference. For the reduced Deutsch-Jozsa algorithm it can be shown that entanglement does not occur until  $n \geq 3$  [34].

The power of quantum computing is not fully understood. Not all of its beneficial sources seem to be identified or are at least not independent of the mathematical models used. The answers to both questions, "What is the excess of computational power provided by quantum mechanics?" and "What gives the extra power to quantum computing?" still remain as great challenges.



## 3 Genetic Programming Fundamentals

*That which is static and  
repetitive is boring. That which  
is dynamic and random is  
confusing. In between lies art.*

---

John A. Locke (1642–1704)

The term *genetic programming*, GP for short, and its underlying programming technique goes back to J. Koza and his treatise entitled “Genetic Programming. On the Programming of Computers by Means of Natural Selection.” [87]. Briefly, genetic programming is an *automatic programming* method based on Darwinian principles of evolution forcing the induction and synthesis of computer programs.

In the present chapter, the basics of GP are discussed. In Section 3.1 GP is considered in the context of its scientific roots: machine learning, Darwinian evolution theory and evolutionary computation or evolutionary algorithms, respectively. Section 3.2 deals with program structures in GP, introducing tree GP, linear and linear-tree GP, and graph and linear-graph GP. Section 3.3 describes how genetic operators act on GP individuals. Fitness functions and selection mechanisms are discussed in Section 3.4. The two basic GP algorithms, generational and steady-state GP, are introduced in Section 3.5. Finally, Section 3.6 goes into more profound subjects of evolution: introns and neutrality and their effects on evolution.

A detailed and comprehensive introduction to GP can be found in [8]. This textbook also forms the basis of this chapter.

### 3.1 GP’s Scientific Roots

The field of automatic programming focuses on inducing computers to develop programs or algorithms which enable them to perform certain tasks. The process of improving computer algorithms iteratively through a “gain of experience” is called *machine learning* (ML) [103]. Experience is acquired by training computer programs on a set of *training instances* which are examples of the relationship between inputs and desired outputs related to a given task or problem. In GP, these training instances are called *fitness cases*. By comparing the outputs of the programs (applied to the training inputs) with

the correct outputs, the ML system is able to “learn”. The distance (based on a defined metric) of the result to the desired output is a measure for the excellence or *fitness* of this program. This learning approach is also called *supervised learning* and is typical for GP systems<sup>1</sup>. However, in this context “learning” does not only mean to realize the weakness of considered computer programs but also to improve them, i. e. to search in the space of all computer programs for those which are better adapted to the training conditions. The final goal of the learning process is a computer program that is able to calculate the desired outputs of the training set from the inputs. The quality of learning can be appraised by applying the best solutions of the ML system to a *test set* containing examples of input-output instances different from the training set. There are many different learning algorithms in ML according to the problem representation. The main differences are how solutions (computer programs) are represented, the selection of search operators (for searching through the solution space) and the search strategy, e. g. blind search, hill climbing or beam search<sup>2</sup> which is GP’s search strategy.

GP’s learning algorithm is inspired by the theory of biological or natural evolution. Slightly simplified, biological evolution is genetic change in a population from one generation to another. Speed and direction of this change are variable with different biological species. Genetic variation in a population is the result of two principal mechanisms, *mutation* and (primarily homologous sexual) *reproduction* through *recombination*<sup>3</sup>. In principle, changes in an organism’s genetic code, the *genotype* (genome), can lead to changes in the organism’s *phenotype* (phenome), the set of its observable properties. In other words, some of the variations in the phenotype of individuals within a population are reflections of variations in the genotype. The phenotype is where *natural selection* – evolution’s main engine – acts upon: In the struggle for life, individuals with properties best adapted to their environmental conditions will have the best chance to survive and to reproduce. Charles Darwin called this principle of preservation “survival of the fittest” [39]. Those adaptations which are inheritable (by means of genetic code transfer), will be passed on to the offspring. In this sense heredity is a key mechanism in evolution. As a result of evolution over the course of time, species modify their phenotypes in ways permitting them to succeed in their (sometimes changing) environment. It remains to be shown, how these biological principles and mechanisms are implemented and how artificial, computer-controlled evolution works in GP.

Since the end of the 1950’s in computer science diverse branches of research emerged using the principle of evolution for problem solving. Evolutionary programming (EP) [53, 54], evolutionary strategies (ES) [124, 131] and genetic algorithms (GA) [74] are

---

<sup>1</sup>In ML other learning approaches are known which can be found in e. g. [8] and are not listed here.

<sup>2</sup>In beam search a constant number of possible solutions, the “beam”, is considered. The best solutions concerning a certain evaluation metric are selected for further transformation by applying some search operators. All other solutions are discarded and replaced by the newly obtained solutions. Together with the selected solutions they build a new beam.

<sup>3</sup>In bacteria also non-homologous (non-sexual) transfer of genetic material is possible.



counted among GP's most influential predecessors. Nowadays, techniques for simulating evolution are included in the umbrella term *evolutionary algorithms* (EA). Although these techniques may differ in essential aspects, the basic ingredients are nearly the same [8]. These are:

- populations of solutions (enabling a parallel search process),
- innovation (by mutation) and conservation (by recombination) operators,
- quality differentials (fitness measure),
- selection mechanism.

In contrast to other evolutionary approaches, the search space of all possible problem solutions in GP consists of computer programs. Like any other EAs, GP operates on a population, usually a small random subset of the entire solution space. During an evolutionary process as is illustrated in Figure 3.1, GP optimizes the fitness of the population, that is, ideally GP transforms the pool of individuals from a more or less random initial population to a pool of best-adapted individuals.

Analogous to nature, GP rates individuals (computer programs) according to their fitness – the quality of their environmental adaptation. GP evaluates the individual's fitness on the basis of a given *fitness function*. Through the selection step those individuals with better fitness values prevail over weaker individuals: the latter are eliminated from the population and replaced by variants of the former. Thus, individuals of higher fitness participate with higher probability in the evolution. The variants or offspring of individuals are created performing so-called *genetic operators* like mutation and recombination operators. They correspond to the search operators in ML. The evolutionary process is not stopped until a certain termination criterion is met, e.g. an individual with a sufficiently “good” fitness value is evolved or a maximum number of generations is reached.

This is only a rough description of the basic evolution scheme in GP. There are many different GP systems differing in program structure, selection mechanism, fitness function or genetic operators. In the following sections these components are inspected more closely. With regard to the overall subject of this thesis some remarks and examples address the evolution of quantum computer programs.

Even though GP is rather young compared to other disciplines in computer science it provides for a large number of practical relevant applications in science, computer science and engineering. [8] gives an extensive outline of problems GP was successfully applied to, including robotics [111, 113, 28], electrical circuits design [90, 91], image classification [147, 38], biochemistry data mining [123] or articulated figure motion animation [60].

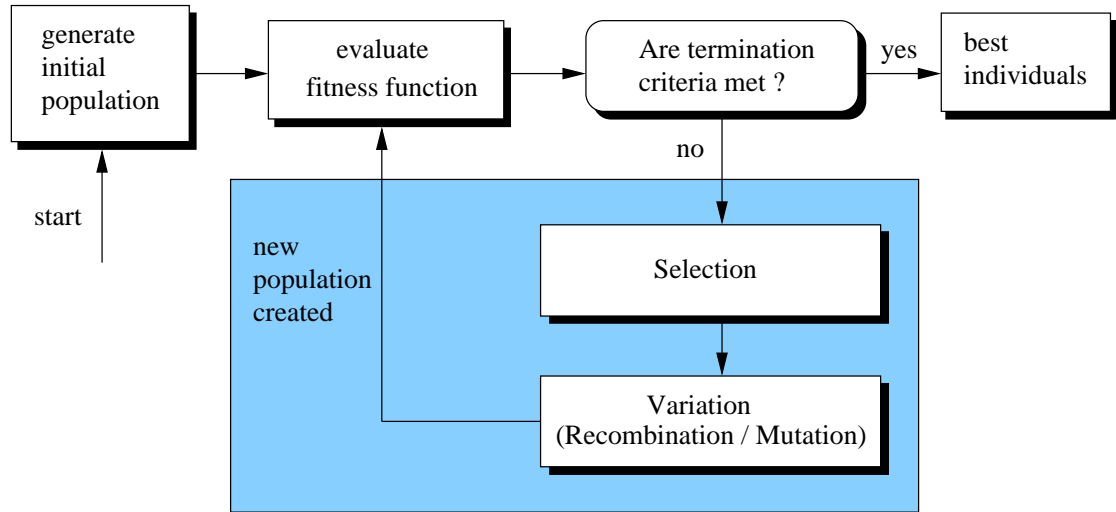


Figure 3.1: Structure of GP algorithms: the evolutionary loop of GP consists of fitness evaluation, Darwinian selection and the generation of variants (by genetic operators). The cycle is iterated until solutions are sufficiently good, or other termination criteria are met.

## 3.2 Fundamental Program Structures

The structure and assembling of computer programs in GP have both been unconsidered so far. Usually GP programs consist of basic “building blocks”, like constants, inputs to the GP program, operators or certain functions. These are the primitives (the genome of) a GP program is composed of. In the case of evolving quantum programs, quantum gates or matrices respectively are the main components of GP individuals. The building blocks are combined in defined program structures which determine the program execution, use and locality of memory, and the application of genetic operators.

In principle three different variants of program structures are used in GP: *tree*, *linear* and *graph*. Besides these basic representation forms there are newer developments of GP structures like *linear-tree* and *linear-graph*.

### 3.2.1 Tree GP

Koza’s original GP system was based on a tree structure as program representation [87, 88]. To implement his approach, Koza used the applicative programming language LISP, in which individuals are stored as symbolic expressions (*S-expressions*). An S-expression is a list of expressions, where the first is an operator and the additional expressions are its arguments, which can be either *atoms*, the values in LISP, or other S-expressions, whose return values, when evaluated, are atoms again. A simple example is

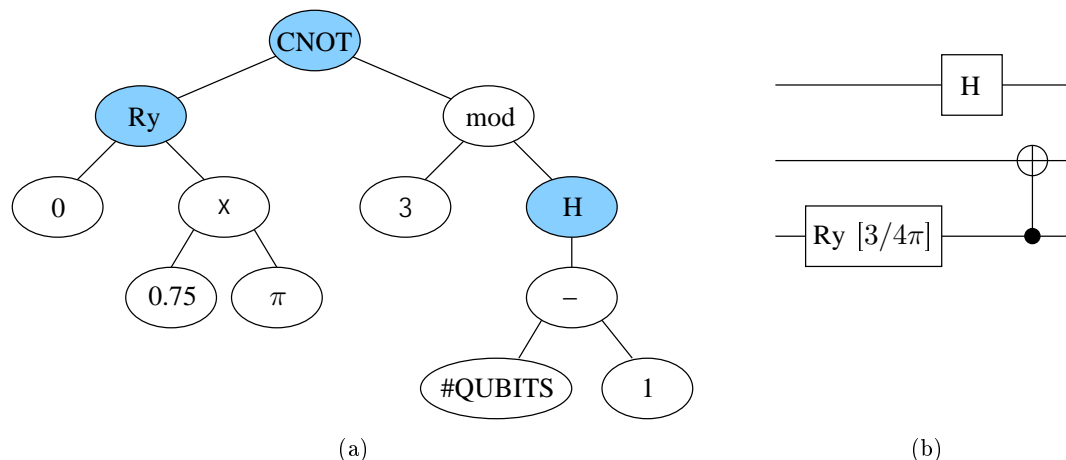


Figure 3.2: (a) Tree structure: example for an individual representing a quantum circuit. The colored tree nodes represent well-known quantum gates. A quantum gate (operator) for which all inputs, i. e. its parameters, are available is added to an initially empty quantum circuit. Like all operators (inner nodes) in tree GP quantum gates have to return a certain value, too. For example, define the Hadamard Gate  $H$  taking one argument (a valid qubit index) and returning this, the rotation gate  $Ry$  taking two arguments (qubit index from the left and rotation angle from the right subtree) and returning the first and the  $CNOT$  gate taking two arguments (control qubit index from the left and target qubit index from the right subtree) and passing on the first argument. (b) Using postfix order, tree execution results in a (probably useless) quantum circuit for a 3-qubit quantum computer.

$(+(\times ab)c)$  which multiplies  $a$  and  $b$  and adds the result to  $c$ . S-expressions are equivalent to *expression trees* – binary, ordered trees, where atoms are the leaves of the tree. In this way, LISP is cut out for representing and manipulating GP tree structures. However, tree GP can be implemented in other programming languages, too. Therefore, tree structures are considered now more generally. The parameters of a function or an operator node, respectively, are given by its child nodes, that is, every inner node needs to return a value (operand) to its father node. Tree leaves may contain constants, input values or zero-argument functions. There are different orders to execute tree structures as programs. The standard convention for tree execution is the so-called *postfix order*: a postfix traversal visits the left subtree first, then the right subtree and finally the root. Tree structures do not need external memory since the operands are always locally accessible to the operator node. Figure 3.2(a) gives an example for a tree-structure individual, interpretable as a quantum computer program. The corresponding graph of the quantum circuit on a three-qubit quantum computer is shown in Figure 3.2(b).

The mapping from a tree (Figure 3.2(a)) to a quantum circuit (Figure 3.2(b)) is governed by certain conventions. In particular, the return value that nodes specifying the type of a quantum gate pass on to their father nodes and the interpretation of arguments received by those nodes have to be defined. The order of quantum gates in the generated circuit depends on the tree execution order. As soon as a quantum gate with all of its parameters is determined during tree execution, it is added to the quantum circuit.

Note that there is also a trivial mapping from quantum circuits to binary trees. The idea of construction is as follows: Arrange the gates (gate types) of the given circuit in a linear list. This can be seen as a (degenerated) tree with the last gate in the root and the first gate as a leaf. To provide the gates with the correct parameters further nodes have to be added to the tree resulting in branchings. The adjustment of parameters may require the change of subtrees and the addition of arithmetical operator nodes.

#### 3.2.2 Linear and Linear-Tree GP

Another widespread GP scheme is linear GP [7, 109]. Here, all *instructions* or operations build a linear list which is executed from top to bottom. The only exception of the linear program flow are jump instructions. Operators get their operands from external memory. In this context linear GP systems are distinguished according to the way in which they store additional data. In *Stack-based GP* [118] operators get their arguments from a stack and push the result back onto it. In register-based and machine-code GP [109, 110] data is available in a small number of memory registers. Instructions take their arguments from one or more (globally accessible) registers and store the result in another register. In contrast to register-based programs which are interpreted before execution, machine code instructions are executed directly by the CPU, highly accelerating the individual's execution speed. Common machine-code GP individuals are two- or three-address register machines.

An example of a linear program is given in Figure 3.3.

Essentially, linear-tree GP [82] is a synthesis of linear GP and tree GP. The structure itself can be regarded as a tree where each node consists of a linear program part and a branching function. In contrast to pure tree-GP, during the interpretation of the linear-tree structure only the nodes of one path from the root node to a leaf are visited. For each visited node the linear program is executed. Afterwards the branching function is evaluated defining the node which is to be visited next. The genetic operators are performed as it is done in standard tree-based GP on the tree structure and in linear GP on the linear program part (see below).

In quantum computing, intermediate measurements are suitable for branching functions. Depending on the result of the (partial) measurement, a linear quantum subprogram is executed. Figure 3.4 illustrates the individual structure of a linear-tree scheme representing a quantum circuit.

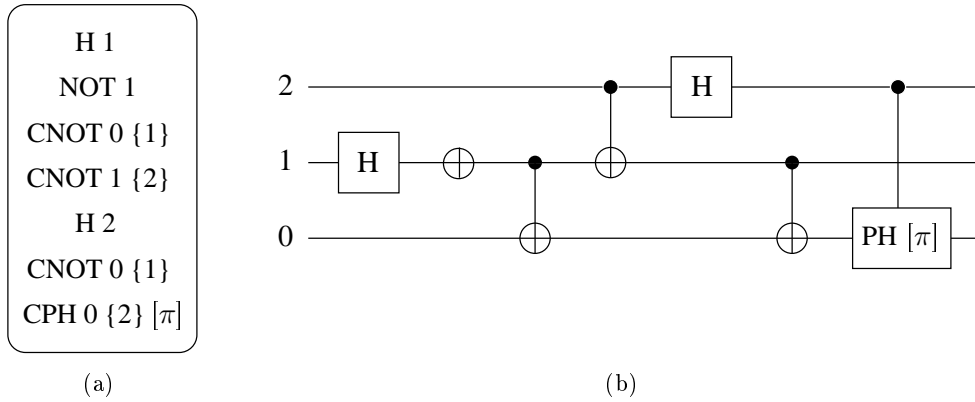


Figure 3.3: (a) Example for a quantum algorithm with linear genome structure. (b) The corresponding quantum circuit representation. Incidentally, this is a quantum teleportation circuit which moves the quantum state of qubit 2 to qubit 0 provided both qubit 0 and 1 are prepared to  $|0\rangle$ .

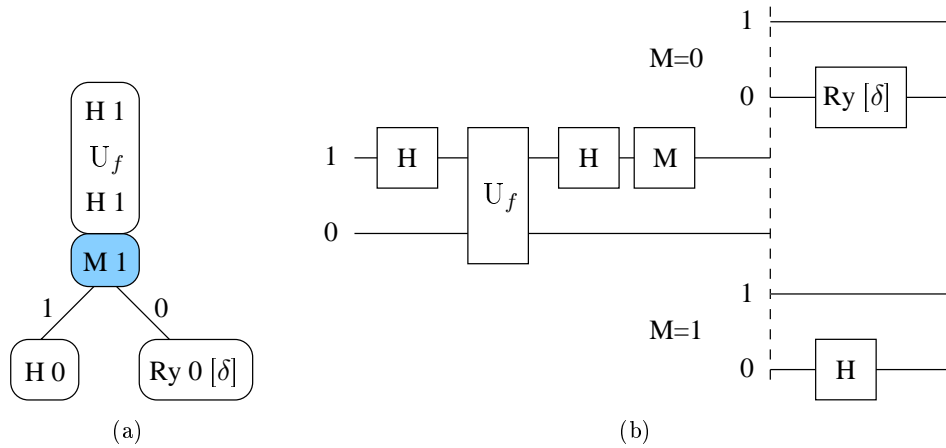


Figure 3.4: (a) Linear-tree structure of a quantum circuit using a single intermediate measurement on qubit 1. Depending on the result, the left or right subtree is executed. (b) Quantum circuit representation of (a). By the way, with angle parameter  $\delta = 0.0749$  this quantum circuit computes OR of a black-box Boolean one-bit function  $f$ , i.e.  $f(0) \vee f(1)$ , on output-qubit 0 with error probability 0.1. This is better than any classical algorithm with a single function evaluation.

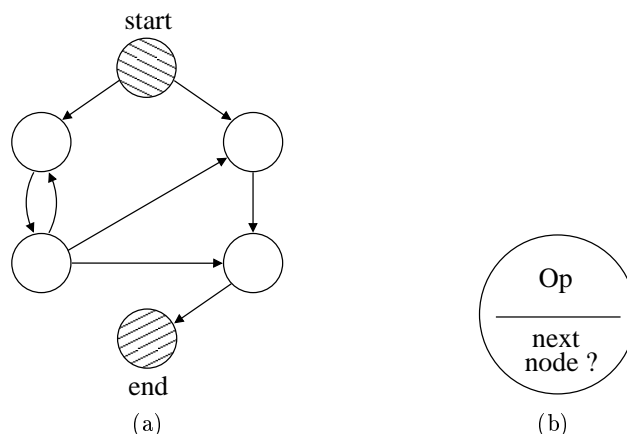


Figure 3.5: (a) Structure of a graph individual in the PADO system. Start and end node are marked. (b) Structure of a single node. When executed a node performs a certain function or operation (Op) and determines the node to be executed next.

### 3.2.3 Graph and Linear-Graph GP

Although graph-based GP structures are not part of this thesis and are not used in previous publications to evolve quantum circuits, graph and linear-graph GP are mentioned here for the sake of completeness. Since there are many different representatives of graph structures, a standard example of graph GP is discussed here: the PADO system by Teller and Veloso [147]. Graph edges are directed and define the program flow between processing nodes. The program execution starts at a certain distinguished node. At each node operations are performed reading the required operands from and writing the results back to a stack. Aside from the stack PADO also uses a globally accessible indexed memory. The next node to execute is determined by the branching function of the node. Reaching a special end node or meeting other certain conditions concludes the execution. In PADO, loops and recursions are possible preventing termination. Therefore the runtime of the program execution is limited. The structure of the PADO system is visualized in Figure 3.5.

Linear-graph GP [83] is a further development of linear-tree GP. The structure of a node in linear-graph GP is the same as in linear-tree GP: it consists of a linear program part and a branching function. However, the overall program structure is a graph which makes program flow more natural in the sense of computer programs hand-written in higher level languages.

The choice of an appropriate program structure in GP seems to be essential to the success and the performance of evolution. Differences in structure may lead to significant differences in the speed of the evolutionary process. Furthermore, the program structure affects the expressiveness and readability of code.

## 3.3 Genetic Operators

During the evolutionary process genetic operators ensure changes in the individual's genome and as a result improve the overall fitness of the population.

The three fundamental GP genetic operators are *mutation*, *recombination (crossover)* and *replication*. The replication operator simply copies the individual. The copy is placed in the population.

### 3.3.1 Mutation

This operator is applied to a single individual changing its genome randomly at one or more positions. There are many different types of mutation for each kind of program structure. In tree GP one type of mutation operator selects a node in the tree randomly and replaces it by a newly generated node. Another type of mutation replaces the entire subtree rooted at the selected node by a randomly created subtree. In linear GP a single instruction from a given individual is selected and its operands and/or arguments are changed. Mutations on graph-based individuals are even more diverse. Besides changes in the operation of a single node its branching function (and therefore the edges of the node) can be altered.

In the case of quantum circuit individuals, a single (one-step) mutation can change for instance the type of a quantum gate or the qubit the gate is applied to or it can replace the entire gate by a new gate. Further mutation operators on quantum circuit individuals are conceivable.

### 3.3.2 Recombination

In the (sexual) recombination or crossover operation, parental individuals reproduce by exchanging parts of their genomes. The resulting individuals are the children. The realization of the crossover operator depends on the structure of the individuals.

Any kind of recombination starts with the selection of the two individuals (the parents), participating in the crossover. The next steps in tree-based crossover are to select randomly a subtree in each parent and to swap these subtrees between the two individuals, resulting in two new individuals. In linear crossover, random sequences of instructions are mutually replaced in both individuals creating the two offspring. Tree-based crossover is visualized in Figure 3.6, linear crossover is described in Figure 3.7. Graph crossover is of course possible, but it is not treated here. In many GP systems crossover is the predominant genetic operation, i. e. it is performed with high probability.

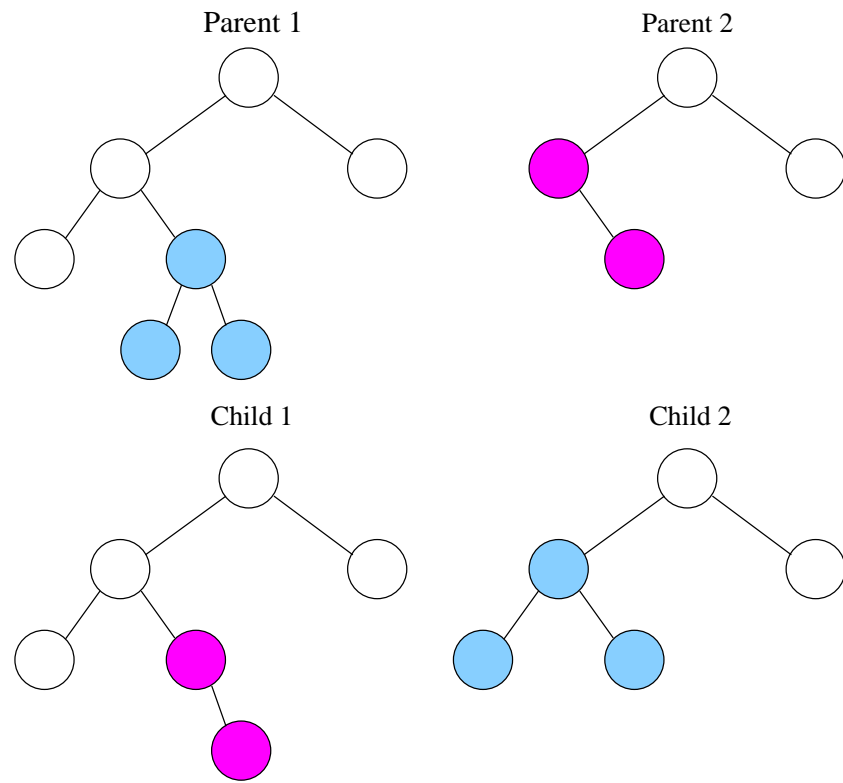


Figure 3.6: Tree-based crossover. The children are created from the parents by exchanging the colored subtrees.

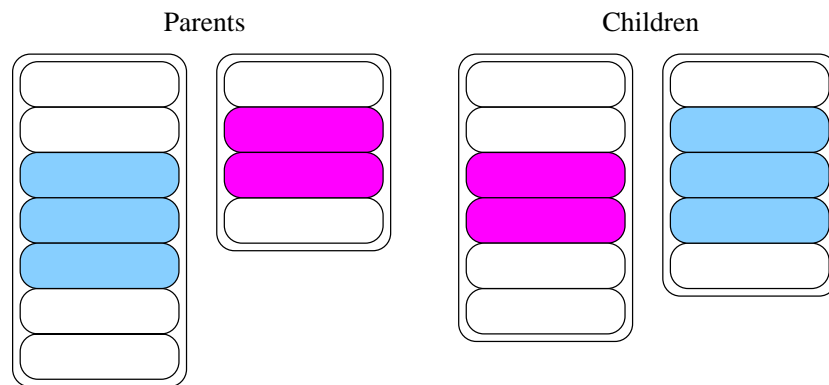


Figure 3.7: Linear crossover. The children are created from the parents by exchanging the colored sequences.



## 3.4 Fitness and Selection

The fitness function assigns every individual a certain value, the fitness value or fitness for short. This is a measure of the quality of a GP individual, i. e. of how well it performs a given task or solves a given problem, respectively. Depending on the fitness of the GP individuals, the selection algorithm decides whether an individual remains in the population or not and determines which individuals of the population are subject to the genetic operators. The selection mechanism forces evolution by competition between individuals in a population.

### 3.4.1 Fitness Functions

A fitness function is very problem specific. It is calculated on a training set of fitness cases  $\{(x_i, y_i)\}_i$  where  $x_i$  is the input and  $y_i$  is the corresponding desired output of fitness case  $i$ . Let  $y'_i$  be the output of a given GP individual on input  $x_i$ . Then, the fitness of the individual is calculated on both,  $y_i$  and  $y'_i$ . For instance, a simple fitness function for a training set of  $n$  fitness cases is

$$\sum_{i=1}^n \|y'_i - y_i\|.$$

Often, the fitness is calculated on the basis of further criteria such as the program length of GP individuals. Then the fitness function is designated as *multiobjective*.

Fitness functions are *continuous* if smaller (larger) improvements in an individual's performance are related to smaller (larger) improvements in the fitness. Usually, fitness functions are also *standardized* and *normalized*. The first property refers to fitness functions in which the fitness value zero is assigned to the fittest individual. The second property means, that the fitness value is always between zero and one.

### 3.4.2 Selection Algorithms

In general, selection is the result of a competition between individuals in a population. The better the individual's fitness in comparison with all other individuals in the population, the higher is its selection probability.

Most selection algorithms are embedded in one of the following two main selection schemes: (i) the *mating selection*, also called the GA scenario, and (ii) the *overproduction selection*, also called the ES scenario. In the mating selection, parental individuals are selected from the given population to build the basis of a new generation which is filled with their offspring. The overproduction selection chooses randomly parents from a given population to generate a set of offspring that is usually larger than the original population. By selection, the pool of offspring and (depending on the implementation) even their parents are reduced to the original size resulting in a new generation. Here,

competition results from an overproduction of individuals and the selection pressure depends on the ratio of the number of offspring (plus parents) to the population size. In both scenarios the result is a new generation, i.e. a complete population. That is why a selection algorithm following one of these schemes is called *generational*. The most popular generational selection algorithms are

- fitness-proportional selection (GA scenario),
- $(\mu^+ \lambda)$  selection (ES scenario), and
- ranking selection (in GA and ES scenario conceivable).

In fitness-proportional selection – a mating selection method – each individual  $i$  (with fitness  $f_i$ ) is given a probability  $p_i = c_i / \sum_i c_i$  with  $c_i = 1 - f_i$  for being able to pass offspring into the next generation.

The  $(\mu, \lambda)$  selection is due to Schwefel [131] and others and originally used in ES-algorithms.<sup>4</sup>  $\mu$  parents are allowed to breed  $\lambda$  offspring, out of which the best  $\mu$  are used as parents for the next generation. A variant of that method is the  $(\mu + \lambda)$  selection [124], where offspring and parents participate in the selection. Both methods belong to the overproduction selections.

In ranking selection the individuals of a population are sorted according to their fitness. Then, the selection probability is assigned to an individual as a function of its rank.

Another important selection mechanism, *tournament selection*, does not belong to generational selection algorithms. Instead, it is based on competition within only a (usually small) subset of the population. The number of individuals taking part in the tournament is selected randomly according to the *tournament size*. In the smallest possible tournament, two individuals compete. The better individuals (the winners) are subject to genetic operators and replace the losers of the tournament. A higher selection pressure can be adjusted by a larger tournament size.

## 3.5 Basic GP Algorithms: Generational vs. Steady-State

In principle there are two basic GP algorithms implementing the evolutionary loop: a *steady-state* approach and a *generational* approach. Both terms relate to the selection algorithm used. Generational GP works explicitly on generations and employs a generational selection algorithm. Generations are well-defined by the number of executed evolution cycles, which consists of the core steps: fitness evaluation, selection, and variation. At the end of each cycle a new generation is created from its predecessor generation and replaces the older one. Steady-state GP, however, causes a continuous change within a population. It is based on tournament selection and waives explicit generations.

---

<sup>4</sup>To be more precise, this method originates in population genetics under the term truncation selection, where it was used by biologists since the early 1970s.

Nevertheless, the progress in the evolutionary process is measured also in generations. Steady-state generations are usually the intervals in which the fitness is evaluated for the same number of individuals as the population size.

The Figures 3.8(a) and 3.8(b) illustrate the features of generational and steady-state GP algorithms. The grey boxes mark the part of the algorithms in which a (completely or partially) new population is created. It comprises especially the selection and the variation mechanism.

### 3.6 Introns and Neutrality

Those parts of a program which are semantically redundant, that is, which do not directly have a meaning and an effect on the program's output, are designated as *introns*. An example of an intron is the instruction  $x = x$  or the identity function, respectively. In biology introns are well-known as regions on a gene (in eucaryotic DNA), which do not code for proteins and have no functions at all regarding the regulation of genes. Whether introns have further hidden functions is unknown.

Despite their uselessness for the single individual and its fitness, introns seem to be of great importance for the evolutionary process as they allow for *neutral* mutations: Inserting or removing introns has no effect, or rather has a neutral effect on the behavior of the individual. However, in view of evolution this seems to be far from neutrality. It can be shown that introns have a beneficial effect on the global and structural protection of *exons* which are the working code affecting the fitness of an individual, against destructive crossover [114].

Another source for *neutrality* are functional redundancies, i. e., two or more different genotypes mapping to the same phenotype. An example of redundancy in quantum programs is given in Figure 2.12, where two functionally identical quantum circuits are illustrated. A genetic transformation from one genotype to another, both representing the same phenotype, is neutral.

Another form of neutrality is explored in [170]. In contrast to the above described *implicit* neutrality, *explicit* neutrality is provided by an additional structure, which allows activation and deactivation of certain parts or "genes" of the program. Genetic changes on inactive genes have neutral effect, while genetic changes on active genes are exposed to the selection mechanism. Results of the evolution of classical circuits solving a certain Boolean benchmark problem (even-3-parity) indicate a positive effect of neutrality on evolvability.

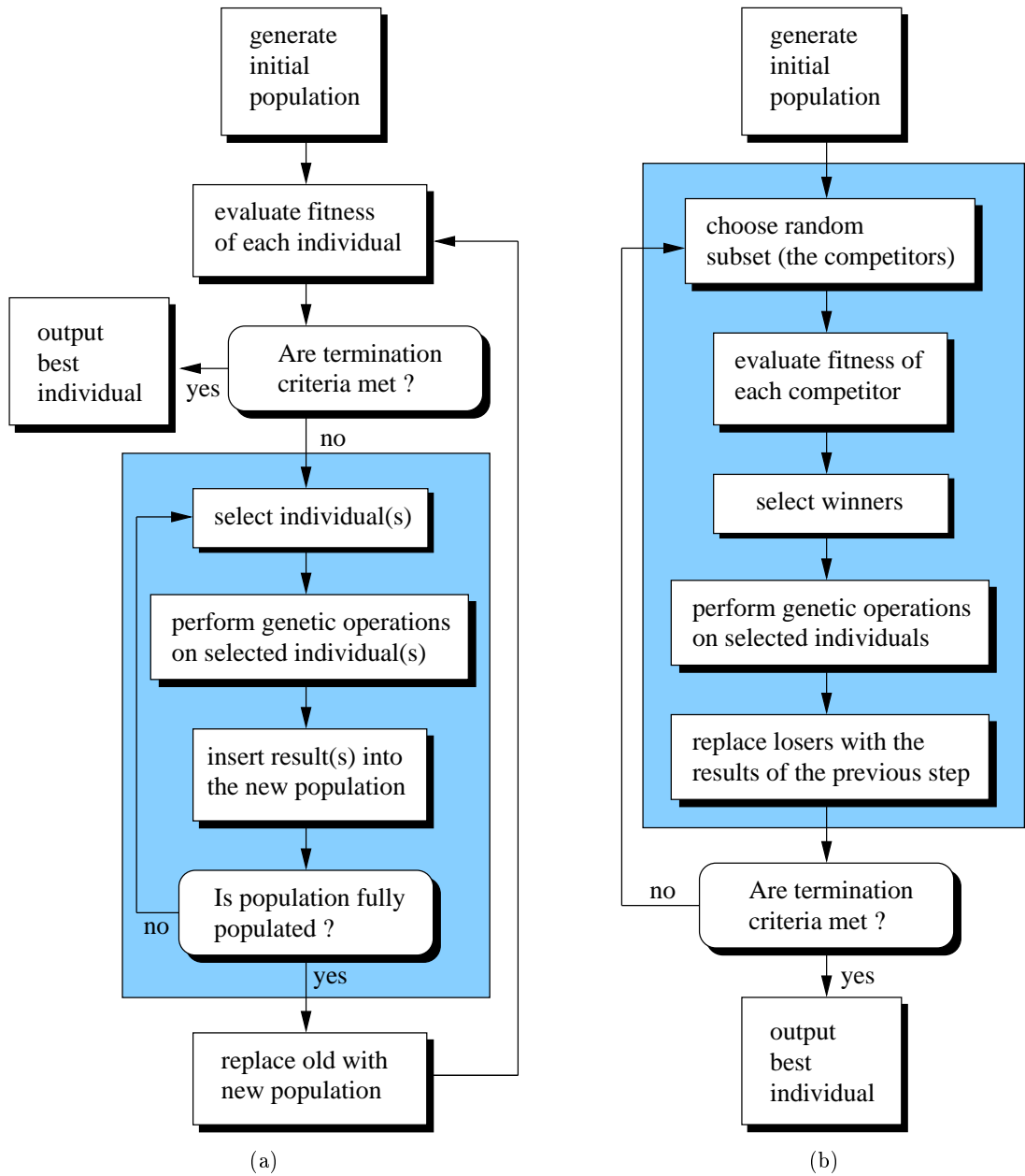


Figure 3.8: Schematic view of (a) the generational GP algorithm and (b) the steady-state GP algorithm.

## 4 Quantum Algorithms and their Classification

*Do not take the lecture too seriously. . . just relax and enjoy it. I am going to tell you what nature behaves like. If you will simply admit that maybe she does behave like this, you will find her a delightful, entrancing thing. Do not keep saying to yourself "But how can it be like that?" because you will get [. . .] into a blind alley from which nobody has yet escaped. Nobody knows how it can be like that.*

---

Richard P. Feynman (1918–1988)

Quantum algorithms consist of a fixed number of computational steps, represented by quantum gates which are comparable to single, elementary instructions in classical computer programming. More complex programming instructions and constructs known from classical programming languages like while-loops which lead to a variable number of steps, are not implementable in the (non-uniform) quantum circuit model.<sup>1</sup> *Hybrid* algorithms remedy this deficiency. They are made of a classical program with quantum subprograms, built by elementary quantum operations. Most of the quantum algorithms (relevant in practice) are hybrid. However, this chapter focuses mainly on quantum subprograms.

It is the aim of this chapter to provide a summary of quantum algorithms which provide an advantage over known classical algorithms and which are based on the quantum circuit model of computation. Although there are not many different quantum algorithms or quantum subprograms respectively, there is no claim to completeness because not all improvements and implementations by means of using other gate sets are mentioned. Furthermore, one must distinguish between the quantum algorithms and the

---

<sup>1</sup>In uniform computation models like the Quantum Turing Machine loops are implementable [17].

problems they are applied to since some algorithms (subprograms) solve more than just one problem or even class of problems. Finally, it cannot be ruled out that there are some more insignificant<sup>2</sup> quantum algorithms which escaped thorough investigation as it stands at the mid-year of 2003. It should be noted that all these quantum algorithms were exclusively developed by hand.

Since there exist only a few quantum algorithms, classifying them may tend to be arbitrary. However, there are characteristics which can lead to partitions of quantum algorithms. One possible classification depends on the technique that the algorithms use: The first class consists of all algorithms which are based on determining a common property of all the output values. A representative of this class is Shor's algorithm, where the period of the function  $f(x) = a^x \bmod N$  is calculated<sup>3</sup>. The second class contains those algorithms which use *amplitude amplification*, like Grover's algorithm. The third class contains algorithms which derive benefit from both methods characterizing the previous two groups, such as the approximate counting algorithm [24].

In a slightly different way, Nielsen and Chuang divide quantum algorithms into three classes. The first class consists of those algorithms based on quantum versions of the Fourier transform. Examples are the Deutsch-Jozsa algorithm as well as Shor's algorithms for factoring and the discrete logarithm. Most of these problems are instances of the hidden subgroup problem in group theory. The second class of quantum algorithms are the quantum search algorithms. At last, the third class of algorithms includes *quantum simulations*, i.e. algorithms, which simulate a quantum physical system on a quantum computer.

A classification based on the programming technique seems tempting. However, the preparation of an equally weighted superposition which can also be regarded as an application of the quantum Fourier transform (QFT), seems<sup>3</sup> to be an imperative step in any quantum algorithm and cannot serve as a distinguishing mark. Whether this can be interpreted as a Fourier transform depends on the problem. The best example is Grover's algorithm, where there are no reasons to read the superpositioning as a QFT. Deutsch's problem, however, can also be regarded as a group theoretic hidden subgroup problem. Therefore, the interpretation of the superposition step as a QFT might be appropriate.

Amplitude amplification means to increase the amplitudes of solution states while decreasing the amplitudes of non-solution states by using a generalization of Grover's iteration. This characterizes many quantum search algorithms. Hogg's algorithm for structured combinatorial search problems, such as highly constrained k-SAT, is an exception, since it is also a quantum search algorithm, but does not use Grover's technique. Instead, its power seems to come from a clever exploitation of the hidden problem structure. The algorithm is described in more detail in Section 4.2.5. Another algorithm,

---

<sup>2</sup>Insignificant in this context means that these quantum algorithms do not have such a big impact like Shor's and Grover's and, therefore did not attract much attention.

<sup>3</sup>The period of a function  $f : 0, 1^n \rightarrow 0, 1^m$  is the value  $r$ ,  $1 \leq r \leq 2^n$ , with  $f(x) = f(x + rm)$ , where  $m$  is an integer, such that  $x, x + rm \in 0, 1, \dots, 2^n - 1$ .

which does not seem to be integrable into these classifications mentioned above is the route finding algorithm by [106]. It finds routes between specified start and end nodes in an undirected, weighted graph which contains no edges, connecting a node to itself, and no multiple edges between the same two nodes. It is mainly based on the idea of superpositional graphs and the realization of a special quantum *AND* operator, which is iteratively applied until an entangled state is generated, which, when measured, collapses to a state representing a path from start to end. A detailed description is omitted.

In Section 4.1 the quantum Fourier transform and the quantum algorithms mainly based on it are represented. Quantum search algorithms, namely Grover's algorithm, other algorithms based on it, including the approximate counting algorithm, and Hogg's algorithm are explained in Section 4.2. Quantum mechanical simulations are briefly discussed in Section 4.3. Finally, Section 4.4 sums up some known limits of quantum computing.

## 4.1 The Quantum Fourier Transform and Algorithms Based on It

### 4.1.1 From DFT to QFT

The classical discrete Fourier transform (DFT) on  $N$  input values  $x_j$ ,  $0 \leq j \leq N - 1$ , is defined as

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \omega_N^{jk}, 0 \leq k \leq N - 1, \quad (4.1)$$

with  $\omega_N := e^{2\pi i/N}$  denoting the  $N^{\text{th}}$  root of unity. Its inverse transformation is  $x_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} y_k \omega_N^{-jk}$ .<sup>4</sup>

Considering the different notations the quantum Fourier transform (QFT) corresponds exactly to the classical DFT in Equation 4.1. Here, the QFT is defined<sup>5</sup> to be the linear operator with the following action on an arbitrary  $n$ -qubit state:

$$\sum_{j=0}^{N-1} x_j |j\rangle \xrightarrow{QFT} \sum_{k=0}^{N-1} y_k |k\rangle$$

---

<sup>4</sup>In some quotations the coefficient is  $1/N$  instead of  $1/\sqrt{N}$ . In this case, the inverse transformation has no factor  $1/\sqrt{N}$ . If the  $x_j$  are values of a function  $f$  at some sampling points, it is often written  $f_j$  instead of  $x_j$  and  $\hat{f}_k$  instead of  $y_k$ .

<sup>5</sup>In literature the definition of the Fourier transform is not consistent. The DFT and its inverse transform are interchanged. In most original papers (e.g. [132]) the QFT is defined as above. However, both the classical Fourier transform and its inverse transform can be deduced from the general form of a Fourier transform (Equation 4.3).

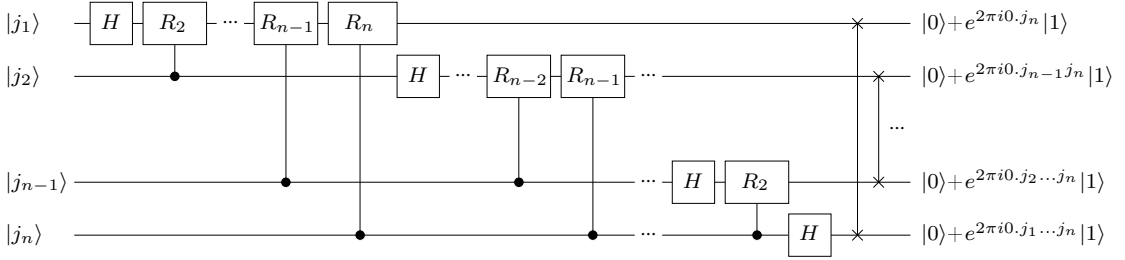


Figure 4.1: A quantum circuit for QFT.

with  $N = 2^n$ . The  $y_k$  are the Fourier transforms of the amplitudes  $x_j$  as defined in Equation 4.1. In product representation, the QFT looks like

$$|j_1, \dots, j_n\rangle \rightarrow \frac{(|0\rangle + e^{2\pi i 0 \cdot j_n} |1\rangle)(|0\rangle + e^{2\pi i 0 \cdot j_{n-1} j_n} |1\rangle) \dots (|0\rangle + e^{2\pi i 0 \cdot j_1 j_2 \dots j_n} |1\rangle)}{2^{n/2}}. \quad (4.2)$$

This leads to an efficient circuit for the QFT [107] which is shown in Figure 4.1. The gate  $R_k$  corresponds to the phase gate  $PH(2\pi/2^k)$  (cf. Section 2.4.1). At the end of the circuit, swap gates reverse the order of the qubits to obtain the desired output. The circuit uses  $n(n-1)/2$  Hadamard and conditional phase gates and at most  $n/2$  swap gates in addition. Therefore, this  $n$ -qubit QFT circuit has a  $\Theta(n^2) = \Theta(\log^2 N)$  runtime. Of course,  $\text{QFT}^{-1}$  can be performed in  $\Theta(n^2)$  steps, too. In contrast, the best classical algorithms (like the Fast Fourier Transform) need  $\Theta(n2^n)$  classical gates for computing the DFT on  $2^n$  elements. That is, the speedup of the QFT compared with the best classical algorithms is exponential. Note that the size of the circuit can be reduced to  $O(n \text{ poly}(\log n))$  by neglecting the phase shifts of the first few bits as this amounts only to an “acceptable” small error. The best known approximate quantum Fourier transform, described in [66], requires only  $O(n \log n)$ .

Nevertheless, there are two major problems concerning the use of the QFT: First, it is not known how to efficiently prepare any original state to be Fourier transformed, and second, the Fourier transformed amplitudes cannot be directly accessed by measurement.

### 4.1.2 Phase Estimation

An important application of the QFT is the *phase estimation* on which in turn many other applications are based. Given a unitary operator  $U$  with eigenvector (or eigenstate)  $|u\rangle$ , the phase estimation problem is to estimate the value  $\phi$  of the eigenvalue  $e^{2\pi i \phi}$  ( $U|u\rangle = e^{2\pi i \phi}|u\rangle$ ). To perform the estimation it is assumed that black-boxes (so-called oracles) are available which prepare the state  $|u\rangle$  and perform the controlled- $U^{2^l}$  operation for integers  $l > 0$ . Figure 4.2 shows the phase estimation procedure. It works on two registers, a  $n$ -qubit register initially in state  $|0\rangle$  ( $n$  depends on the number of digits of



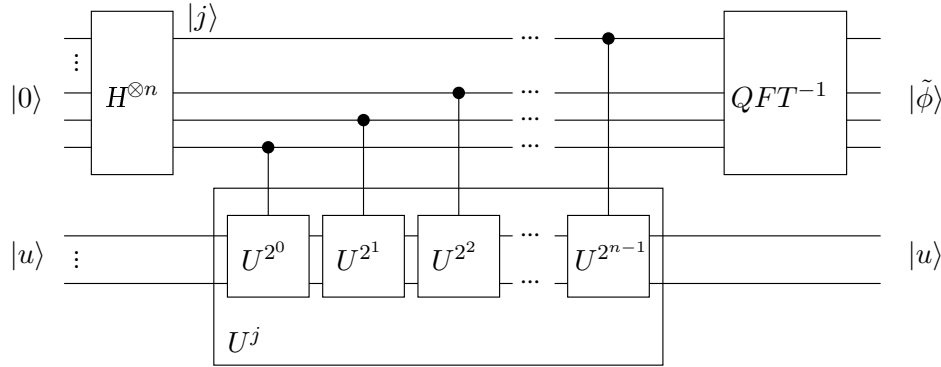


Figure 4.2: A quantum circuit for the phase estimation procedure.

accuracy in the estimate for  $\phi$  and the probability that the procedure is successful) and a register in the initial state  $|u\rangle$ . The circuit begins by applying the  $n$ -qubit Hadamard transform  $H^{\otimes n}$  to the first quantum register, followed by the application of controlled  $U$ -operations on the second register, with  $U$  raised to successive powers of two. This sequence maps

$$|0\rangle|u\rangle \longrightarrow \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle U^j |u\rangle = \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} e^{2\pi i \phi j} |j\rangle |u\rangle.$$

With  $\phi = 0.\phi_1 \dots \phi_n$  this state may be rewritten in product form which is equivalent to the product representation (Equation 4.2). If  $\phi$  is exactly expressed in  $n$  qubits the application of the inverse QFT to the first qubit register leads to the state  $|\phi\rangle = |\phi_1 \dots \phi_n\rangle$ , or a good estimator  $|\tilde{\phi}\rangle$  otherwise. Computing  $\text{QFT}^{-1}$  can be seen as the actual *phase estimation step*.

A final measurement of the first register yields to the result  $\phi$  or  $\tilde{\phi}$  respectively. To estimate  $\phi$  with  $k$ -bit accuracy and probability of success at least  $1 - \epsilon$ , choose  $n = k + \lceil \log(2 + \frac{1}{2\epsilon}) \rceil$ . This algorithm, summarized below, computes the approximation  $\tilde{\phi}_u$  to  $\phi_u$  using  $O(n^2)$  operations and one query to the controlled- $U^j$  oracle gate.

Even if the eigenvector  $|u\rangle$  is unknown and cannot be prepared, running the phase estimation algorithm on an arbitrary state  $|\psi\rangle = \sum_u c_u |u\rangle$  (written in terms of eigenstates  $|u\rangle$ ) yields a state  $\sum_u c_u |\tilde{\phi}_u\rangle |u\rangle$ , where  $\tilde{\phi}_u$  is a good approximation to  $\phi_u$ . Assuming that  $n$  is chosen as specified above, the probability for measuring  $\phi_u$  with  $k$ -bit accuracy is at least  $|c_u|^2(1 - \epsilon)$ .

---

**Quantum Algorithm 4.1.1:** *Quantum phase estimation*

---

1. Create the equally weighted superposition:

$$|0\rangle|u\rangle \xrightarrow{H^{\otimes n}} \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle|u\rangle$$

2. Apply the quantum oracle:

$$\xrightarrow{U^j} \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle U^j |u\rangle = \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} e^{2\pi i \phi j} |j\rangle |u\rangle$$

3. Calculate the inverse quantum Fourier transform and measure the first register:

$$\xrightarrow{QFT^{-1}} |\tilde{\phi}\rangle |u\rangle \xrightarrow{M_1} \tilde{\phi}$$


---

### 4.1.3 Order-Finding and Other Applications

The *order-finding problem* reads as follows: for  $x, N \in \mathbb{N}$ ,  $x < N$ ,  $\gcd(x, N) = 1$ , determine the least  $r \in \mathbb{N}$ , such that  $x^r = 1 \pmod{N}$ . It is believed to be a hard problem on classical computers. Let  $n = \lceil \log N \rceil$  be the number of bits needed to specify  $N$ . The quantum algorithm for order-finding is just the phase estimation algorithm applied to the unitary operator  $U|y\rangle \equiv |xy \pmod{N}\rangle$  with  $y \in \{0, 1\}^n$  and  $|u\rangle = |1\rangle$ , a superposition of eigenstates of  $U$  (for  $N \leq y \leq 2^n - 1$ , define  $xy \pmod{N} := y$ ).

The entire sequence of controlled- $U^{2^t}$  operations can be implemented efficiently using *modular exponentiation*: It is

$$\begin{aligned} |z\rangle U^z |y\rangle &= |z\rangle U^{z_{t-1}2^{t-1}} \dots U^{z_0 2^0} |y\rangle \\ &= |z\rangle |(x^{z_{t-1}2^{t-1}} \times \dots \times x^{z_0 2^0})y \pmod{N}\rangle = |z\rangle |x^z y \pmod{N}\rangle. \end{aligned}$$

In a first step modular multiplication is used to compute successively (by squaring)  $x^{2^j} \pmod{N}$  from  $x^{2^{j-1}}$  for all  $j = 1 \dots t-1$ . In a second step  $x^z \pmod{N}$  is calculated by multiplying the  $t$  terms  $(x^{z_{t-1}2^{t-1}} \pmod{N}) \dots (x^{z_0 2^0} \pmod{N})$ . On the basis of this procedure the construction of the quantum circuit computing  $U^z : |z\rangle |y\rangle \longrightarrow |z\rangle |x^z y \pmod{N}\rangle$  is straightforward, using  $O(n^3)$  gates in total.

To perform the phase estimation algorithm, an eigenstate of  $U$  with a nontrivial eigenvalue or a superposition of such eigenstates has to be prepared. The eigenstates of  $U$

are

$$|u_s\rangle \equiv \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \omega_r^{-sk} |x^k \bmod N\rangle,$$

for  $0 \leq s \leq r-1$ . Since  $\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle$  it is sufficient to choose  $|u\rangle = |1\rangle$ .

Now, applying the phase estimation algorithm on  $t = 2n + 1 + \lceil \log(2 + 1/2\epsilon) \rceil$  qubits in the first register and a second quantum register prepared to  $|1\rangle$  leads with a probability of at least  $(1 - \epsilon)/r$  to an estimate of the phase  $\phi \approx s/r$  with  $2n + 1$  bits accuracy. From this result, the order  $r$  can be calculated classically using an algorithm, known as the *continued fraction expansion*. It efficiently computes the nearest fraction of two bounded integers to  $\phi$ , i.e. two integers  $s', t'$  with no common factor, such that  $s'/t' = s/r$ . Under certain conditions this classical algorithm can fail, but there are other methods to circumvent this problem [107].

An application of the order-finding quantum subroutine is Shor's factorization algorithm [132, 133]. Other problems which can be solved using the order-finding algorithm are period-finding and the discrete logarithm problem. These and some other problems can be considered in a more general context, which will be explained in the following two subsections.

---

**Quantum Algorithm 4.1.2: Order-finding**

---

1. Create superposition:

$$|0\rangle|1\rangle \xrightarrow{H^{\otimes n}} \frac{1}{2^{t/2}} \sum_{j=0}^{2^t-1} |j\rangle|1\rangle$$

2. Apply the black-box oracle  $U_{x,N} : |j\rangle|k\rangle \rightarrow |x^j k \bmod N\rangle$ , with  $x$  co-prime to  $N$ :

$$\xrightarrow{U_{x,N}} \frac{1}{2^{t/2}} \sum_{j=0}^{2^t-1} |j\rangle|x^j \bmod N\rangle \approx \frac{1}{2^{t/2}\sqrt{r}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j/r} |j\rangle|u_s\rangle$$

3. Calculate the inverse QFT of the first register and measure this:

$$\xrightarrow{QFT^{-1}} \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |s/r\rangle|u_s\rangle \xrightarrow{M_1} |s/r\rangle$$


---

#### 4.1.4 Fourier Transform on Arbitrary Groups

The more general definition of the Fourier transform  $\text{FT}_G$  on an arbitrary group  $G$  needs some background in algebra and in representation theory over finite groups. See chapter 2 in [69] or appendix 2 in [107] for a brief introduction in the required mathematics.

Let  $G$  be a finite group of order  $N$  and  $f : G \rightarrow \mathbb{C}$ . The *Fourier transform of  $f$  at the irreducible representation*<sup>6</sup>  $\rho$  of  $G$  (denoted  $\hat{f}(\rho)$ ) is defined to be the  $d_\rho \times d_\rho$  matrix

$$\hat{f}(\rho) = \sqrt{\frac{d_\rho}{N}} \sum_{g \in G} f(g) \rho(g). \quad (4.3)$$

The collection of matrices  $\langle \hat{f}(\rho) \rangle_{\rho \in \hat{G}}$  is designated as the *Fourier transform of  $f$* . Thus, the Fourier transform maps  $f$  into  $|\hat{G}|$  matrices of varying dimensions which have totally  $\sum_\rho d_\rho^2 = |G|$  entries. This means that the  $|G|$  complex numbers  $\langle f(g) \rangle_{g \in G}$  are mapped into  $|G|$  complex numbers organized into matrices. Furthermore, the Fourier transform is linear in  $f$  ( $\widehat{f_1 + f_2}(\rho) = \hat{f}_1(\rho) + \hat{f}_2(\rho)$ ) and  $\hat{f}(\rho)$  is unitary for all  $\rho \in \hat{G}$ .

If  $G$  is a finite Abelian group, all irreducible representations have dimension one. Thus, the representations correspond to the  $|G|$  (irreducible) *characters*  $\chi : G \rightarrow \mathbb{C}^\times$  of  $G$ . It is  $(G, +) \simeq (\hat{G}, \cdot)$ . This is quite useful to simplify the notation by choosing an isomorphism  $g \mapsto \chi_g$ .

With the operation  $(\chi_1 \cdot \chi_2)(g) = \chi_1(g) \cdot \chi_2(g)$  the set  $\hat{G}$  of all characters of  $G$  is an Abelian group, called the *dual group of  $G$* . Any value  $\chi(g)$  is a  $|G|^{\text{th}}$  root of unity, i. e.  $\chi(g)^{|G|} = 1$ . For instance, if  $G = \mathbb{Z}_N$  then the group of characters is the set  $\{\chi_k(j) = \omega_N^{jk} | j, k = 0 \dots N-1\}$  and the Fourier transform corresponds to the classical DFT. For  $G = \mathbb{Z}_2^n$  the Fourier transform  $\text{FT}_{\mathbb{Z}_2^n}$  corresponds to the Hadamard transform  $H^{\otimes n}$ .

By the structure theorem of finite Abelian groups,  $G$  is isomorphic to products of cyclic groups  $\mathbb{Z}_{p_i}$  of prime power order with addition modulo  $p_i$  being the group operation, i. e.  $G \simeq \mathbb{Z}_{p_1} \times \dots \times \mathbb{Z}_{p_m}$ . Any  $g \in G$  can be written equivalently as  $(g_1, \dots, g_m)$ , where  $g_i \in \mathbb{Z}_{p_i}$ . This naturally leads<sup>7</sup> to the description of the characters  $\chi^G$  on  $G$  as the product of the characters  $\chi^{\mathbb{Z}_{p_i}}$  on  $\mathbb{Z}_{p_i}$ . It is

$$\chi_h^G(g) = \prod_{i=1}^m \chi_{h_i}^{\mathbb{Z}_{p_i}}(g_i) = \prod_{i=1}^m e^{2\pi i g_i h_i / p_i} = e^{2\pi i (\sum_i g_i h_i q_i)},$$

---

<sup>6</sup>A representation  $\rho$  of a finite group  $G$  is a homomorphism  $\rho : G \rightarrow GL(V)$ , where  $V$  is a  $\mathbb{C}$ -vector space. The dimension of  $V$  is called the *dimension of the representation  $\rho$* , denoted  $d_\rho$ .  $\rho$  is said to be *irreducible*, if no other subspaces  $W$  are  $G$ -invariant, i. e.  $\rho(g)W \subseteq W, \forall g \in G$ , except 0 and  $V$ . Up to isomorphism, a finite group has a finite number of irreducible representations.  $\hat{G}$  denotes such a set of irreducible representations, which is a complete system of representatives of all isomorphism classes.

<sup>7</sup>The following mapping is used:  $(g_1, \dots, g_m) \mapsto \prod_{i=1}^m \chi_{h_i}^{\mathbb{Z}_{p_i}}(g_i)$

with  $h = (h_1, \dots, h_m)$  and  $q_i := N/p_i$ . Note that  $\chi_h(g) = \chi_g(h)$ .

With Equation 4.3, the Fourier transform of a function  $f : G \rightarrow \mathbb{C}$  may be written as

$$\hat{f}(h) = \frac{1}{\sqrt{N}} \sum_{g \in G} f(g) \chi_g(h),$$

with  $h \in G$ . Using the conventional notation, the QFT on an Abelian group  $G$  acts on the basis states as follows:

$$QFT_G |h\rangle = \frac{1}{\sqrt{|G|}} \sum_{g \in G} \chi_g(h) |g\rangle.$$

### 4.1.5 The Hidden Subgroup Problem

Nearly all known problems that have a quantum algorithm which provides an exponential speedup over the best known classical algorithm can be formulated as a *hidden subgroup problem* (HSP). Problem instances are for example order-finding, period-finding and discrete logarithm [80, 105]. The HSP is:

Let  $G$  be a finitely generated group,  $H < G$  a subgroup,  $X$  a finite set and  $f : G \rightarrow X$  a function such that  $f$  is constant on cosets  $\bar{g} = gH \in G/H$  and takes distinct values on distinct cosets, i. e.  $f(\bar{g}_1) \neq f(\bar{g}_2)$  for  $\bar{g}_1 \neq \bar{g}_2$ . Find a generating set for  $H$ !

Define  $H^\circ := \{g \in G : \chi_g(h) = 1, \forall h \in H\}$ , also called the *annihilator group of  $H$* . Note that  $H^\circ$  is equivalent to the perp subgroup  $H^\perp = \{\chi : H \subseteq \ker \chi\}$ , which in turn is isomorphic to the dual of  $G/H$ . The algorithm for solving the HSP uses the following two properties of the Fourier transform over  $G$  [67, 158]:

- The FT of the *convolution*  $*$  of two vectors is the pointwise product  $\cdot$  of the FT of each vector:

$$QFT\left(\sum_k \alpha_k |k\rangle * \sum_l \beta_l |l\rangle\right) = \sqrt{|G|} \left[ QFT\left(\sum_k \alpha_k |k\rangle\right) \cdot QFT\left(\sum_l \beta_l |l\rangle\right) \right]. \quad (4.4)$$

- The superposition state, uniform on  $H$ , is mapped to  $H^\circ$ :

$$\frac{1}{\sqrt{|H|}} \sum_{h \in H} |h\rangle \xrightarrow{QFT} \sqrt{\frac{|H|}{|G|}} \sum_{k \in H^\circ} |k\rangle. \quad (4.5)$$

Using this, it follows for a coset state  $1/\sqrt{|H|} \sum_{h \in H} |g_0 h\rangle$ :

$$\frac{1}{\sqrt{|H|}} \sum_{h \in H} |g_0 h\rangle = |g_0\rangle * \frac{1}{\sqrt{|H|}} \sum_{h \in H} |h\rangle$$

$$\stackrel{QFT_G, \text{Eqs. 4.4, 4.5}}{\rightarrow} \sqrt{\frac{|H|}{|G|}} \left[ \left( \sum_{g \in G} \chi_g(g_0) |g\rangle \right) \cdot \left( \sum_{k \in H^\circ} |k\rangle \right) \right] = \sqrt{\frac{|H|}{|G|}} \sum_{k \in H^\circ} \chi_k(g_0) |k\rangle.$$

So, the QFT takes a coset state to the annihilator group state of the corresponding subgroup where the coset is encoded in the phase of the basis vectors.

Assume that  $G$  is Abelian. A hybrid algorithm to solve the Abelian HSP consists of the following quantum subroutine:

---

**Quantum Algorithm 4.1.3: Abelian HSP**

---

1. Create a random coset state:

$$|0\rangle|0\rangle \stackrel{QFT_{G \otimes I}}{\rightarrow} \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle|0\rangle \xrightarrow{U_f} \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle|f(g)\rangle \xrightarrow{M_2} \frac{1}{\sqrt{|H|}} \sum_{h \in H} |g_0 h\rangle$$

2. Fourier sample the coset state:

$$\frac{1}{\sqrt{|H|}} \sum_{h \in H} |g_0 h\rangle \stackrel{QFT_G}{\rightarrow} \sqrt{\frac{|H|}{|G|}} \sum_{k \in H^\circ} \chi_k(g_0) |k\rangle \xrightarrow{M_1} k_i \in H^\circ$$


---

$M_i$  denotes the measurement of the  $i$ -th quantum register. The process of computing the Fourier transform over a group  $G$  and measuring subsequently is also called *Fourier sampling*. After applying  $M_2$ , it is sufficient to observe the first register. The last measurement results in one out of  $|G|/|H|$  elements  $k_i \in H^\circ$  with probability  $\frac{|H|}{|G|}$ . Repeating this process  $t = \text{poly}(\log |G|)$  times [67] leads to a set of elements which describe the  $\chi_i \in H^\perp$ . Thus,  $H$  can be classically computed as the intersection of the kernels of  $\chi_i$ :  $H = \bigcap_{i=0}^{t-1} \ker \chi_i$ . The efficient quantum circuit for the Abelian HSP is shown in Figure 4.3.

Generalizations of the Abelian HSP quantum algorithm to the non-Abelian case have been attempted by many authors [126, 49, 50, 67, 69, 59, 79], unfortunately only with limited success. Up to now, the problem is still open, except for only a few particular instances.

#### 4.1.6 A coarse outline on QFT-based algorithms

Table 4.1 shows an overview on problems efficiently solvable by means of quantum algorithms based on the QFT. The following explanations and annotations refer to single problems given below in the table.

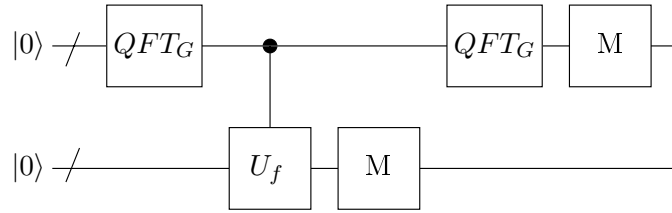


Figure 4.3: A quantum circuit for the Abelian HSP.

**Abelian group stabilizer problem:** Let  $G$  be an Abelian group acting on a set  $X$ . Find the stabilizer  $G_x := \{g \in G \mid g \cdot x = x\}$  for  $x \in X$ .

**Decomposing finite Abelian groups:** Any finite Abelian group  $G$  is isomorphic to a product of cyclic groups. Find such a decomposition. For many groups no classical algorithm is known which performs this task efficiently.

The quantum algorithm described in [30] mainly uses the quantum algorithm for HSP to solve a partial problem. The rest is done classically.

An application of this algorithm is the efficient computing of class numbers (assuming the Generalized Riemann Hypothesis) [162].

**Deutsch's problem:** Determine whether a black-box binary function  $f : \{0, 1\} \rightarrow \{0, 1\}$  is constant (or balanced) (cf. Section 2.8.1).

**Hidden linear function problem:** Let  $f : \mathbb{Z}^k \rightarrow S$  be a function for an arbitrary range  $S$  with  $f(x_1, \dots, x_k) = h(x_1 + \alpha_2 x_2 + \dots + \alpha_k x_k)$  for a function  $h$  with period  $q$  and  $\alpha_i \in \mathbb{Z}$ . Recover the values of all the  $\alpha_i \pmod{q}$  from an oracle for  $f$ . This problem is an instance of the HSP.

**Inner product problem (Bernstein-Vazirani):** For  $a \in \{0, 1\}^n$ , let  $f_a : \{0, 1\}^n \rightarrow \{0, 1\}$  be defined by  $f_a(x) = a \cdot x$ . Calculate  $a$ ! This problem is not directly an instance of the HSP. Nevertheless, Fourier sampling helps finding a solution, too.

**Orders of finite solvable groups:** The problem is described in [162].

**Pell's Equation:** Given a positive non-square integer  $d$ , Pell's equation is  $x^2 - dy^2 = 1$ . Find integer solutions. The quantum algorithm developed in [68] calculates the regulator of the ring  $\mathbb{Z}[\sqrt{d}]$ , which is a closely related problem. The necessary background in computational algebraic number theory can be found in [32]. The quantum step in this algorithm is a procedure to efficiently approximate the irrational period  $S$  of a function in time polynomial in  $\ln S$ .

**Principal ideal problem:** Given an ideal  $I$ , determine (if existing) an  $\alpha \in \mathbb{Q}(\sqrt{d})$  such that  $I = \alpha\mathbb{Z}[\sqrt{d}]$ . The algorithm reduces to a discrete log type problem.

**Shifted Legendre symbol problem (SLSP):** Given a function  $f_s$  and an odd prime  $p$  such that  $f_s(x) = \left(\frac{x+s}{p}\right)$ , for all  $x \in \mathbb{Z}_p$ , find  $s$ ! Variants of this problem are the *shifted Jacobi symbol problem* and the shifted version of the quadratic character  $\chi$  over finite fields  $\mathbb{F}_q$  (*shifted quadratic character problem*). The classical complexities of these problems are unknown.

**Simon's problem:** Let  $a \in \{0, 1\}^n$  and  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  a function with  $f(x \oplus a) = f(x)$  ( $\oplus$ : bitwise xor). Calculate  $a$ ! Simon's problem was the first that was shown to have an expected polynomial time quantum algorithm but no polynomial time randomized algorithm [134]. Brassard and Høyer devised in [23] an exact quantum polynomial-time algorithm to solve this problem.<sup>8</sup>

Two other transformations which can be recovered from the DFT are the *discrete cosine transformation* and the *discrete sine transformation*. In [86] Klappenecker and Rötteler show that both transformations of size  $N \times N$  and types I-IV can be realized in  $O(\log^2 N)$  operations on a quantum computer instead of  $O(N \log N)$  on a classical computer. Another class of unitary transforms, the wavelet transforms, are efficiently implementable on a quantum computer, as shown in [52]. For a certain wavelet transform, a quantum algorithm is designed using the QFT.

## 4.2 Quantum Search Algorithms

Search problems are well-known and intensively investigated in computer science. Generally spoken, a search problem is to find one or more elements in a (finite or infinite, structured or unstructured) search space, which meet certain properties.

Suppose, a large database contains  $N \gg 1$  items in a random order. On a classical computer it takes  $O(N)$  comparisons to determine the item searched for. However, there is a quantum search algorithm, also called *Grover's algorithm*, which requires only  $O(\sqrt{N})$  operations. Grover's algorithm is optimal for unstructured search problems.

Also in some cases of structured search spaces quantum algorithms can do better than classical as demonstrated by Hogg's algorithm for 1-SAT and highly constrained  $k$ -SAT.

### 4.2.1 Grover's Algorithm

It is assumed that  $N = 2^n$ . The database is represented by a function  $f$  which takes as input an integer  $x$ ,  $0 \leq x \leq N - 1$  (the database index), with  $f(x) = 1$  if  $x$  is a solution to the search problem and  $f(x) = 0$  otherwise. Let  $\mathcal{L}$  be the set of solutions, i.e.,  $\mathcal{L} = \{x | f(x) = 1, x \in \{0, \dots, N - 1\}\}$  and  $M = |\mathcal{L}|$  the number of solutions.

---

<sup>8</sup>For that purpose they used a generalized version of the Grover operator (see Section 4.2.1)



Problem	Runtime	References
(discrete) QFT	$\Theta(n^2)$ or $O(n \log n)$ resp.	[35, 132] [66]
Deutsch <sup>†</sup>	1 oracle query	[43, 31]
Deutsch-Jozsa <sup>†</sup>	1 oracle query	[46, 31]
Bernstein-Vazirani	1 oracle query	[17]
Simon <sup>†</sup>	$O(n)$ repet. with 1 oracle query each	[134, 135, 23]
Period-finding <sup>†</sup> f with $f(x+r) = f(x)$ , $x, r \in \mathbb{N}_0$ , $0 < r < 2^n$ , a periodic function, output r!	1 oracle query, $O(n^2)$ operations	[107]
Phase estimation	$O(n^2) + 1$ oracle query	[107]
Order-finding <sup>†</sup>	$O(n^3)$	[132, 133]
Factoring <sup>†</sup>	$O(n^3)$ operations, $O(n^2 \log n \log \log n)$	[132] [133]
Discrete logarithm <sup>†</sup> given: $a, b = a^s \bmod N$ ; determine $s$	polyn. time QA	[133]
Hidden linear function problems <sup>†</sup>	polyn. time QA	[19]
Abelian stabilizer <sup>†</sup>	polyn. time QA	[85]
Shifted Legendre symbol problem and variants	polyn. time QA	[153]
Computing orders of finite solvable groups	polyn. time QA	[162, 79]
Decomposing Finite Abelian Groups	polyn. time QA	[30]
Pell's Equation & Principal Ideal Problem	polyn. time QA	[68]

Table 4.1: The quantum Fourier transform and algorithms based on it. Problems marked with <sup>†</sup> are special instances of the HSP. The parameter  $n$  is the input length of the given problem.

Furthermore, let be

$$|\alpha\rangle = \frac{1}{\sqrt{N-M}} \sum_{x \notin \mathcal{L}} |x\rangle, \quad |\beta\rangle = \frac{1}{\sqrt{M}} \sum_{x \in \mathcal{L}} |x\rangle$$

and

$$|\phi\rangle = \sqrt{\frac{N-M}{N}} |\alpha\rangle + \sqrt{\frac{M}{N}} |\beta\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle.$$

This algorithm works on two quantum registers  $|x\rangle|y\rangle$ , where  $|x\rangle$  is the index register and  $|y\rangle$  is a single ancilla qubit. Let  $U_\alpha$  be the black-box which computes  $f$ . Then, applied to the state  $1/\sqrt{2}|x\rangle(|0\rangle - |1\rangle)$ , the oracle “marks” all solutions by shifting the phase of each solution (see Section 2.8: computing a function into the phase). Ignoring the single qubit register, the action of  $U_\alpha$  may be written as

$$|x\rangle \longrightarrow (-1)^{f(x)} |x\rangle$$

or

$$U_\alpha = I - 2|\beta\rangle\langle\beta|.$$

Geometrically,  $U_\alpha$  induces a reflection about the vector  $|\alpha\rangle$  in the plane defined by  $|\alpha\rangle$  and  $|\beta\rangle$ . Another important operation, denoted with  $U_\phi$ , is the so-called *inversion about the mean*:

$$U_\phi = H^{\otimes n}(2|0\rangle\langle 0| - I)H^{\otimes n} = 2|\phi\rangle\langle\phi| - I. \quad (4.6)$$

Applying  $U_\phi$  to a general state  $|a\rangle = \sum_k \alpha_k |k\rangle$  leads to

$$U_\phi|a\rangle = 2|s\rangle\langle s|a\rangle - |a\rangle = 2 \sum_k A|k\rangle - \sum_k a_k |k\rangle = \sum_k (2A - a_k)|k\rangle$$

using that  $\langle s|a\rangle = \frac{1}{\sqrt{2^n}} \sum_k a_k = \sqrt{2^n}A$ , where  $A = \frac{1}{2^n} \sum_k a_k$  is the mean of the amplitude. Thus, the amplitudes are transformed as  $U_\phi : a_k \rightarrow 2A - a_k$ , i.e., the coefficient of  $|k\rangle$  is reflected about the mean value of the amplitude. In other words,  $U_\phi$  is a reflection about the vector  $\phi$  in the plane spanned by  $|\alpha\rangle$  and  $|\beta\rangle$ . The combination of both operators  $U_\alpha$  and  $U_\phi$  leads to the Grover operator<sup>9</sup>, defined to be

$$U_G = U_\phi U_\alpha = -H^{\otimes n}(2|0\rangle\langle 0| - I)H^{\otimes n}(2|\beta\rangle\langle\beta| - I) \quad (4.7)$$

(see Figure 4.4). Thus, the product of the two reflections<sup>10</sup>  $U_\alpha$  and  $U_\phi$  is a rotation in the two-dimensional subspace spanned by  $|\alpha\rangle$  and  $|\beta\rangle$  rotating the space by an angle  $\theta$ , defined by  $\sin \theta/2 = \sqrt{M/N}$ , as shown in Figure 4.5.

---

<sup>9</sup>In many papers the Grover operator is called *Grover iteration*. This is misleading, as the iterated application of the Grover operator is in fact the iteration and applying  $U_G$  is just a single step within it.

<sup>10</sup>This interpretation of the Grover operator was first pointed out by [4].

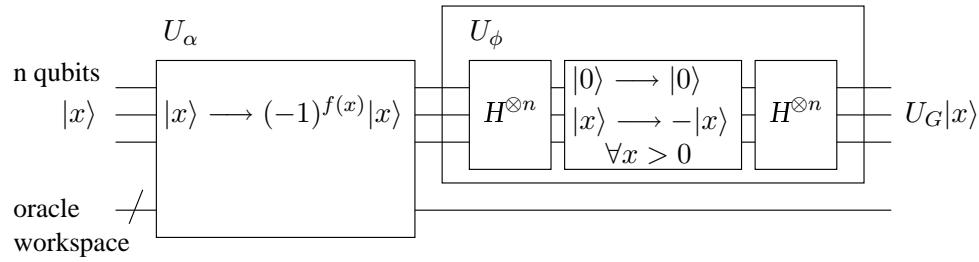


Figure 4.4: Quantum circuit for the Grover operator.

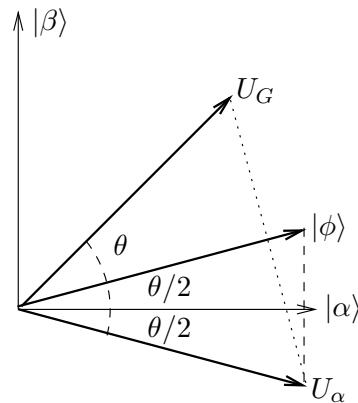


Figure 4.5: Geometric visualization of a single step in the Grover iteration.

Repeating the Grover operator  $T \simeq \pi/4\sqrt{N/M} = O(\sqrt{N/M})$  times rotates the initial system state  $\phi$  close to  $|\beta\rangle$ . Observation of the state in the computational basis yields a solution to the search problem with probability at least  $p \geq 1/2$ . When  $M \ll N$  this probability is at least  $1-M/N$ , that is nearly 1. Figure 4.6 illustrates the entire quantum search algorithm. It should be mentioned that the quantum search algorithm is optimal, i. e. no quantum algorithm can perform the task of searching  $N$  items using fewer than  $\Omega(\sqrt{N})$  accesses to the search oracle [21, 14].

A concluding note: In the basis  $\{|\alpha\rangle, |\beta\rangle\}$  the Grover operator may also be written as

$$U_G = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

where  $0 \leq \theta \leq \pi/2$ , assuming without limitation that  $M \leq N/2$ . Its eigenvalues are  $e^{\pm i\theta}$ . If it is not known whether  $M \leq N/2$ , this can be achieved for certain by adding a single qubit to the search index, doubling the number of items to be searched to  $2N$ . A new augmented oracle ensures that only those items are marked which are solutions and whose extra bit is set to zero.

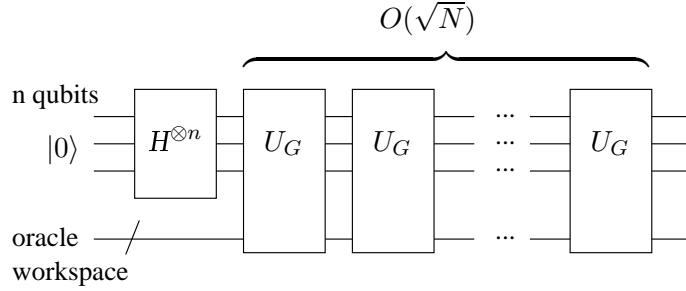


Figure 4.6: Quantum circuit for the quantum search algorithm.

A generalization of the Grover iteration to boost the probability of measuring a solution state is called *amplitude amplification*. According to Gruska [65], there are at least two (only slightly different) methods which meet this condition, one proposed by Grover [63], the other by Brassard et al. [24]. The most general version of a Grover operator is presented by Bihan et al. [18]. All methods have in common, that they use an arbitrary unitary transformation  $U$  instead of the Hadamard transformation  $H^{\otimes n}$  as in the original Grover operator (Equation 4.7). The following transformation might be deemed as the *generalized Grover operator*:

$$U_G = -UI_s^\beta U^{-1}I_f^\gamma,$$

where  $I_s^\beta = I - (1 - e^{i\beta})|s\rangle\langle s|$  is a rotation of a fixed basis state  $|s\rangle$  by an angle  $\beta$  and  $I_f^\gamma = \sum_x e^{i\gamma f(x)}|x\rangle\langle x|$  is a rotation by an arbitrary phase  $\gamma$ .

#### 4.2.2 Quantum Counting: Combining Grover Operator and Phase Estimation

Provided that  $M$  is known, Grover's algorithm can be applied as described above. If  $M$  is not known in advance, it can be determined by applying the phase estimation algorithm to the Grover operator  $U_G$ , estimating one of its phases  $\pm\theta$ . Figure 4.7 shows a quantum circuit for performing approximate quantum counting. From the equation<sup>11</sup>  $\sin^2(\theta/2) = M/2N$  and the estimate for  $\theta$  it follows an estimate for the number of solutions  $M$ . Further analysis shows, the error in this estimate for  $M$  is less than  $(\sqrt{2MN} + \frac{N}{2^{k+1}})2^{-k}$ , provided that  $\theta$  has a  $k$ -bit accuracy. Summarizing, the algorithm requires  $O(\sqrt{N})$  oracle calls to estimate  $M$  to high accuracy. Finding a solution to a search problem when  $M$  is unknown requires to apply both algorithms, first the quantum counting and then the quantum search algorithm. Errors arisen in the estimates for  $\theta$  and  $M$  affect the total probability to find a solution to the search problem. However, the probability can be increased close to 1 by a few repetitions of the combined counting-search algorithm.

<sup>11</sup>The search space is expanded to  $2N$  to ensure that  $M \leq N/2$ . As already described above this is done by adding a single qubit to the search space.

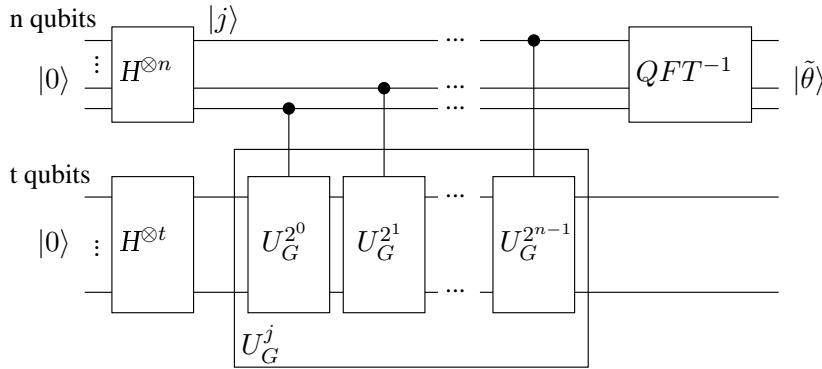


Figure 4.7: Circuit for the quantum counting algorithm, an application of the phase estimation procedure to estimate the eigenvalues of the Grover operator  $U_G$  which enables to determine the number of solutions  $M$  to the search problem. The first register contains  $n$  qubits and the second register contains  $t$  qubits, sufficient to implement the Grover operator on the augmented search space of size  $2N$ . The state of the second register is initialized to  $\sum_x |x\rangle$  by  $H^{\otimes n}$ . But this is a superposition of the two eigenvectors of  $U_G$  with corresponding eigenvalues  $e^{i\theta}$  and  $e^{i(2\pi-\theta)}$ . Therefore, applying the phase estimation procedure results in an estimate for  $\theta$  or  $2\pi - \theta$  which is equivalent to the estimate for  $\theta$  ( $\sin^2(\theta/2) = \sin^2((2\pi - \theta)/2)$ ).

Counting the number of solutions and, consequently, determining the solvability of a search problem has many applications, including decision variants of NP-complete problems.

### 4.2.3 Applications of Grover's Algorithm

The quantum search algorithm or at least its main operator can be applied to solve many kinds of search problems. Table 4.2 presents a survey of these problems. Some of them are now described briefly.

**Claw finding:** Given two functions  $f : X \rightarrow Z$  and  $g : Y \rightarrow Z$ , find a pair  $(x, y) \in X \times Y$  such that  $f(x) = g(y)$ .

**Collision finding:** Given a function  $f : X \rightarrow Y$ , find two different  $x_1, x_2$  ( $x_1 \neq x_2$ ), such that  $f(x_1) = f(x_2)$  under the promise that such a pair exists.

**Database retrieval:** Given a quantum oracle which returns  $|k, y \oplus X_k\rangle$  on an  $n + 1$  qubit query  $|k, y\rangle$ , the problem is to obtain all  $N = 2^n$  bits of  $X_k$ .

**Element distinctness:** Given a function  $f : X \rightarrow Y$ , decide whether  $f$  maps different  $x \in X$  to different  $y \in Y$ .

**Minimum-finding:** Let  $T[0 \dots N - 1]$  be an unsorted table of  $N$  (distinct) items. Find the index  $y$  of an item such that  $T[y]$  is minimal.

**Scheduling problem:** Given two unsorted lists of length  $N$  each. Find the (promised) single common entry. In [70] the complexity is measured in quantum memory accesses and accesses to each list.

**String matching:** Determine whether a given pattern  $p$  of length  $m$  occurs in a given text  $t$  of length  $n$ . The algorithm combines quantum searching algorithms with deterministic sampling, a technique from parallel string matching. Grover's search algorithm is applied twice in conjunction with a certain oracle in each step.

**Triangle-finding:** Given an undirected graph  $G=(V,E)$ , find distinct vertices  $a, b, c \in V$  such that  $(a, b), (a, c), (b, c) \in E$ .

**Weighing matrix problem:** Let  $M$  be a  $W(n, k)$  weighing matrix<sup>12</sup>. A set of  $n$  functions  $f_s^M : \{1, \dots, n\} \rightarrow \{-1, 0, +1\}$  for  $s \in \{1, \dots, n\}$  is defined by  $f_s^M(i) := M_{si}$ . Determine  $s$ !

Problem	Runtime	References
Quantum database search (Grover's algorithm)	$O(\sqrt{N})$ , quadr. speedup	[61, 62]
Quantum counting	$O(\sqrt{N})$	[21, 24, 104]
Scheduling problem	$O(N^{3/4} \log N)$	[70, 64]
Minimum-finding	$O(\sqrt{N})$	[48]
Database retrieval	$N/2 + O(\sqrt{N})$	[151]
String matching	$O(\sqrt{n} \log \sqrt{\frac{n}{m}} \log m + \sqrt{m} \log^2 m)$	[122]
Weighing matrix problem	2 oracle queries	[152]
Element distinctness Collision finding Claw finding Triangle-finding	comparison complexity always better than classical	[27]

Table 4.2: Algorithms based on quantum searching.

#### 4.2.4 Quantum Search and NP Problems

Solving problems in the complexity class NP may also be sped up using quantum search. The entire search space of the problem (e.g. orderings of graph vertices) is represented

---

<sup>12</sup>A matrix  $M \in \{-1, 0, +1\}^{n \times n}$  is called a *weighing matrix*, iff  $M \cdot M^T = k \cdot I_n$  for some  $k, 0 \leq k \leq n$ .

by a string of qubits. This string has to be read as defined by the problem specifications (e.g. the string consists of blocks of the same length storing an index of a single vertex). In order to apply the quantum search algorithm, the oracle which depends on the problem instance must be designed and implemented. It marks those qubit strings which represent a solution. As the verification of whether a potential solution meets the requirements is much easier than the problem itself, even on a classical computer, it is sufficient to convert the classical circuit to a reversible circuit. Now, this circuit can be implemented on a quantum computer. Thus, the quantum counting algorithm which determines whether or not a solution to the search problem exists requires the square root of the number of operations that the classical “brute-force” algorithm requires. Roughly speaking, the complexity changes from  $O(2^{\theta(n)})$  to  $O(2^{\theta(n)/2})$ , where  $\theta$  is some polynomial in  $n$ . Nonetheless, the complexity is still exponential and in the sense of complexity theory these problems are not efficiently solved.

#### 4.2.5 Hogg’s Algorithm

Certain structured combinatorial search problems can be solved effectively on a quantum computer as well, even outperforming the best classical heuristics. In [72, 73] Hogg introduced a quantum search algorithm for 1-SAT and highly constrained  $k$ -SAT.

##### The satisfiability problem (SAT)

A satisfiability problem (SAT) consists of a logical formula in  $n$  variables  $v_1, \dots, v_n$  and the requirement to find an assignment  $a = (a_1, \dots, a_n) \in \{0, 1\}^n$  for the variables that makes the formula true. For  $k$ -SAT the formula is given as a conjunction of  $m$  clauses, where each clause is a disjunction of  $k$  literals  $v_i$  or  $\bar{v}_i$  respectively with  $i \in \{1 \dots n\}$ . Obviously there are  $N = 2^n$  possible assignments for the formula. A solution must satisfy every clause. Clauses which become false for a given assignment are called *conflicts*. Let  $c(a)$  denote the number of conflicts for assignment  $a$ . A  $k$ -SAT problem is called *maximally constrained* if the formula has the largest possible number of clauses for which a solution still exists. Thus, any conceivable additional clause will prevent the satisfiability of the formula. For  $k \geq 3$  the satisfiability problem is NP-complete and, in general, the computational costs grow exponentially with the number of variables  $n$ . For  $k = 1$ ,  $k = 2$ , and some  $k$ -SAT problems with a certain problem structure, there are classical algorithms which require only  $O(n)$  *search steps*, that is, the number of sequentially examined assignments. Also, quantum algorithms can exploit the structure of these problems to improve the general quantum search (ignoring any problem structure) and perform better than classical algorithms.

##### Hogg’s quantum algorithm for 1-SAT

Hogg’s quantum algorithm for 1-SAT primarily requires  $n$  qubits, one for each variable. A basis state specifies the values assigned to each variable and consequently, assignments and basis states correspond directly to each other. Information about the particular

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & i & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & i & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -i \end{pmatrix}$$

Figure 4.8: Input matrix for the logical formula  $f(v_1, v_2, v_3) = \bar{v}_1 \wedge \bar{v}_2 \wedge \bar{v}_3$ . The matrix has diagonal coefficients  $(i^{c(000)}, \dots, i^{c(111)})$ . For instance, the assignment  $a = 101$  ( $v_1 = \text{true}$ ,  $v_2 = \text{false}$ ,  $v_3 = \text{true}$ ) makes two clauses false, i. e.  $c(101) = 2$  and  $i^2 = -1$ .

problem to be solved is accessible by a special diagonal matrix  $R$  (of dimension  $2^n \times 2^n$ ), where

$$R_{aa} = i^{c(a)} \tag{4.8}$$

is the matrix entry at position  $(a, a)$ . Thus, the problem description is entirely encoded in this input matrix by the number of conflicts in all  $2^n$  possible assignments to the given logical formula. For the moment it is assumed that such a matrix can be implemented efficiently. Further implementation details will be given in the next paragraph. Figure 4.2.5 exemplifies the input matrix  $R$  for a 1-SAT problem with  $n = 3$  variables. Furthermore, let  $U$  be the matrix defined by

$$U_{rs} = 2^{-n/2} e^{i\pi(n-m)/4} (-i)^{d(r,s)},$$

where  $d(r, s)$  is the Hamming distance between two assignments  $r$  and  $s$ , viewed as bit strings.  $U$  is independent of the problem and its logical formula. The first step of the quantum algorithm is the preparation of an unbiased, i. e. equally weighted, superposition. As already demonstrated, this is achieved by applying the Hadamard gate on the  $n$  qubits ( $H^{\otimes n}$ ) initially in state  $|0\rangle$ . Let  $|\psi\rangle$  denote the resulting quantum state. Then, applying  $R$  and  $U$  to  $|\psi\rangle$  gives

$$|\phi\rangle = UR|\psi\rangle.$$

It can be proven that the final quantum state is the equally weighted superposition of all assignments  $a$  with  $c(a) = 0$  conflicts. Thus, considering a soluble 1-SAT problem, a final measurement will lead with equal probability to one of the  $2^{n-m}$  solutions. If there is no solution of the problem, the measurement will return a wrong result. Hence, finally the resulting assignment has to be verified.



### Implementation

The matrices  $R$  and  $U$  have to be decomposed into elementary quantum gates to build a practical and implementable algorithm. The operation  $R$  can be performed using a reversible (quantum) version of the classical algorithm to count the number of conflicts of a 1-SAT formula and a technique to adjust the phases which are powers of  $i$ . Therefore, two ancillary qubits are necessary which are prepared in the superposition

$$|\Psi\rangle = \frac{1}{2}(|00\rangle - i|01\rangle - |10\rangle + i|11\rangle),$$

where the local phases correspond to the four possible values of  $R_{aa}$  for any assignment  $a$ . As suggested by Hogg, the superposition  $|\Psi\rangle$  can be constructed by applying  $H$  on qubit 1 and  $iRx(3/4\pi)$  on qubit 0, both of the qubits being initially prepared to  $|1\rangle$ . Then, the reversible operation

$$F : |a, x\rangle \rightarrow |a, x + c(a) \bmod 4\rangle$$

acts on  $|\psi, \Psi\rangle$  as follows

$$|\psi, \Psi\rangle \rightarrow 2^{-n/2} \sum_a i^{c(a)} |a\rangle |\Psi\rangle,$$

performing the required operation  $R$ . To see this, further intermediate calculation steps are necessary which can be read up in [73]. Note that after applying  $F$ , the ancillary qubits reappear in the original superposition form and can therefore be dropped, since they do not influence  $U$ . In a single application of  $F$ ,  $c(a)$  is evaluated once.

The matrix  $U$  can be implemented in terms of two simpler matrices  $H^{\otimes n}$  and  $\Gamma$ , where  $\Gamma$  is diagonal with

$$\Gamma_{aa} = i^{|a|}.$$

Here  $|a|$  is the number of 1-bits in the bit string of assignment  $a$ . Using this definition, it is  $U \equiv H^{\otimes n} \Gamma H^{\otimes n}$ . For implementing  $\Gamma$ , Hogg suggests to use similar procedures to those for the implementation of the matrix  $R$ , using a quantum routine for counting the number of 1-bits in each assignment instead of the number of conflicts [73]. Thus, the elements of the matrix  $\Gamma$  can be computed easily and the operation  $U$  is efficiently computable in  $O(n)$  bit operations:  $2n$  Hadamard gates plus another  $C \cdot n$  elementary single qubit gates, with a constant  $C \geq 1$ , for the computation of  $\Gamma$  (to give a rough estimation). In 6.1 it is shown as a result of the GP evolution of a 1-SAT quantum algorithm that  $U = Rx(3/4\pi)^{\otimes n}$ , and consequently it can be implemented even more efficiently by  $n$  elementary rotation gates.

Once again, Hogg's algorithm consists of the following steps:

---

**Quantum Algorithm 4.2.1:** *1-SAT / maximally constrained k-SAT*

---

1. Apply  $H^{\otimes n}$  on  $|0\rangle$ .
  2. Compute the number of conflicts with the constraints (clauses) of the problem into the phases by applying the input gate  $R$ .
  3. Applying  $U = H^{\otimes n}\Gamma H^{\otimes n}$  results in an equally weighted superposition of solution states.
- 

An experimental implementation of Hogg's 1-SAT algorithm for logical formulas in three variables is demonstrated in [117].

**Performance**

The matrix operations and the initialization of  $|\psi\rangle$  contribute  $O(n)$  bit operations to the overall costs. Evaluating the number of conflicts results in costs of  $O(m)$  for a  $k$ -SAT problem with  $m$  clauses. In total the costs of the quantum algorithm amount to  $O(n+m)$ . This corresponds to the costs of a single search step of a classical search algorithm which is based on examinations of neighbors<sup>13</sup> of assignments. While the quantum algorithm examines all assignments in a single step, the best (local search based) classical algorithm needs  $O(n)$  search steps.

A slide modification of Hogg's algorithm for 1-SAT can also be applied to maximally and highly constrained  $k$ -SAT problems for arbitrary  $k$  to find a solution with high probability in a single step. For all 1-SAT and also maximally constrained 2-SAT problems, Hogg's algorithm finds a solution with probability one. Thus, an incorrect result definitely indicates the problem is not soluble [72, 73].

Note that a comparison of Hogg's algorithm with any classical algorithms according to the number of search steps is only permissible, if the classical algorithms are based on local search, especially on the examination of the number of conflicts. Otherwise, the comparison must be based on a more fundamental measure. Local search is not always the best solution. For instance, it is unreasonable to solve 1-SAT using local search since it is trivial to find an assignment satisfying the given Boolean formula in  $O(m)$ .

### 4.3 Quantum Simulation

The third class of quantum algorithms consists of those algorithms which simulate quantum mechanical systems. Commonly, the problem of simulating a quantum system is

---

<sup>13</sup>These are assignments that differ in a single value.

classically (at least) as difficult as simulating a quantum computer. This is due to the exponential growth of the Hilbert space which comprises the quantum states of the system. Therefore, simulation of quantum systems by classical computers is possible, but in general only very inefficiently. A quantum computer can perform the simulation of some dynamical systems much more efficiently, but unfortunately not all information from the simulation is accessible. The simulation leads inevitably to a final measurement collapsing the usually superimposed simulation state into a definite basis state. Nevertheless, quantum simulation seems to be an important application of quantum computers.

A rough outline of the quantum simulation algorithm described in [107] is presented now. Further reading matter are the original papers dealing with the simulation of quantum physical systems on a quantum computer, among them [2, 116, 148, 164, 171, 172].

Simulating a quantum system means “predicting” the state of the system at some time  $t_f$  (and/or position) as accurately as possible given the initial system state. Simulation is mainly based on solving differential equations. Unfortunately, the number of differential equations increases exponentially with the dimension of the system to be simulated. The quantum counterpart to the simple differential equation  $dy/dt = f(y)$  in classical simulations is  $i d|\psi\rangle/dt = H|\psi\rangle$  for a Hamiltonian  $H$ . Its solution for a time-independent  $H$  is  $|\phi(t)\rangle = e^{-iHt}|\phi(0)\rangle$ . However,  $e^{-iHt}$  is usually difficult to compute. High order solutions<sup>14</sup> are possible especially for those Hamiltonians which can be written as sums over local interactions:  $H = \sum_{k=1}^L H_k$ , acting on an  $n$ -dimensional system,  $L = \text{poly}(n)$ , where each Hamiltonian  $H_k$  acts on a small subsystem. Now, the single terms  $e^{-iH_k t}$  are much easier to approximate by means of quantum circuits than  $e^{-iHt}$ . How to calculate  $e^{-iHt}$  from  $e^{-iH_k t}$ ? In general  $e^{-iHt} \neq \prod_k e^{-iH_k t}$  because of  $[H_j, H_k] \neq 0$ .<sup>15</sup> Instead, approximate  $e^{-iHt}$  with e. g.

$$e^{i(A+B)\Delta t} = e^{iA\Delta t/2} e^{iB\Delta t} e^{iA\Delta t/2} + O(\Delta t^3).$$

This and other approximations are derived from the *Trotter formula*

$$\lim_{n \rightarrow \infty} (e^{iAt/n} e^{iBt/n})^n = e^{i(A+B)t}.$$

Now, let the  $n$ -qubit state  $|\tilde{\psi}\rangle$  approximate the system. Assume further, that the operators  $e^{-iH_k \Delta t}$  have efficient quantum circuit approximations. Then, the approximation of  $e^{i(\sum_k H_k)\Delta t}$  can be efficiently implemented by a quantum circuit  $U_{\Delta t}$ . With it, the quantum simulation algorithm can be formulated as follows:

<sup>14</sup>Often the easier computable first order solution  $(I - iH\delta t)|\psi(t)\rangle$  is insufficient.

<sup>15</sup> $[A, B] := AB - BA$  is the *commutator* between two operators  $A$  and  $B$ .

---

**Quantum Algorithm 4.3.1:** *Quantum simulation algorithm*


---

1. Initialization:  $|\tilde{\psi}_0\rangle$  (initial system state at time  $t = 0$ ),  $j = 0$ ;
  2. Iterative evolution: **do**  $|\tilde{\psi}_{j+1}\rangle = U_{\Delta t}|\tilde{\psi}_j\rangle$ ;  $j = j + 1$ ; **until**  $j\Delta t \geq t_f$
  3. Output:  $|\tilde{\psi}(t_f)\rangle = |\tilde{\psi}_j\rangle$
- 

## 4.4 Speedup Limits for Quantum Algorithms

This section summarizes some theoretical results about the limitations of quantum computing. The methodologies to prove lower bounds for the complexity and the speedup of quantum algorithms are not the subject matter, but they can be read up in the given literature.

An interesting result on limitations of quantum algorithms refers to black-box algorithms [11]. Many quantum algorithms use oracle or black-box gates, which hide an unknown function, or they are at least expressible as a black-box algorithm. Accessing the black-box can be thought of as a special subroutine call or query whose invocation only costs unit time. Speaking more formally, let  $X = (x_0, \dots, x_{N-1})$  be a such an oracle, containing  $N$  Boolean variables  $x_i \in \{0, 1\}$ . On input  $i$  the oracle returns  $x_i$ . A property  $f$  of  $X$  which is any Boolean function  $f : \{0, 1\}^N \rightarrow \{0, 1\}$  is only determinable by oracle queries. The complexity of a black-box algorithm is usually rated by the number of queries necessary to compute the property. Then, Beals et al. prove in [11] that black-box quantum algorithms for which no inner structure is known, i.e., for which no promises are made restricting the function<sup>16</sup>, can achieve maximally a polynomial speedup compared to probabilistic and deterministic classical algorithms. In addition, they specify exact bounds for the (worst case) quantum complexity of *AND*, *OR*, *PARITY* and *MAJORITY* for different error models (assuming exact -, Las Vegas - and Monte-Carlo algorithms). For the *exact* setting (the algorithm returns the correct result with certainty) the quantum complexities are  $N$  for *OR* and *AND*,  $N/2$  for *PARITY* and  $\Theta(N)$  for *MAJORITY*. The bound for the parity problem was independently obtained by Farhi et al. [51].

As shown by Buhrman et al. [27] the following problems cannot be solved more efficiently on a quantum computer:

---

<sup>16</sup>Then, the function is said to be *total*.

**Parity-collision problem:** Given a function  $f : X \rightarrow Y$ , find the parity of the cardinality of the set  $\{(x_1, x_2) \in X \times X | x_1 < x_2 \wedge f(x_1) = f(x_2)\}$ .

**No-Collision problem:** Given a function  $f : X \rightarrow Y$ , find an element  $x \in X$  that is not involved in a collision, i. e.,  $f^{-1}(f(x)) = \{x\}$ .

**No-range problem:** Given a function  $f : X \rightarrow Y$ , find an element  $y \in Y$  such that  $y \notin f(X)$ .

Høyer et al. [75] determined the quantum complexity for further problems.

**Ordered searching:** Given a sorted list of  $N$  numbers  $(x_0, x_1, \dots, x_{N-1})$ ,  $x_i \leq x_{i+1}$ , and a number  $y \leq x_{N-1}$ , find the minimal index  $i$  such that  $y \leq x_i$ . This problem can be regarded as a non-Boolean promise problem. The best known quantum lower bound is  $1/\pi(\ln(N) - 1)$  queries. In [75], a quantum algorithm using  $\log_3(N) + O(1) \approx 0.631 \log_2(N)$  queries is presented. A slightly better algorithm ( $0.526 \log_2(N)$  queries) is given in [51].

**Sorting:** Given a list of numbers  $(x_0, x_1, \dots, x_{N-1})$ , output a permutation  $\sigma$  on the index set such that the list  $(x_{\sigma_0}, x_{\sigma_1}, \dots, x_{\sigma_{N-1}})$  is in non-decreasing order. The quantum lower bound for this problem is given by  $\Omega(N \log N)$  binary comparisons.

**Element distinctness:** Given a list of numbers  $(x_0, x_1, \dots, x_{N-1})$ . Decide whether they are all distinct. The problem has a quantum lower bound of  $\Omega(\sqrt{N} \log N)$  binary comparisons.



## 5 Evolution of Quantum Algorithms

*Everything what is possible  
happens, if only there is enough  
time to.*

---

Herodotus (484–425 BC)

Evolution of quantum circuits faces two major challenges, increasing costs for simulating quantum algorithms for multiple qubits on conventional computers and the complex and large search spaces of quantum programs. On one hand the exponential growth of the state space (with the number of qubits) makes simulation increasingly time consuming, but on the other hand the same state space is the source of entangled and superposed states responsible for the power of quantum computation. Thus, for quantum computing and the development of better-than-classical quantum algorithms, the Hilbert space seems to be likewise a curse and a blessing. The exponential growth of the search space is due to the impact of combinatorics, in the form of the number of variations (with repetitions) a certain number of gates can be chosen from a given set of gates. The complexity and structure of the search space depends on the problem.

In this chapter the problematic nature of the evolution of quantum circuits is confronted and addressed. The second considerable part deals with implemented GP systems, those described in previous work and those developed in the course of this thesis. In detail this chapter is organized as outlined in the following paragraphs:

Section 5.1 deals with the simulation of quantum circuits on a classical computer and its complexity. It describes the quantum computer simulator as it is used in all experiments for this thesis, discusses implementation details and the time consumption for quantum circuit evaluation.

Designing a quantum circuit to solve a given problem is highly non-intuitive and therefore difficult for a human programmer. Unfortunately, empirical studies in the scope of this work showed that large search spaces in apparently simple problems together with the usually exponentially or even super-exponentially increasing number of fitness cases render evolutionary approaches nearly unable to achieve breakthrough solutions in the development of *new* quantum algorithms. Section 5.2 makes some more general remarks about searching the space of quantum circuits and discusses possible test- and benchmark-problems, which are unfortunately not present in large numbers.

Using genetic programming to evolve quantum circuits is not a novel approach. It was pioneered in 1997 by Williams and Gray [167]. Since then, several other papers [9, 10, 57, 127, 137, 136, 140, 139, 168] dealt with the connection of quantum computing and genetic programming or genetic algorithms, respectively. The different approaches to GP/GA-based quantum circuit design described in these papers are inspected more closely in Section 5.3.

Finally, Section 5.4 describes both GP systems, linear-tree and linear GP; the former supporting intermediate measurements, the latter preventing them. It also discusses different kinds of fitness functions.

## 5.1 Quantum Circuit Simulation

Quantum computer simulators are computer programs simulating the action of a quantum computer on a classical computer. Since (universal) quantum computer hardware is not yet available<sup>1</sup> simulations on classical hardware enable researchers to develop and, in particular, to test quantum algorithms.

There are a lot of quantum simulators available on the Internet, and a detailed, up-to-date overview of them was presented for some time on the former homepage of Julia Wallace at the University of Exeter, UK. Unfortunately, this page no longer exists. Yet, in [161] of October 1999, she offers a (at that time) topical review on a variety of different simulators available to download. They are all programmed to simulate the behavior of a quantum mechanical system and operations applied to it. From the multitude of (special purpose) quantum computer simulators, QCL by B. Ömer is noteworthy, because of its additional functionality. QCL (Quantum Computation Language) is a high level programming language with classical syntax, containing variables and expressions, operators and program constructs, such as loops, subroutines and conditional branchings. Quantum algorithms are programmed in QCL and run on an interpreter, simulating the quantum computer with an (in theory) arbitrary number of qubits.

For quantum circuit evolution the quantum computer simulator has to meet certain requirements. Especially, it has to be very efficient, since unnecessary costs multiply rapidly during the evolutionary process. The simulator described in this section is that used in the linear and linear-tree GP systems (cf. Section 5.4) and is, like the GP systems, implemented in C++. It simulates quantum algorithms, based on the idealized, noiseless, i.e., decoherence-free, quantum circuit model. The task of the simulator is simply: to read and to interpret the quantum circuit given as a sequence of descriptive instructions, to read the initial system state, and to calculate (from this information) the final state vector. Using intermediate measurements will naturally lead to further calculations of

---

<sup>1</sup>Here, the seven-qubit realization [154] is not regarded as a quantum computer — in the same way as a pocket calculator is not comparable to a PC. This is not to disrespect the value of this research achievement, but to point out that its applications are highly restricted.



state vectors, since for the overall assessment of a quantum circuit, all possible outcomes must be considered.

In the following subsections some of the implementation details, such as the representation and storage of quantum circuits, the calculation of the final vector (using matrix-vector or matrix-matrix multiplications) and intermediate measurements are discussed. Time measurements of circuit evaluations on the implemented simulator convey an impression of the time consumption. In addition, quantum circuit reduction based on substitution rules is also touched upon.

### 5.1.1 Representation of Quantum Circuits

Strictly speaking, the representation of quantum circuits for use in the quantum computer simulator does not necessarily correspond to their representation as individuals in the GP system. Disregarding intermediate measurements which provide branchings and, consequently, a (linear-)tree structure, naturally quantum circuits are absolutely linear. Only in conjunction with classical program control constructs – resulting in hybrid algorithms – other program flow structures, such as simple loops, are implementable. In contrast to the pure linear representation used in the simulator, the individual representation of quantum circuits in a GP system might be based on trees or even graphs.

In the quantum computer simulator quantum circuits are represented and described as sequences (lists) of *instructions*, each entirely determining a certain quantum gate. An instruction consists of a shorthand expression of the gate's type, such as *H*, *Rx*, or *CNOT* (cf. Section 2.4) followed by different type-specific parameters. For all single-qubit gates, the first parameter is the number of the qubit the gate acts on. Some rotation gates need an additional real-valued angle parameter. A global system parameter restricts the resolution of the angle parameters to  $r$  bits allowing only rotations as multiples of  $2\pi/2^r$ . For all controlled-gates, the first parameter is the target qubit. The second parameter is, in the case of the *CPH* gate, a single control qubit followed by an angle parameter defining the phase shift and, in the case of a  $C^k$ *NOT* gate, a set of control qubits. The parameters of the *SWAP* gate are merely two qubits, which are to be swapped. Intermediate measurements, indicated by the gate type *M* and the qubit which is to be measured, require an additional structure in which the branching instructions along with their pre-measurement vectors and the branching probabilities are stored. This is explained in more detail in Section 5.1.5. The linear-tree GP scheme (cf. Section 5.4.1) arises naturally from the use of intermediate measurements which provide quantum circuits with an additional tree structure. The identity *ID* needs no further parameters, as it has no effect at all. The only reason for the use of this gate is its action as an intron in the context of evolution (cf. Section 3.6). Thus, it provides additional (implicit) neutrality or neutral mutations respectively. Though, there are already gates working as introns, such as  $Rx(0)$ ,  $Ry(0)$  or  $Rz(0)$ . A special gate is the *input gate* (*INP*) which is provided by the GP system and the problem description. It is a placeholder for one

or more unitary matrices encoding the problem instance. To distinguish between several input gates these gates are numbered. Moreover, a certain input gate could have to be applied more than once within a quantum circuit to allow problem solving. Both cases are possible and can be handled in the quantum simulator. The number of input gates in a circuit is restricted by a global system parameter.

The set of quantum gates (gate types) implemented in the quantum computer simulator (except measurements) is

$$\{ID, H, NOT(= X), Y, Z, C^k NOT, Rx(\phi), Ry(\phi), Rz(\phi), CPH(\phi), SWAP, INP\}.$$

For the evolution of quantum circuits usually a universal subset of these gates were used. All gates are administered in a pool, that is, they are globally accessible and the memory to hold the matrix coefficients is allocated only once during the initialization of the simulator and the entire GP system. For the *NOT* and *C<sup>k</sup>NOT* gates, which are in principle only permutations, it is of course sufficient to store just the qubits they are applied to and not necessary to store the *NOT* matrix. The use of a gate pool saves much time otherwise arising from periodical allocations and deletions.

### 5.1.2 Matrix-Vector vs. Matrix-Matrix Multiplications

The costs for evaluating a linear quantum program depend essentially on the number of qubits, that is, the dimension of the vector space, and the length of the quantum algorithm, in other words, the number of elementary quantum gates. It is obvious that the impact on the costs is larger by adding a qubit than by adding another quantum gate.

#### Single Quantum Circuit Evaluation

Consider a quantum computer consisting of  $n$  qubits. Each operation on that quantum computer corresponds to a unitary  $2^n \times 2^n$  matrix. As already mentioned, quantum gates operating on  $m < n$  qubits have to be adapted to higher dimensions. However, they are still referred to as  $m$ -qubit gates, since all other qubits are not affected. From the information of a given instruction – be it a single-qubit gate  $U$  or a controlled-gate  $C^k(U)$  of a single-qubit operation  $U$  – such a higher dimensional matrix can be assembled, as explained in the Sections 2.4.1 and 2.4.2. This matrix solely consists of the  $2^{n-1}$  or  $2^{n-1-k}$  submatrices  $U$ , respectively. Hence, in the worst case, a quantum circuit is only made of elementary single-qubit gates. For now, input matrices are left out.

Consider a quantum circuit of  $L$  elementary (single-qubit) quantum gates, denoted  $U_1 U_2 \cdots U_L$ . Let  $v_0$  be the initial system state or vector, respectively. When the quantum algorithm is applied to an initial quantum state, there are basically two possible approaches to calculate the final state vector  $U_L U_{L-1} \cdots U_2 U_1 v_0$ :

1. by matrix-vector multiplications, i. e., successively applying the transformations of the sequence on the present state vector:

$$U_1 v_0 =: v_1; \quad U_2 v_1 =: v_2; \quad \dots \quad U_L v_{L-1} =: v_L$$

or

2. by matrix-matrix multiplications, i. e., calculating the entire transformation matrix  $U_{alg}$  of the quantum algorithm and performing a single matrix-vector multiplication at the end:

$$U_L U_{L-1} \dots U_2 U_1 =: U_{alg}; \quad U_{alg} v_0 =: v_L.$$

A short analysis shows what was to be expected: the first approach is more efficient than the second for all parameter settings of  $n$  and  $L$ . The comparison is based on the overall number of multiplications and additions of two complex values needed to calculate the result. Concerning the first approach, the number of multiplications of a  $2 \times 2$  submatrix with a subvector of the entire state vector amounts to  $L \cdot 2^{n-1}$ , resulting in  $L \cdot 2^{n+1}$  multiplications of two complex values in total. The number of additions is  $L \cdot 2^n$ . Also the second technique benefits from sparse transformation matrices. Using the standard “quick and dirty” method the number of multiplications in  $\mathbb{C}$  involved in producing the product of two  $N \times N$  matrices requires  $N^3$  multiplications of two complex values. However, multiplication of an arbitrary  $2^n \times 2^n$  matrix by a transformation matrix of a single-qubit gate requires only  $2^{2n+1}$  multiplications of two complex numbers, since such a matrix has in each row and column only two coefficients not equal to zero. The final matrix-vector multiplication contributes an additional  $2^{2n}$  multiplications. In summary, this approach needs  $(L - 1) \cdot 2^{2n+1} + 2^{2n}$  multiplications of complex values. The number of additions is  $(L - 1) \cdot 2^{2n}$  for the matrix-matrix multiplications and  $2^n(2^n - 1)$  for the final matrix-vector multiplication.

Possibilities for improvements are not in sight. Using Strassen’s fast algorithm for matrix multiplication [145] is not helpful in this particular case. This algorithm requires  $N^{\log_2 7}$  multiplications to multiply two arbitrary  $N$  by  $N$  matrices, with  $N = 2^k$ . But the power of the “divide and conquer”-algorithm is based on the idea of partitioning matrices, which basically has no effect here because of the special matrix form. So, savings could mainly result from simple  $2 \times 2$  matrix multiplications, where Strassen’s algorithm only needs 7 instead of 8 multiplications of numbers. Unfortunately, then the number of additions and subtractions increases from 4 to 18. Moreover, even assuming there could be a benefit from Strassen’s algorithm, experimental results suggest that it would not outperform the standard method until  $N > 100$  [33] (this would correspond to about 7 qubits). The current best result for matrix multiplication is Coppersmith and Winograd’s  $O(N^{2.376})$  algorithm [36]. However, this has more of a theoretical rather than a practical relevance since the asymptotic gain of this method is noticeable only for “astronomical matrix dimensions” [25]. Finally, though one of the matrices is very

sparse,  $2^{2n}$  has to be the lower bound for the number of multiplications since it is assumed that the other matrix is dense and each of its matrix coefficients has to be multiplied at least once. In this respect,  $2^{2n+1}$  multiplications are an acceptable result. Note, it still remains an unsolved problem in algebraic computational complexity theory to determine the minimum number of arithmetic operations required for matrix multiplication.

### Reuse of Results in the Multiple Fitness Case Scenario

In a broader context, the number of fitness cases within the scope of the GP evolution has to be considered. At first sight, the second approach might be especially expedient for large numbers of fitness cases when the higher costs of the matrix-matrix multiplications pay off because of the reuse of certain results: This method calculates all *invariable* components of the quantum algorithm only once. Then, for each fitness case, it combines the resulting matrices with one or more variable input matrices to build the transformation matrix of the given fitness case. At the end, a single matrix-vector multiplication is performed. In contrast, if the fitness cases do not differ in the initial system state on which the quantum algorithm has to be applied, the first method can save computational costs by calculating the last system state before applying the (first) input matrix and, afterwards, using this result for each fitness case to compute the final outcome. However, the savings are not too large if the (first) input matrix is placed in the beginning of a quantum circuit right after creating a superposition state using  $n$  Hadamard gates. In any case, considering also the use of the simulator within the GP system may change the efficiency rating of the two approaches.

For a closer inspection, let  $c$  be the number of fitness cases which are described by a single input matrix. Furthermore, let the initial state vector be the same for all fitness cases. For the first method, the analysis is done without considering possible savings by any precalculations, whereas for the second approach, it is supposed that the input matrix divides the quantum circuit into two parts no matter where this happens. Finally, it is assumed that the input matrix is a dense matrix, as in principle any unitary matrix can be an input matrix – although this is usually a permutation or even a diagonal matrix. These assumption are justified in the sense of a worst-case analysis. Then, for the first method the number of multiplications in  $\mathbb{C}$  comes to

$$c \cdot (L - 1) \cdot 2^{n+1} + c \cdot 2^{2n} .$$

The additive  $c \cdot 2^{2n}$  corresponds to the costs caused by multiplications with the input matrices. The second method calculates the two invariable parts of the quantum circuit ( $U'$ ,  $U''$ ) – before and after the input matrix ( $U_{inp}$ ) – requiring  $(L - 3) \cdot 2^{2n+1}$  multiplications. For each fitness case, an additional  $2 \cdot 2^{3n}$  multiplications arise for calculating the entire transformation matrix ( $U'' \cdot U_{inp} \cdot U'$ ) including the input matrix using the standard method, plus  $2^{2n}$  multiplications resulting from the final matrix-vector product. This adds up to

$$(L - 3) \cdot 2^{2n+1} + c \cdot (2 \cdot 2^{3n} + 2^{2n})$$

multiplications in  $\mathbb{C}$ . As already pointed out, there is only a rather small scope for improvements in the two multiplications of the two dense matrices when using Strassen's algorithm. The Figures 5.1–5.3 visualize the costs of both methods for  $n = 2 \dots 4$  and different settings for  $L$  and  $c$ .

The exact number of additions of two complex numbers is

$$c \cdot (L - 1) \cdot 2^n + c \cdot 2^n(2^n - 1),$$

for the first method and

$$(L - 3) \cdot 2^{2n} + c \cdot (2 \cdot 2^{2n}(2^n - 1) + 2^n(2^n - 1))$$

for the second. The number of additions inevitably has the same order of magnitude as the number of multiplications.

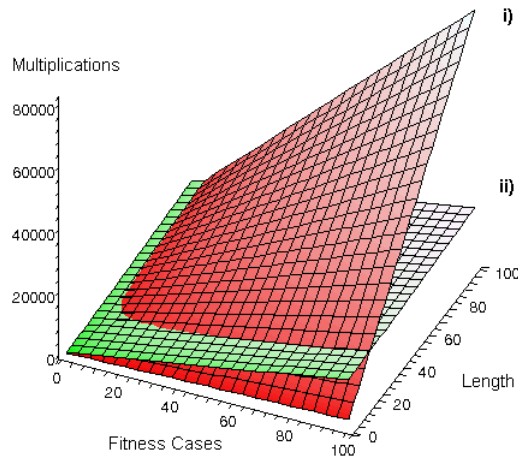


Figure 5.1: Comparison of i) the matrix-vector multiplication and ii) the matrix-matrix multiplication approach for quantum circuit evaluation for  $n = 2$  qubits.

As a first result revealed by the plots, both the length of the quantum algorithm and the number of fitness cases have to exceed certain thresholds, so that the second method should be favored. In particular, the threshold for the circuit length  $L$  is much larger than the threshold for the number of fitness cases  $c$ . The curve of intersection is given by the following equation:

$$c = \frac{(L - 3)2^n}{L - 1 - 2^{2n}}.$$

The formula and its inverse specify the asymptotes for  $L$  and  $c$ . The threshold for  $c$  amounts to  $2^n$ . Since the number of fitness cases usually increases exponentially or even

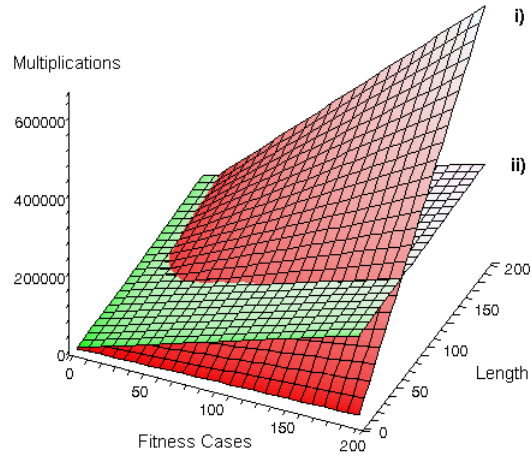


Figure 5.2: Comparison of i) the matrix-vector multiplication and ii) the matrix-matrix multiplication approach for quantum circuit evaluation for  $n = 3$  qubits.

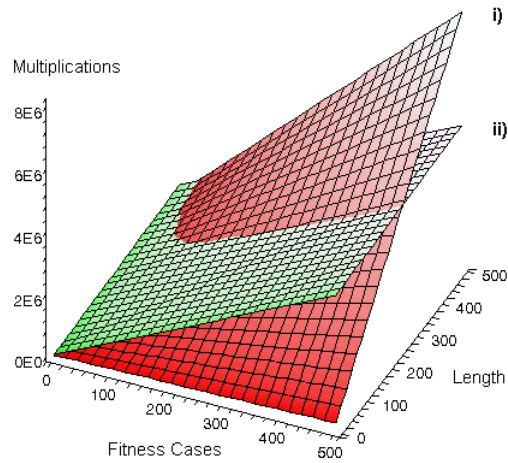


Figure 5.3: Comparison of i) the matrix-vector multiplication and ii) the matrix-matrix multiplication approach for quantum circuit evaluation for  $n = 4$  qubits.

super-exponentially with the number of qubits (such as is the case for the Deutsch-Jozsa problem and the satisfiability problem) it will in general not be decisive for the choice of evaluation methods. Provided a sufficiently large number of fitness cases,  $L$  has to exceed the limit of  $1 + 2^{2n}$ . Thus, the threshold for  $L$  is rather huge already for only few qubits: for  $n = 2 \dots 5$  qubits the threshold values are 17, 65, 257, and 1025. Moreover, with an increasing number of qubits it becomes harder and harder for the second approach to prevail against the first method.

Provided the usually sparse input matrices, such as diagonal or permutation matrices, the second method performs better and the threshold for  $L$  reduces to  $2^{2n-1} + 2^{n-1}$ , which is roughly speaking a little bit more than half the value of the general threshold. As already mentioned above, the first method (matrix-vector multiplications) can even do better by reusing the once calculated quantum state, which is the last state before applying the input gate. Assuming, that the input matrix is in the middle of the quantum algorithm, the method can save about  $(c-1) \cdot \lfloor L/2 \rfloor \cdot 2^{n+1}$  multiplications (although, according to the quantum algorithms already given, the input matrix is to be expected more likely at the beginning, right after building a usually equally weighted superposition). This would again increase the threshold by a factor of two.

As a consequence of the analysis above, the first method is implemented. Even for small numbers of qubits the quantum circuits have to be very large so that the second method might become interesting. However, the number of elementary quantum gates used to implement already known quantum algorithms is below the threshold. Further possibilities for speed-ups cannot be ruled out, but are not apparent.

### 5.1.3 Implementation and Time Measurements

In the following, a program subroutine is specified implementing the application of a general single-qubit gate to an arbitrary quantum state. So, let  $a, b, c$  and  $d$  be any complex numbers such that

$$U = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

is unitary. Suppose  $U$  acts on qubit  $q$  of  $n$  qubits in total and is applied to a quantum state  $v$  in the  $2^n$  dimensional vector space, i. e.

$$\underbrace{(I \otimes \dots \otimes I)}_{n-(q+1)} \otimes U \otimes \underbrace{(I \otimes \dots \otimes I)}_q \cdot v.$$

Compare also to Figure 2.5 in Chapter 4, which gives a simple example for the form of the composed matrix, with its blocks and submatrices, illuminating the working of this algorithm.

**Algorithm**

(\*  $bl$  is the number of major matrix blocks;  $sm$  is the number of  $2 \times 2$  submatrices within a block;  $step$  is the step size from one block's first row to the next block's first row;  $pos_1$  and  $pos_2$  are the vector positions, affected by the submatrix multiplication;  $a_1$  and  $a_2$  are auxiliary variables;  $v_{pos_1}$  and  $v_{pos_2}$  are the components of vector  $v$  at position  $pos_1$  and  $pos_2$ ; \*)

1.  $bl \leftarrow 2^{n-q-1}$
2.  $sm \leftarrow 2^q$
3.  $step \leftarrow 2 * sm$
4. **for**  $i \leftarrow 0$  **to**  $(bl - 1)$
5.     **do** (\* for each block \*)
6.         **for**  $j \leftarrow 0$  **to**  $(sm - 1)$
7.             **do** (\* for each submatrix within the block \*)
8.                  $pos_1 \leftarrow i * step + j$
9.                  $pos_2 \leftarrow pos_1 + sm$
10.                  $a_1 \leftarrow v_{pos_1}$
11.                  $a_2 \leftarrow v_{pos_2}$
12.                  $v_{pos_1} \leftarrow a * a_1 + b * a_2$
13.                  $v_{pos_2} \leftarrow c * a_1 + d * a_2$

Since calculating powers of base two up to an exponent  $n$  is a recurrent task in the evaluation routine and therefore is quite expensive, the powers are calculated in advance during the initialization of the quantum simulator and stored in a globally accessible array.

As already reported, for controlled gates the number of multiplications with submatrices is reduced to  $2^{n-1-k}$ , where  $k$  is the number of control qubits. Aside from this, the implementation of the gate, being applied to a quantum state, is nearly the same and therefore left out. Obviously, for *NOT* and controlled-*NOT* gates, it is not necessary to perform any multiplications. Since those gates just act as permutations, it is sufficient to carry out swap operations on the state vector.

To get an impression of how time-consuming quantum circuit evaluation is, several time measurements are performed using the first method (matrix-vector multiplications) for different numbers of qubits  $n$  and circuit lengths  $L$ . Table 5.1 provides the time in *seconds* required by the quantum simulator to evaluate 10,000 randomly created quantum circuits of given length  $L$ . The instructions of the quantum circuits are chosen from the gate (type) set  $\{H, NOT, CNOT, Rx(\phi), Ry(\phi), Rz(\phi)\}$ . Identity gates, input gates or intermediate measurements are not allowed. In particular, creating quantum circuits without input gates implies also that the evaluations are independent of any problem instance and any fitness cases. Moreover, the number of control qubits for  $C^kNOT$  is restricted to just one qubit. The quantum circuit generator does not select randomly from the set of all (concrete) gates. Instead, it first selects the gate type (equally distributed)



and, afterwards, the specific gate from the pool of all gates of this type. Every quantum circuit is applied to the first base vector  $(1, 0, \dots, 0)$ . The evaluations are performed on an *Intel Pentium M* processor with *1500MHz*.

L	Qubits n									
	2	3	4	5	6	7	8	9	10	15
5	0.111	0.161	0.26	0.351	0.501	0.821	1.472	2.764	5.378	178.547
10	0.201	0.291	0.36	0.52	0.851	1.472	2.754	5.378	10.516	341.251
15	0.28	0.341	0.45	0.691	1.151	2.093	3.996	7.802	15.282	507.83
20	0.31	0.401	0.541	0.852	1.462	2.835	5.227	10.335	20.499	671.06
25	0.331	0.44	0.631	1.072	1.823	3.395	6.469	12.719	25.307	834.3
30	0.381	0.501	0.731	1.192	2.123	3.986	7.751	15.282	30.464	992.43
35	0.42	0.561	0.811	1.372	2.424	4.667	8.993	17.875	35.26	1163.08
40	0.451	0.611	0.911	1.532	2.784	5.298	10.315	20.239	40.248	1324.71
45	0.481	0.671	1.001	1.703	3.114	5.858	11.827	23.073	45.275	1491.75
50	0.521	0.721	1.091	1.873	3.445	6.549	12.808	25.577	50.383	1645.97

Table 5.1: Time consumption (in seconds) for the evaluation of 10,000 arbitrary quantum circuits of length  $L = 5, 10, 15, \dots, 50$  on  $n = 2, 3, \dots, 10, 15$  qubits, performed on a PC with 1500MHz Intel Pentium M processor.

A quick glance of the measured times seems to be quite acceptable. However, the slowdown caused by the exponentially increasing Hilbert space is evident, as can be seen in Figure 5.4.

The effect on the evolution of quantum circuits is revealed only by considering the overall number of evaluations which depends on the “extent” of the problem, given by the number of fitness cases, and the complexity of the search space. As a matter of fact, huge and, above all, complex search spaces make evolutionary search difficult. It is not necessary to always evaluate a given quantum circuit for all fitness cases. Instead, doing this in a more restrictive manner can save much time. The time analysis is continued in the next section in the context of GP-based search.

#### 5.1.4 Circuit Reduction

The frequent evaluation of quantum circuits causes high computational costs. Therefore, it goes without saying to search for methods which reduce these costs. Simplification of the usually redundant quantum circuits before evaluation seems to be an obvious way: Applying *reduction rules*, such as  $H \cdot H \rightarrow I$  or  $Rx(\phi) \cdot Rx(\psi) \rightarrow Rx(\phi + \psi)$  may reduce the length of a circuit and hence the effective costs (multiplications and additions in  $\mathbb{C}$ ) for its evaluation. These rules replace parts of the quantum circuit by smaller circuits

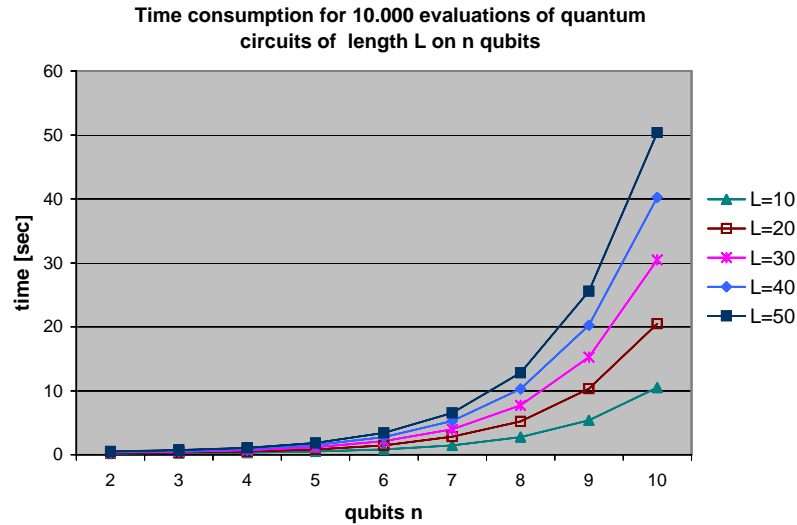


Figure 5.4: Graphical view of the time consumption as given in Table 5.1. The exponential increase of the evaluation time with an increasing number of qubits becomes visible.

or even remove parts, if they act like the identity. Naturally, it will depend on the set of rules whether a circuit reduction is possible. Specifically, the minimal quantum circuit consisting of gates from the elementary gate set needs not to be derivable by using these rules. Moreover, the application of rules will usually not be deterministic, that is, more than just one rule may be applied and at more than only a single position within the circuit. Besides, two rules can exclude each other if they are applicable to overlapping parts.

Given a set of reduction rules, to find the shortest derivable quantum circuit every possible sequence of applicable rules must be tracked. However, in general this will be an unreasonable approach as it exceeds the computational costs it saves by reducing the circuit length. Instead, a heuristics is suggested which works as follows: Scan the quantum circuit from left to right and for each circuit position the list of rules from top to bottom. If possible apply a rule. The substitution step is iterated until no further rule can be applied.

Assuming only rules which reduce a gate pair, in a single substitution step two elementary gates are substituted by another elementary gate or by the empty gate, i. e., they are removed. To find a substitutable gate pair, the algorithm has to run through the circuit, that is,  $O(L)$  gate pairs, each pair comparing with the left-hand side of  $R$  rules. In total, the circuit reduction results in  $O(L^2R)$  term comparisons. The exact number

of substitutions depends on the rule set and on the frequencies substitutable gate pairs occur.

Note that the reduction phase is not implemented in the current quantum simulator. Furthermore, the reduction phase should act only on a copy of the individual. A reduction of the individual can affect the evolutionary process, since introns and functional redundancies might be removed. They can be regarded as a source for beneficial changes in the genotype (cf. Section 3.6).

A selection of simple *substitution rules* for the gate set

$$S := \{H, NOT, CNOT, Rx(\phi), Ry(\phi), Rz(\phi)\}$$

is presented in Table 5.2. Indices refer to the qubits a gate acts on. Whenever gates are substituted by  $\epsilon$  they are removed from the circuit (and not substituted by  $I$ ).

Substitution Rules:	
R1	$Rz_q(0) \rightarrow \epsilon$
R2	$NOT_q \cdot NOT_q \rightarrow \epsilon$
R3	$H_q \cdot H_q \rightarrow \epsilon$
R4	$Rv_q(\phi) \cdot Rv_q(\psi) \rightarrow Rv_q(\phi + \psi), \forall v \in \{x, y, z\}$
R5	$CNOT_{t,c} \cdot CNOT_{t,c} \rightarrow \epsilon$
R6	$CNOT_{t,c} \rightarrow \epsilon$ if control qubit $c$ is definitely $ 0\rangle$
R7	$NOT_q \cdot Ry_q(7/4\pi) \rightarrow H_q$
R8	$U_{1q_1} \cdot U_{2q_2} \rightarrow U_{2q_2} \cdot U_{1q_1}, U_1, U_2 \in S \setminus \{CNOT\}, q_1 \neq q_2$
R9	$Rz_q(\phi) \cdot CNOT_{t,q} \rightarrow CNOT_{t,q} \cdot Rz_q(\phi)$
R10	$Rx_q(\phi) \cdot CNOT_{q,c} \rightarrow CNOT_{q,c} \cdot Rx_q(\phi)$

Table 5.2: A set of rules for gate substitutions. The indices indicate the qubit the gate is applied to. In case of  $CNOT_{t,q}$ , the first index refers to the target, the second to the control qubit.  $\epsilon$  is the empty gate.

Not all substitution rules, in principle useful for circuit reduction, have to be length reducing by themselves but may prepare a subsequent reduction. Good examples are the substitution rules R8, R9 and R10 in Table 5.2.

The described problem of quantum circuit reduction using certain substitution rules can also be viewed in the more general context of so-called *rewrite systems*. They are defined as sets of directed substitution rules used to compute by repeatedly replacing elements of a given expression with other elements until the simplest form possible is obtained [42]. Rewrite systems are extensively studied in computer science and mathematics, especially in the context of symbolic algebraic computation, automated theorem proving, program specification and verification. The problem of finding the shortest expression for a general set of rules especially including expansion rules (rules expanding the expression) is not computable. For general information on the subject of rewrite systems, see e. g. [20, 41, 42, 84].

### 5.1.5 Intermediate Measurements

The use of intermediate measurement necessarily leads to higher administrative costs. In particular, for each intermediate measurement one final state vector has to be calculated additionally. However, storing the provisional results and the probabilities for the measurement outcomes at the branching nodes in a buffer reduces the computational overhead to a minimum, since each quantum gate is still applied only once. Then, comparable to the infix traversal of trees, after evaluating the measurement gate (the node), first the subprogram following outcome “0” (arbitrarily defined to be the left subtree) and then the subprogram following outcome “1” (consequently the right subtree) is executed. Of course, the order of subprogram evaluations can also be switched. The result is a probability distribution of certain outcomes regarding a final measurement.

Using density matrices<sup>2</sup> does not change the necessity to run through the entire linear-tree structure. Let  $m$  be the number of intermediate measurements. For each of the  $m + 1$  paths from the root to a leaf a certain density matrix  $\rho_i = |\phi_i\rangle\langle\phi_i|$  evolves, where  $|\phi_i\rangle$  is the corresponding pure state and  $i = 0 \dots m$ . Another density matrix  $\rho$  can be used to compactly represent the ensemble of (pure) states which result from the traversal:  $\rho = \sum_i p_i \rho_i = \sum_i p_i |\phi_i\rangle\langle\phi_i|$  where  $p_i$  is the probability corresponding to state  $|\phi_i\rangle$ . Instead of calculating the measurement probabilities for every single pure state and adding them up to a total probability, it is of course possible to calculate this probability from the density matrix. The benefit of this approach depends on the number of intermediate measurements  $m$ , on the number of qubits  $n$ , and on the number of (single qubit) measurements at the end of the computation. For small numbers  $m$  and  $n$  (as they are used in the GP experiments) the benefit is negligible since also the generation of  $\rho$  contributes costs.

h

## 5.2 Searching the Space of Quantum Circuits

Quantum circuit evolution actually means searching the space of all possible or allowed quantum circuits, referred to as the *search space*, to find a circuit which is assigned the best or at least approximately the best fitness value. This does not necessarily mean that this circuit solves the problem it is designed for. In fact, a solution does not have to exist.

The search space is basically limited by two constrains: (i) the number of gates, and (ii) the maximum length of a quantum circuit. In particular, the search space is finite (discrete) provided that the additional (angle) parameters for certain gates are discrete, as well. Yet, this is important since otherwise a search can get lost in the infinite parameter subspaces. Furthermore, the number of gates is determined by the number of qubits.

---

<sup>2</sup>Readers unfamiliar with density matrices are referred to [107].

Figure 5.5 shows how the size of the search space scales for different numbers of qubits  $n$  and circuit lengths  $L$  for the gate set  $\{H, NOT, CNOT, Rx(\phi), Ry(\phi), Rz(\phi)\}$  with angle parameter  $\phi$  restricted to four bits allowing rotations as multiples of  $(1/8)\pi$ .

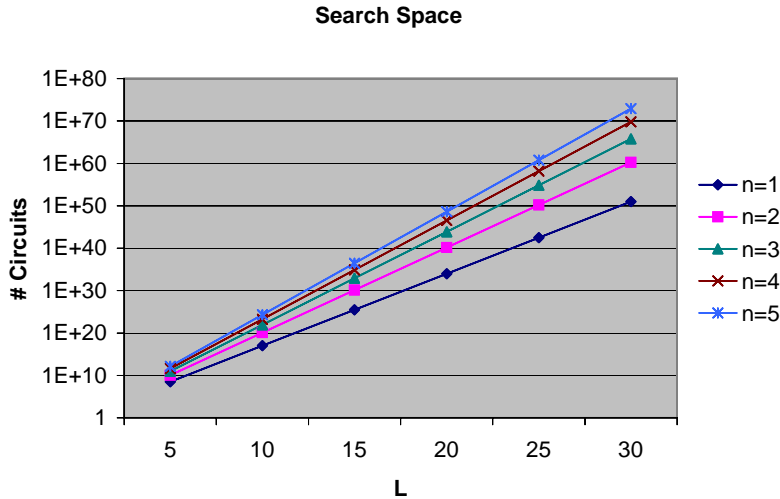


Figure 5.5: Number of quantum circuits forming the search space. The gate types  $H$  and  $NOT$  contribute  $n$  specific gates,  $Rx(\phi)$ ,  $Ry(\phi)$  and  $Rz(\phi)$  contribute  $n \cdot 2^4$  gates and  $CNOT$  contributes  $n(n-1)$  gates to the overall number of gates  $X$ . Then, the number of circuits is just  $X^L$ . To grasp the size and the growing of the search space: the number of circuits for five qubits and length  $L = 32$  finds its physical analogy approximately in the number of atoms in the universe, which is estimated to be  $10^{77}$  without dark matter [129]. Note the logarithmic scaling on the vertical axis.

There are of course other search heuristics than GP which could be used, but there are good reasons in favor of GP:

- It is very difficult to characterize the structure of the solution space. Not even knowledge about the unitary transformation matrix  $U$  helps to reduce the search space [167]. This has to be considered when choosing a search strategy.
- Local search strategies are very expensive since the entire solution must be evaluated in order to evaluate the effect of a local change in a circuit candidate. Here, GP seems to be more effective than other search strategies.
- The search method has to be “capable of considering solution structures of variable length” [167]. GP is able to work with structures of varying sizes.

### 5.2.1 Suitable Problems for GP-based Quantum Circuit Design

First of all, those problems for which quantum circuits could already be evolved successfully, are *per definition* suitable. Here, “successfully” means that the evolved circuit was either at least equivalent to the existing quantum algorithm developed by hand or better than any classical algorithm solving this problem. In Section 5.3, the different approaches to evolutionary quantum circuit design are described in detail. Yet, the problems treated in the related work are anticipated in Table 5.3. All these quantum algorithms, except for 2-AND/OR, existed already and did not need to be evolved *de novo*.

Problem	No. of Qubits ( $n$ )	max. No. of Fitness Cases ( $c$ )	min. Circuit Length	cf. Section
Quantum Teleportation	3	$\infty$	8	2.8.2, 5.3.4
Deutsch’s problem (2 bit)	3	8	5	2.8.1, 5.3.2
Quantum search (4 items)	3	4	9/16	4.2.1, 5.3.2
2-AND/OR problem	3	16	10	5.3.2
maximum entanglement	2–5	1	2	2.3.2, 5.3.3

Table 5.3: List of problems for which quantum circuits could be evolved. For quantum teleportation the number of fitness cases (third column) is theoretically infinite, since for an appropriate quantum circuit, any quantum state has to be teleported correctly. In practice, a small set of fitness case evaluations per individual is sufficient. However, to be effective they have to change during the evolution. The minimal circuit length necessary to solve the problem (fourth column) naturally depends on the elementary gate set. Here, this length corresponds to the number of gates in the shortest solution circuit known for a certain gate set used in the corresponding evolutionary algorithm. The circuit length for two different elementary gate sets is shown for the 2-bit (4 items) quantum database search problem. However, the order of magnitude is of importance, not the exact value. A description of the problem and information about the evolutionary algorithms used for circuit design can be found under the stated references.

What makes these problems suitable for quantum circuit evolution? First of all, only a small number of qubits is necessary to encode the problem instances. Second, proper quantum circuits solving the problems need only a small number of qubits. Third, the length of the quantum circuits solving the problems are moderate. And finally, the number of fitness cases is rather small. Only quantum teleportation has an infinite number of fitness cases. As experiments show, for this problem it is still sufficient to look at only a few fitness cases to evolve solutions.

Another problem suitable for automatic quantum circuit design is 1-SAT, for which a scalable quantum circuit already exists (cf. Section 4.2.5). It has all the beneficial properties which characterize the problems above: problem instances for  $n$  variables need only  $n$  qubits and the number of fitness cases amounts to  $\sum_{k=1}^n \binom{n}{k} 2^k$ , which is

sufficiently small for small  $n$ . Due to Hogg's implementation, the best minimal circuit length one could expect for a certain elementary gate set including the Hadamard gate was  $C \cdot n$  with a small constant  $C > 4$ . Hogg suggests the use of the input matrix plus  $3n$  Hadamard gates plus another diagonal matrix, which can be implemented at best using another  $n$  gates. This is however far from being obvious. In Section 6.1, it is proven that  $2n$  Hadamard and  $n$   $Rx(3/4\pi)$  gates are still sufficient. Note, these considerations hold only for problem instances, where the input matrix is given in Hogg's diagonal matrix form.

*PARITY*<sup>3</sup> is another suitable problem. Recently, Stadelhofer [142] presented results on the evolution of pure and mixed state quantum algorithms solving *PARITY*. For  $n$  qubits, the values of  $N = 2^n$  variables  $x_i$  are represented by an oracle matrix. A proper quantum circuit has to compute  $x_0 \oplus x_1 \oplus \dots \oplus x_{N-1}$ . The number of fitness cases is  $2^N$ . The evolved pure state circuit meets the lower bound for the query complexity, that is the number of oracle gates applied within the circuit, which is  $N/2$  as proven by Farhi et al. [51]. In total, the number of gates in the  $n$ -qubit circuit adds up to  $N + 1$ .

*Approximate quantum copying*, also called *approximate cloning*, is possible in contrast to exact copying (cf. Section 2.6). An optimal *quantum duplicator* or quantum copier, generating two identical copies, is presented in [29]. It needs three qubits and nine gates, six *CNOT* and three *Rx* gates. Of course, the number of fitness cases is theoretically infinite. This should not prevent a successful evolution of such a circuit. Also a *quantum triplicator network*, producing three copies, is suggested. However, it is not known whether this is optimal. Moreover, it is not clear how to construct the best general *multiplicator*, i. e., a quantum circuit producing multiple copies. In other words, it is still a challenge to find a "good" scalable approximate copy transformation.

It is difficult to find other problems which are prospectively suitable for quantum circuit evolution. Common problems in computer science usually need much more qubits to represent even smallest problem instances. Besides, proper quantum computations may need additional ancillary qubits to work. Furthermore, only those problems are of interest for which an evolved quantum circuit stands a chance to be better than any classical solutions. Ultimately, this is the point!

Thinking about suitable problems must also consider "suitable" input encodings, which are crucial for the success of quantum circuit evolution. Looking at the quantum algorithms in Chapter 4 reveals: inputs (or problem instances) are often encoded as quantum gates (input gates) and not as the initial base state of the quantum system. For example this is the case in the Deutsch-Jozsa algorithm and in Hogg's algorithm. Recapitulating the meaning of input gates explained in Section 2.4.5, an input gate is a black-box hiding a more or less complex quantum circuit which in general needs additional input and ancillary qubits. Input gates replace the intuitive problem inputs in the form of qubit

---

<sup>3</sup>Here, the parity problem is whether a given Boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  returns an even or odd number of 1s on the  $2^n$  different inputs.

assignments and provide the inputs in a preprocessed way. How this preprocessing is implemented is of secondary importance for the quantum circuit where the input gate is used in. Usually, the operation inherent to the input gate can be performed efficiently by a classical circuit from which the reversible quantum circuit can be derived. The knowledge about this is sufficient and the specific black-box circuit is not elaborated explicitly. By hiding the inside of the input gate, quantum circuits become more “legible” and structured as they focus only on the substantial part of the algorithm.

In the context of quantum circuit evolution, the choice of the input encoding by means of the input matrix is very important, since it influences the minimum number of qubits and gates needed to generate a quantum circuit solving the problem. For instance, the input matrix of Hogg’s algorithm conceals an extensive quantum circuit which calculates the number of conflicts (cf. Section 4.2.5). To evolve this circuit, additional qubits and quantum gates would be required making evolution much more time consuming and more difficult. Whether a given input coding is suitable may be hard to decide in advance. In any case, the choice of the input encoding has an effect on the evolution and the quantum circuit solution being evolved (if at all).

The experiments and investigations made within the scope of this thesis mainly refer to the evolution of quantum circuits for the Deutsch-Jozsa problem and 1-SAT. Additional experiments were done for the 2-AND/OR problem and the 2-bit quantum search problem. The evolved quantum algorithms (using linear GP) correspond to the known algorithms of Spector and Grover, but these results are not discussed here.

### 5.2.2 Time Consumption for Individual-Evaluations in GP-Search

The time consumption analysis, started in Section 5.1.3, is continued, now considering the overall costs of individual evaluations arising from the evolutionary search. To do that, it is easiest to regard the number of fitness cases as an average value, averaged over all evaluated individuals. Section 5.4 reports about the *early termination* mechanism, implemented in the GP systems to avoid further evaluations of fitness cases for individuals which are already classifiable as being “bad”.

To convey an impression about the *accumulated* costs of quantum circuit simulations during the evolutionary process, Tables 5.4 and 5.5 specify the time consumption for different qubit numbers, average circuit lengths, different numbers of fitness cases and individuals (10,000, 100,000, 1,000,000), each being evaluated for all fitness cases. The selection of parameter values is based on “real” problems: the Deutsch-Jozsa problem (Table 5.4) and the 1-SAT problem (Table 5.5). In particular, the number of fitness cases relates to the problem specifications. The percentage values in Table 5.4 (Deutsch-Jozsa) refer to the ratio of the number of fitness cases to its total number, which is  $2 + \binom{2^n}{2^n-1}$  for a given number of qubits  $n$ . For 1-SAT, only the maximum number of fitness cases,  $\sum_{k=1}^n \binom{n}{k} 2^k = 3^n - 1$ , are considered. Moreover, the circuit lengths are chosen appropriate to the problem instances, that is, they are a little larger than the optimum circuit



sizes. Evolution on the “edge” of the search space, caused by excessive constraints, is generally less successful. Consequently, the maximum circuit length should always allow adaptations to solutions from both directions, coming from the search subspace comprising smaller circuits and from that comprising larger circuits. However, the optimal solution size is usually unknown in advance and it seems to be a matter of experience to find the adequate setting.

To avoid misunderstandings, the indicated times are not determined by any GP runs but only by multiplying the number of fitness cases, the number of individuals and the time consumption for a single circuit evaluation, extracted from Table 5.1.

Qubits $n$	Length $L$	No. of fitness cases		No. of evaluated individuals		
				10,000	100,000	1,000,000
2	10	8	100%	1.6 sec	16 sec	2:40 min
2	15	8		2.2 sec	22.4 sec	3:44 min
3	10	72	100%	21 sec	3:30 min	34:55 min
3	15	72		25 sec	4:06 min	40:55 min
4	15	129	1%	58 sec	9:40 min	1:36 h
4	15	1288	10%	9:40 min	1:36 h	16:06 h
4	15	12872	100%	1:36 h	16:06 h	6 days, 17 h
5	15	60108	0.001%	11:32 h	4 days, 19 h	48 days
5	15	601080	0.01%	4 days, 19 h	48 days	480 days
5	15	6010804	0.1%	48 days	480 days	13 years

Table 5.4: Accumulated time consumption (according to Table 5.1) for the evaluation of a multitude of quantum circuits of average length  $L$  acting on a given number of qubits ( $n = 2 \dots 5$ ). The settings for the number of fitness cases are closely related to evolutionary runs for instances of the Deutsch-Jozsa problem.

Qubits $n$	Length $L$	No. of fitness cases		No. of evaluated individuals		
				10,000	100,000	1,000,000
2	10	8		1.6 sec	16 sec	2:40 min
3	15	26		7.5 sec	1:15 min	12:36 min
4	15	80		36 sec	6 min	1 h
5	15	242		2:47 min	27:52 min	4:39 h
6	20	728		17:44 min	2:57 h	1 day, 5 h

Table 5.5: Accumulated time consumption (according to Table 5.1) for the evaluation of a multitude of quantum circuits of average length  $L$  acting on a given number of qubits ( $n = 2 \dots 5$ ). The settings for the number of fitness cases are closely related to evolutionary runs for instances of the 1-SAT problem.

Looking at the tables, the core problem becomes visible: the rapidly increasing time consumption is jointly caused by the exponentially increasing Hilbert space and combinatorics. The overall number of inputs depends on the problem the quantum circuit has to solve, and as seen for the Deutsch-Jozsa problem, may have a big impact on the overall performance. Irrespective of this effect, even for smaller numbers of fitness cases such as it is the case for 1-SAT, the evolutionary search has to make rapid progress, since otherwise time is adding fast on the overall costs, owing to the large number of evaluated individuals. These findings indicate that any evolution of a quantum algorithm, which can find a “promising” solution, is limited to quantum systems with rather few qubits.

The following Table 5.6 supports this estimation in another way. Here, for each number of qubits  $n = 2 \dots 10$  and a given reasonable average circuit length  $L$ , the (average) number of fitness case evaluations is indicated for which the evolution requires at most *one hour*, expecting 100,000 examined circuits. This is of course an arbitrary example, however, it shows very well the complexity of quantum circuit evolution.

Qubits $n$	2	3	4	5	6	7	8	9	10
Length $L$	10	10	15	15	15	20	20	25	25
No. of fitness cases	1792	1238	800	521	313	127	69	29	15

Table 5.6: Assuming that the fitness of 100,000 quantum circuits of given length  $L$  on  $n$  qubits has to be calculated, the table specifies the maximum number of fitness case evaluations for each individual (on average) for which the evolution requires at most one hour. The data is inferred from Table 5.1.

Some further comments on the tables above and on evolutionary search of quantum circuits in general appear relevant:

- The measurements consider only time consumption in the context of quantum circuit evaluations, that is, matrix-vector multiplications. They do not consider the fitness function evaluation, and other costs resulting from the GP system, e. g. by applying genetic operators.
- As already mentioned, the circuit lengths are chosen appropriately following already known minimum solutions. For other problems the mandatory number of circuits might be much larger. The time consumption would increase to astronomical numbers, assuming the present upper limit for the length of an arbitrary quantum circuit decomposition (into elementary gates) to be the maximum length of quantum circuits (cf. Section 2.4.4).
- In this context, it is also important to realize that requiring smallest possible solutions usually makes evolution substantially harder. In [94] W. B. Langdon analyzes evolutionary search of reversible classical circuits solving the Boolean Six Multi-

plexor problem. He concludes that “these examples provide additional evidence, that requiring tiny solutions hurts evolvability”.

- The tables do not state anything about the influence of the (in length  $L$ ) exponentially increasing search space on the evolutionary search. The structure of the search space (when assigning each individual its fitness value) is, of course dependent on the problem. In fact, in the case of the Deutsch-Jozsa problem, the number of evaluated individuals on the average until a proper individual is found increases rapidly with the problem size, i. e., with any additional qubit (cf. with various plots in Chapter 6).
- Even when *early termination* or other mechanisms for saving fitness evaluations are applied, a quantum circuit is only a solution if it works properly on every input and, sooner or later, the circuit has to be applied to all inputs.
- The termination criterion of the evolutionary runs determines the duration of the evolution. Such a criterion might be a fitness threshold which is achieved or a certain number of evaluations of GP individuals which is exceeded. The search for a shortest quantum circuit solving a given problem is usually much more difficult and therefore time consuming. An approximately good solution may be “good enough”. Besides, optimizing such a solution in a postprocessing step might be much faster than continued searching for the shortest result.
- The times are determined on a certain computer. Of course, other computers are much faster and will reduce the time consumption. Furthermore, matrix-vector multiplications with matrices of the given block-structure are well suited for parallel computations. This can again save much time.

## 5.3 Previous Work

### 5.3.1 Automated Circuit Design by Williams & Gray

The idea of using genetic programming to design quantum circuits was discussed first in [167]. Given a unitary matrix  $U$  representing a desired quantum computation the aim was to find its decomposition into a sequence of simple quantum gate operations. In contrast to subsequent GP schemes for the evolution of quantum circuits, a circuit for the given problem is already known.

Williams and Gray focused on demonstrating a GP-based search heuristic which finds a correct circuit more efficiently than the exhaustive enumeration strategy. For this purpose they applied successfully a GP algorithm to the problem of discovering quantum circuits for the ‘send’ and ‘receive’ parts of quantum teleportation [15, 22]. In ten runs 26.4 generations were required on the average until the ‘send circuit’ with three qubits was

Algorithm:	generational GP
Program Structure:	linear
Function Set:	finite, approximate-universal quantum gate set; adequate undercomplete subsets, e. g. $\{CNOT, L, R\}$
Terminal Set:	the initial system state, e. g. $ 0\rangle$ , the qubits acted on by the gate, continuous gate parameters (were not implemented)
Representation:	a 3-tuple consisting of the gate matrix or its name respectively, gate parameters if any and the embedding (the qubits, the gate is working on followed by the total number of qubits); e. g. $\{H, params[], \{1;3\}\}$
Fitness Function:	$f(S, U) = \sum_{i=1}^{2^n} \sum_{j=1}^{2^n}  U_{ij} - S_{ij} $ , where $S, U \in U(2^n)$ ; $S$ is the unitary matrix of the quantum circuit generated by GP, $U$ is the unitary matrix of the target circuit
Selection:	ranking-based scheme; the selection probability corresponds to a quadratic form $P(r) = ar^2 + br + c$ , where $r$ is the individual's ranking
Genetic Operators:	mutation, i. e. changing only a gate's embedding, substitution (replacing existing gates), crossover (given two circuits A and B, only one child is created), transposition (inserting a subcircuit of A in circuit B), insertion (of a randomly constructed gate sequence), deletion (of a random subcircuit)
Termination:	$f(S, U) = 0$

Table 5.7: The GP algorithm by Williams and Gray, used to evolve decompositions of the unitary matrices representing the ‘receive’ and the ‘send circuit’.

found, using a population size of 100 circuits. The evolved ‘receive circuit’ (Figure 5.6) was even better than the best known teleportation receiver so far. For both evolutions the gate selection set consisted of the three gates  $L = X \cdot H$ ,  $R = Z \cdot H$  and  $CNOT$ . The specifications of the utilized GP algorithm are summarized in Table 5.7.

### 5.3.2 Quantum Circuit Evolution by Spector et al.

To evolve new quantum algorithms, in [140] Spector et al. draw up three GP schemes: the standard tree-based GP (TGP) and both stack-based and stackless linear genome GP (SBLGP/SLLGP). These are applied to evolve algorithms for Deutsch’s two-bit early promise problem using TGP, the scaling majority-on problem using TGP as well, the quantum four-item database search problem using SBLGP, and the two-bit-and-

or problem using SLLGP. The scaling majority-on problem is the same as Deutsch's problem, except that  $f$  may be any arbitrary Boolean function, that is, not necessarily constant or balanced. A program has to determine if the majority of the function's outputs are '1'. The two-bit-and-or problem is to determine whether the expression  $(f(0) \vee f(1)) \wedge (f(2) \vee f(3))$  is true or false for a Boolean two-bit function  $f$ . Better-than-classical algorithms could be evolved for all but the scaling majority-on problem. TGP and SBLGP are also designed to find scalable quantum algorithms using a *second-order encoding* technique. That is, the GP system evolves a classical program which, when executed generates a quantum algorithm dependent on the size of the problem instance it is applied to. In the following, the three GP schemes are described in more detail.

The TGP approach (Table 5.8) uses three different kinds of functions: algorithm-building functions, iteration control structures and arithmetic functions. The former add gates from the set  $\{ H, RY(\theta), CNOT, NOT\text{-}CCNOT, ORACLE \}$  to an initial empty quantum algorithm. The number and kind of arguments of these functions depend on the gate they represent. The return value is only a copy of one of its input arguments. Thus, in the sense of functional programming, the quantum algorithms are built by 'side-effects' of the algorithm-building functions. As the apparent pointless return values are again inputs to any other functions, this may still affect the construction of the quantum algorithm. The iteration control structures were included to help evolve scalable quantum algorithms. Arithmetic functions operating on terminals and return values of other functions are  $+$ ,  $1+$ ,  $-$ ,  $1-$ ,  $*$ ,  $*2$ ,  $sqrt$ ,  $\%p$  (protected division),  $\%2$ , and  $1/x$ .

In the SBLGP system (Table 5.9), a linear program consists of functions, which can use a global stack for temporary data storage. For that purpose the algorithm-building functions and arithmetic functions of TGP are easily adapted: The algorithm-building functions do not return any values on the stack and the arithmetic functions take their arguments from the stack and return the result on it. Stack-based iteration mechanisms replace the iteration structures from TGP. Furthermore, some stack-related functions like *pop* (to remove the top stack element) are added. To get to a degree constant memory requirement during the evolution, the program length is fixed. To still allow for shorter programs, a non-functional *noop* operator is used.

The SLLGP scheme (Table 5.9) abandons the iteration structures, which allow for scaling of quantum algorithms. As it is not necessary for the gates to share parameter values, stack and return values are dispensable. Thus, the function set can be reduced to the *noop* function and so-called encapsulated gates, which combine the gate type and its essential parameter values. SLLGP uses *ephemeral-random-quantum-gate*, a function which creates an encapsulated gate at random.

The fitness function is the same for all three GP schemes. The evaluation depends on three components: *misses*, which is the number of fitness cases in which the program failed, the total *error* of these fitness cases and the program *length*, and the number of gates in the quantum algorithm. Here, 'failed' means that the probability for a correct

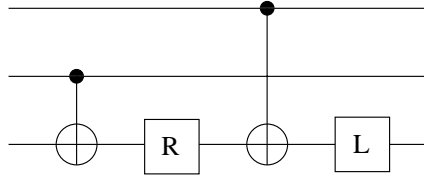


Figure 5.6: An efficient ‘receive circuit’ found by GP.

Algorithm:	generational GP
Program Structure:	tree
Initialization:	ramped half-and-half method
Function Set:	algorithm-building functions for a set of gates; iteration control structures; arithmetic functions
Terminal Set:	system constants: number of qubits, number of input/output qubits; random floating point constants and other useful constants (0, 1, 2, $\pi$ , $i$ )
Representation:	functions/operations in prefix notation
Fitness Function:	standardized, 3-component ( <i>misses, error, length</i> ) function with lexicographic ordering
Selection:	tournament
Genetic Operators:	reproduction, tree crossover (at any point/ at function points), mutation
Termination:	max. generations exceeded or threshold fitness achieved

Table 5.8: The standard TGP algorithm by Spector et al., used to evolve solutions for Deutsch’s two-bit early promise problem and the scaling majority-on problem.

Algorithm:	generational GP
Program Structure:	linear, stack-based (SBLGP) or stackless (SLLGP)
Function Set:	SBLGP: algorithm-building functions for a set of gates, incl. <i>noop</i> , arithmetic functions, stack-related functions ( <i>pop</i> , etc.), iteration control structures (optional); SLLGP: <i>ephemeral-random-quantum-gate</i> , <i>noop</i>
Terminal Set:	SBLGP: random floating point constants and other useful constants ( $0, 1, 2, \pi, i$ )
Fitness Function:	standardized, 3-component ( <i>misses, error, length</i> ) function with lexicographic ordering
Selection:	tournament
Genetic Operators:	several linear crossover and mutation operators
Termination:	max. generations exceeded or threshold fitness achieved

Table 5.9: Both stack-based and stackless linear GP (SBLGP/SLLGP) algorithms, applied to solve the quantum four-item database search problem and the two-bit-and-or problem.

output is below 0.48. The three components are used in the above mentioned order (misses (most significant), error, length (least significant)) in a standardized lexicographic fitness function.

Without doing a thorough comparison, Spector et al. point out some pros and cons of the three GP schemes. The tree structure of individuals in TGP simplifies the evolution of scalable quantum circuits as it seems to be predestined “for adaptive determination of program size and shape” [140]. For that reason, in addition to constants for the qubit operands and some parameter values, the terminal set also contains a constant for the number of qubits in the quantum system. Then, the scalability of a quantum program can be tested with several input values for the number of qubits. A disadvantage of the tree representation is its higher costs of time, space and complexity. Furthermore, possible return-value/side-effect interactions may make evolution more complicated for GP. The linear representation in SBLGP/SLLGP seems to be better suited to evolution because the quantum algorithms are themselves sequential. Moreover, the genetic operators in linear GP are simpler to implement and the memory requirements are perspicuously reduced compared to TGP. The return-value/side-effect interaction is eliminated in SBLGP, as the algorithm-building functions do not return any values. Overall, Spector et al. state that applied to their chosen problems, results appeared to emerge more quickly with SBLGP than with TGP. If scalability of the quantum algorithms is not so important, the SLLGP approach is to be preferred.

Some parameter settings for runs on the different problems are listed in tables 5.10 and 5.11.

Concerning the evolved results, it is noted that though the best evolved quantum circuit for Deutsch's two-bit early promise problem is better than classical, it is clearly worse than Deutsch's algorithm (max. error prob. is 0.3). For the scaling majority-on problem, the GP system found a quantum algorithm comparable to a probabilistic classical algorithm, but not better than classical. The solution found for the 4-item database search problem essentially matched Grover's algorithm, except for some unimportant algebraic sign changes for some computational basis states. A better-than-classical quantum algorithm could be evolved for the two-bit and-or problem.

Population Size:	10000
Max. Generations:	1001
Max. Tree Depth of New Individuals:	6
Max. Depth of New Subtrees for Mutants:	4
Max. Depth after Crossover:	12
Reproduction Prob.:	0.2
Crossover Prob. (any points):	0.1
Crossover Prob. (function points):	0.5
Mutation Prob.:	0.2
Tournament Selection Size:	5

Table 5.10: Parameter settings in the GP algorithms for Deutsch's two-bit early promise problem and the scaling majority-on problem for  $n$ -bit oracles with  $n = 1 \dots 4$ .

	4-DBSP	2-AND/OR
Max. Generations:	1001	1000
Population Size:	1000	100
Max. Number of Gates:	256	32
Reproduction Prob.:	0.5	0.2
Crossover Prob.:	0.1	0.4
Mutation Prob.:	0.4	0.4
Tournament Selection Size:	5	8

Table 5.11: GP parameters for runs on the quantum four-item database search problem (4-DBSP) and the two-bit-and-or problem (2-AND/OR).



Spector et al. improved their stackless linear GP system. Their newer system, presented in [139] and [10], is steady-state<sup>4</sup> and supports true variable-length genomes. The function set is composed of the following gates:  $H$ ,  $U(\theta) := Ry(-\theta)$ ,  $U_2[\alpha, \theta, \phi, \psi]$  (general one-qubit gate in  $Z$ - $Y$  decomposition),  $CNOT$ ,  $CPH(\psi)$ ,  $ORACLE$  (or  $U_f$  respectively),  $M0$ ,  $M1$ ,  $M$ . The measurement gates  $Mx$  terminate the computation if the result  $x$  is obtained. The fitness components were completed by *expected-queries*, the number of oracle queries being expected, averaged over all fitness cases. Genetic Operators included in the system are

- Reproduction: produces a new copy of an individual;
- Crossover: produces a new individual by appending an initial segment of one parent to a tail segment of the other parent;
- Mutation: substitutes a single instruction (gate) by a randomly generated instruction;
- Insertion: inserts a gate sequence, randomly created or chosen from another individual, in a given individual;
- Deletion: removes a gate sequence from a given individual;
- Angle-Mutation: works only on gates with angle parameters;
- Minimization: each gate in the individual is checked, whether it can be removed to achieve a better fitness;
- Pair-Minimization: like minimization, but examines every pair of gates for possible removal;
- Multiple-Angle-Perturbation: creates a new individual by adding small constants to a sample of angle parameters randomly chosen from another individual

Both minimization and pair-minimization perform a local hill-climbing search, which is very expensive, but whose profit is not sufficiently proven. Moreover, the evolution can be distributed across a cluster of workstations. Applied to the two-bit-and-or problem, the new GP scheme found an improved quantum algorithm. A hand-tuned version is illustrated in Figure 5.7.

Lee Spector compiles these and additional results in his book [138] which will probably be available later on this year. There, he intensively deals with the application of genetic programming to the automatic programming of quantum computers.

---

<sup>4</sup>Note: The GP systems described above are supposed to be generational, not steady-state, but it is not specified how the tournament selection is implemented in this context.

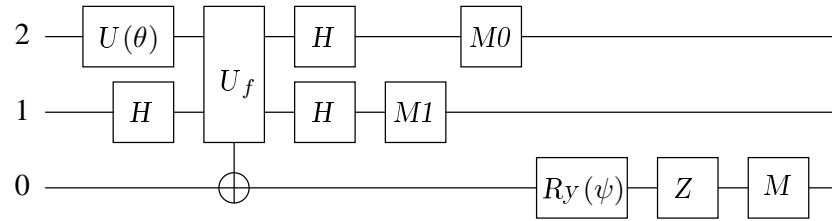


Figure 5.7: Quantum circuit for the 2-AND/OR-problem, with  $\theta = \pi/4$  and  $\psi = 0.0749\dots$

### 5.3.3 Rubinstein's GP Scheme

In [127], another GP scheme is presented and its working demonstrated by generating quantum circuits for the production of between two and five maximally entangled qubits (Figure 5.8) in the form  $1/\sqrt{2}(|00\dots 0\rangle + |11\dots 1\rangle)$ . In this scheme, gates are represented by a gate type and by bit-strings coding the qubit operands and gate parameters. Qubit operands and parameters have to be interpreted appropriate to the gate type. Assigning a binary key to each gate type, the gate representation is completely based on bit strings.

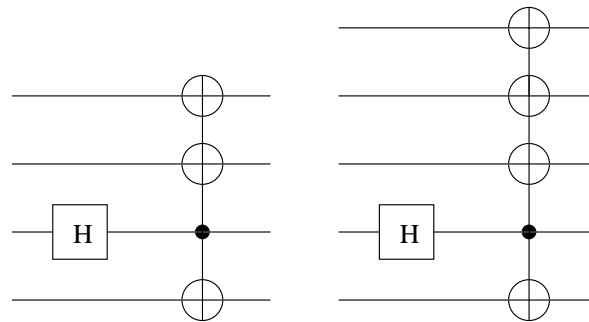


Figure 5.8: Quantum circuits for the four and five-entangled qubit problem

Two examples: assuming a 3-qubit circuit and a function set of eight quantum gates, the bit-string 011 100 001 might decode to the *CNOT* gate (011) with the leftmost qubit as the control qubit (100) and the rightmost qubit as the target (001) whereas the bit-string 100 010 1100101 might decode to a simple rotation gate *RY* (100) acting on the middle qubit (010) with a real-valued parameter (1100101) specifying the rotation angle when mapped to a specific interval, e. g.  $[-\pi, \pi]$ .

The genetic operators, applied in this GP scheme are:

- mutation of a quantum gate by replacing it with a new random gate;
- crossover between two parent circuits, i. e. swapping all gates between the parents

after randomly chosen gates, the crossing points, in the parent circuits (standard linear crossover);

- crossover between binary strings of like structure (qubit or parameter strings), i. e. swapping the bits between two strings after a randomly chosen crossing point.

The two crossover operators are combined using *uniform* and *weighted* crossover. In uniform crossover, a gate, a qubit operand or a parameter are chosen at random from a parent circuit, deciding which crossover operator has to be applied. In weighted crossover, a gate, qubit operand (normal, target or control qubit) and parameter binary string have associated weights for the random selection.

A fitness case consists of the input quantum register and the corresponding desired output quantum register. A measure for the fitness of an individual is the *error* (see Table 5.12), the sum of magnitudes of the differences between the amplitudes of the outcome register and the desired output for all fitness cases. To get a standardized fitness function, i. e. the fitness value lies in the range  $[0, 1]$ , the calculated error is divided by the error of the worst individual so far.

Further specifications of the GP algorithm are shown in Table 5.12, some parameter settings for the GP runs on the  $n$ -entangled qubit problem are listed in Table 5.13.

Algorithm:	generational GP
Program Structure:	linear
Function Set:	adequate finite quantum gate set (incl. measurement gate)
Representation:	bit-strings
Terminal Set:	number of qubits $n$ and the bit length for gate parameters
Fitness Function:	$error = \sum_i \sum_{j=0}^{2^n-1}  o_{ij} - d_{ij} $ , where $o_{i*}$ is the amplitude of the outcome register and $d_{i*}$ the amplitude of the desired output register for each fitness case $i$
Selection:	roulette wheel selection
Genetic Operators:	mutation (with very small probability) gate crossover (linear crossover) qubit and parameter bit-string crossover (fixed length crossover)
Termination:	max. generations exceeded or threshold fitness achieved

Table 5.12: The GP scheme by Rubinstein, applied to evolve maximally entangled quantum states.

Generations:	50
Population Size:	5000
n (Number of Qubits):	2...5
Max. Number of Gates:	3
Threshold Fitness:	0.001
Crossover Prob.:	0.8
Mutation Prob.:	0.01
Kind of Crossover:	uniform
Quantum Gates:	<i>ID, H, NOT, CNOT, NOT-CCNOT, CPH(<math>\theta</math>), RY(<math>\theta</math>), M</i>

Table 5.13: Some parameter settings in Rubinstein’s GP scheme for the  $n$ -entangled qubit problem.

### 5.3.4 GAs for Quantum Circuit Design according to Yabuki & Iba

In [168], Yabuki and Iba present a genetic algorithm<sup>5</sup> to evolve a quantum circuit for quantum teleportation (cf. Section 2.8.2). They use the same set of gates  $\{CNOT, L, R\}$  as Williams and Gray. Quantum circuit individuals are represented by strings of fixed length (genes) over a four letter alphabet. The string is composed of codons each consisting of three letters. One of the four letters is used to separate the string into three parts, provided that it is found at the first position of a codon: One part for the EPR pair generation, one for Alice (A) and one for Bob (B). The second occurrence of this letter, separating between A’s “send”- and B’s “receive”-circuit, is always connected to a single measurement of the first and second qubit at the end of A’s part. Other occurrences are without any relevance. The interpretation of a codon as a quantum gate depends on the part the codon is in. Further constraints were given, for example that in the EPR-part only the zeroth and first qubit and in A’s part only the first and the second qubit can be operated.

Further specifications of the GA algorithm are shown in Table 5.14. Some parameter settings for the GA runs are listed in Table 5.15.

The simplest evolved quantum circuit for quantum teleportation uses eight gates including A’s measurement, two for generating the EPR pair, and three gates each for the “send”- and “receive”-circuit.

---

<sup>5</sup>The main difference between GP and GA is the representation of individuals, which is modeled on genes, and their fixed length.

Algorithm:	fixed length GA
Program Structure:	linear
Function Set:	adequate (problem specific) finite quantum gate set;
Representation:	string over a four letter alphabet, composed of three letter codons, each codon represents a specific gate, different codons may encode identical gates
Terminal Set:	the initial system state; for the particular problem: $(p 0\rangle + q 1\rangle) \otimes  0\rangle \otimes  0\rangle$ ;
Fitness Function:	problem specific; for the particular problem: $f = 1/(1 + 10 \sum_j error_j)$ , where $error_j$ is a certain error function rating the difference between the actual output and the desired output for each fitness case.
Selection:	roulette wheel selection
Genetic Operators:	mutation two-point crossover
Termination:	max. generations exceeded

Table 5.14: GA scheme by Yabuki and Iba, used to evolve a quantum teleportation circuit.

Generations:	1000
Population Size:	5000
n (Number of Qubits):	3
String (Gene) length:	$\leq 150$ (corresponds to $\leq 50$ gates)
Crossover Prob.:	0.7
Mutation Prob.:	$1/(\text{gene length})$
Quantum Gates:	$\{CNOT, L, R\}$ ; a single measurement is implicitly provided

Table 5.15: Some parameter settings for teleportation circuit evolution.

## 5.4 Implemented GP Systems for Quantum Circuit Evolution

In the course of this thesis, two GP systems were implemented in the programming language C++, a linear and a linear tree GP. This section specifies these GP systems according to their genome structure and describes the implemented genetic operators. Furthermore, it gives some additional information on the evaluation of individuals and the used fitness functions. Special implementations such as additional selection mechanisms which are part of certain explorations are explained in the corresponding Sections.

Two other GP systems were implemented, this thesis does not deal with this (but nevertheless they should be mentioned):

Inspired by [65, 94], a modified linear GP system was built to evolve *CNOT* circuits, i. e. circuits exclusively using *CNOT* gates or  $C^k$ *NOT* gates with  $k < n$  for circuits on  $n$  qubits. They can be identified as classical circuits, if they are exclusively applied to basis states. These gates are not universal for quantum computation but they suffice to build any classical Boolean circuit as the Toffoli gate (*CCNOT*) is universal for classical reversible computation. Some simple arithmetical circuits for addition and multiplication modulo  $N$  could be evolved.

Another GP system was implemented for matrix decomposition based on elementary quantum gates, such as they are used in NMR quantum computing. This was applied to the 2- and 3-qubit QFT resulting in suitable, experimentally implementable quantum algorithms.

### 5.4.1 Genome Structure - linear vs. linear-tree GP

Intermediate measurements in quantum circuits compare to conditional branchings in programming languages. Due to this, quantum circuits have a natural linear-tree structure. Linear-tree GP naturally supports the use of measurements as an intermediate step in quantum circuits. It was built to achieve more “degrees of freedom” in the construction and evolution of quantum circuits compared to stricter linear GP schemes (like in [127, 140]). Moreover, the linear-tree structure may simplify legibility and interpretation of quantum algorithms. A simple example, given in [107], is the quantum teleportation circuit, described in Section 2.8.2 and shown in Figure 2.15. In principle the teleportation circuit can be transformed into another circuit having no intermediate measurement in A’s part. In this case, the resulting circuit would hardly be interpreted as a quantum teleportation circuit.

However, the principle of deferred measurements suggests a purely sequential circuit structure. The advantage of the linear genome structure is the disadvantage of the linear-tree structure of quantum circuits, where higher costs for administration and fitness evaluation arise from the additional tree structure.

The structure of individuals in the linear-tree GP system consists of linear program segments, which are sequences of unitary quantum gates (instructions), and branchings caused by single qubit measurement with respect to the standard basis  $\{|0\rangle, |1\rangle\}$ . Depending on the measurement result, ‘0’ or ‘1’, the corresponding linear program branch, the ‘0’- or ‘1’-branch is executed. That is, the measurement gate is employed to conditionally control subsequent quantum gates, like an “if-then-else”-construct in a programming language. Since measurement results occur with certain probabilities, usually both branches have to be evaluated. Therefore, the quantum gates in the ‘0’- and ‘1’-branch have to be applied to their respective post-measurement states. From the branching probabilities the probabilities for each final quantum state can be calculated.

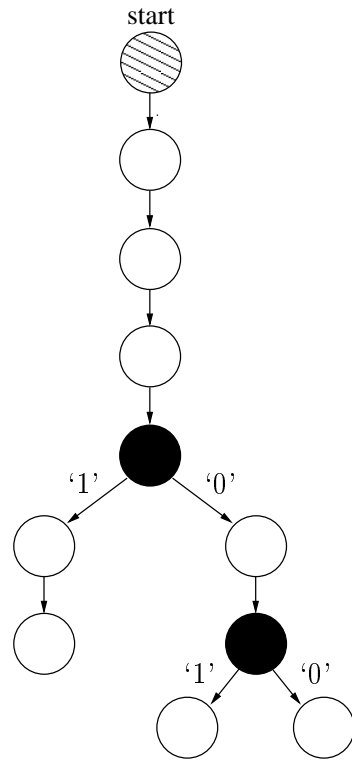
For an arbitrary individual, information about its structure is administered, including the start and the branching nodes, the length of their linear subprograms (‘0’- and ‘1’-branch), the number of branchings and the total program length (number of unitary gates plus measurements). The start node is a special node without any functional meaning. By convention, the start node has (only) a ‘0’-branch and the ‘1’-length is set to 0. This information is helpful for implementing the linear and tree crossover operators. Figure 5.9 illustrates the genome structure and shows the information stored for the exemplary individual. The maximum number of possible branches is set by a global system parameter; without using any measurement gates genomes have just a linear structure. Nevertheless, also an independent merely linear GP system without any structural overhead was implemented. The overall length or size of a quantum circuit, i. e. the number of quantum gates, is limited by a global parameter value as well.

### 5.4.2 Genetic Operators




For the linear GP scheme a crossover operator and different mutation operators are implemented. The linear crossover operator is already described in Section 3.3.2, Figure 3.7. The following mutation operators used are

- random *deletion* of a single quantum gate in the quantum circuit,
- *insertion* of a single quantum gate randomly chosen from the pool of quantum gates at a random position,
- random *replacement* of a single quantum gate by a randomly chosen quantum gate from the pool,
- *alteration* of parameters of a randomly chosen single quantum gate (if a gate has more than one parameter, the parameter to be mutated is chosen randomly), and
- random *swap* of two neighboring quantum gates in the circuit.

Crossover and mutation rate can be set by global system parameters. Either both genetic operators are applied or only one of them which is controlled by another system



(a) Program structure.

structural information:			
no.	0	1	2
knots			
'0'-length:	3	1	1
'1'-length:	0	2	1
program length:	10		
number of branchings:	2		

(b) Stored program information.

Figure 5.9: Individual with linear-tree genome. Solid circles represent single qubit measurements, open circles correspond to instructions and the pattern filled circle marks the start of the quantum program.



parameter. Which mutation operator is applied is determined at random considering the different number of possibilities for each mutation operator. That means, for example that a *certain* swap operation of  $L - 1$  possible swap operations in a circuit of length  $L$  is as likely as any of the possible  $L$  deletions or replacements. Therefore, on average a replacement will occur more often than a swap or a deletion.

The linear-tree GP scheme uses linear and tree crossover. For linear crossover, two randomly chosen linear program sequences are swapped. For tree crossover, in each individual a subtree — not necessarily the left or right subtree belonging to a branching instruction — is randomly chosen and completely swapped. The mutation operators implemented in linear-tree GP are just random deletion, random insertion and random replacement. Applied to a non-branching (non-measurement) instruction the operators act as in the linear GP scheme. If a measurement gate is deleted, then one of its subtrees is randomly chosen and deleted as well. If a measurement gate is inserted, then the remaining subtree is randomly connected either as the ‘0’- or the ‘1’-branch of the measurement. The replacement operator is implemented accordingly to these directions.

### 5.4.3 Fitness Cases and Fitness Functions

The fitness value decides whether an individual participates in the evolutionary process and, thus contributes or not to the evolution of a solution. The fitness value is calculated by a fitness function on the basis of fitness cases, which are pairs of inputs and the corresponding desired outputs of a certain given problem. There are many ways to define fitness cases, especially the outputs, and the fitness function which may lead to completely different search spaces and fitness landscapes which result from assigning each individual its fitness value. In the following, two of them are described exemplarily for the Deutsch-Jozsa problem (cf. Section 2.8.1). The GP systems can handle both descriptions of fitness cases and have implemented the appropriate fitness functions described below.

#### Specific Basis State Representation

Desired outputs for problem instances of the Deutsch-Jozsa problem (and also 1-SAT) can be defined as certain basis states or superpositions of certain basis states representing a solution (or several solutions as in the case of 1-SAT) to the problem. For example, for  $n = 2$  (three qubits), any superposition state  $\sum_{x=0}^3 \alpha_x |x\rangle |1\rangle$  with  $\sum_x |\alpha_x|^2 = 1$  might represent a constant function while any other state represents a balanced function. Then, measuring the state would lead to a basis state  $|--1\rangle$  for the two constant and  $|--0\rangle$  for the six balanced function, where a ‘-’ (don’t care) allows any replacement by 0 or 1. An individual’s fitness is basically determined by the probabilities for each fitness case of measuring a wrong basis state, i. e.,  $|--0\rangle$  when a basis state  $|--1\rangle$  is demanded and vice versa.

The fitness function takes a few specific values of the individual’s evaluation into account:

- the number of *misses*  $M$ , that is, the number of fitness cases, where the probability for a correct result of the individual is below a certain threshold value, e. g., 0.52,
- the *maximum error*  $\epsilon_{max}$ , which is the maximum probability for an incorrect result over all fitness cases,
- the *total error*  $\epsilon_{acc}$ , which is the error accumulated over all fitness cases, and
- the *number of quantum gates*,  $num$ , the individual consists of; this is optionally used to force the evolution to find smaller solutions.

These *fitness components* enter the fitness function with varying weights. The number of gates is the input parameter of a special penalty function  $p(num)$ . It is defined as

$$p(num) = (1/\pi) \cdot [\arctan(0.5 \cdot (num - ip)) + \arctan(0.5 \cdot ip)],$$

where  $ip$  is a system parameter fixing the inflection point and shifting the function along the  $x$ -axis if altered. Figure 5.10 shows a typical graph of the penalty function. Of course, other penalty functions are conceivable. As is the case for this function, the penalty should be still very small and increase slowly for circuit lengths as long as they are below a certain threshold. Higher penalty values put enhanced pressure to the evolution of small individuals but simultaneously reduce the pressure to the evolution of good solutions. That is, the larger the penalty value the higher the risk that the evolution favors smaller individuals above better but larger individuals.

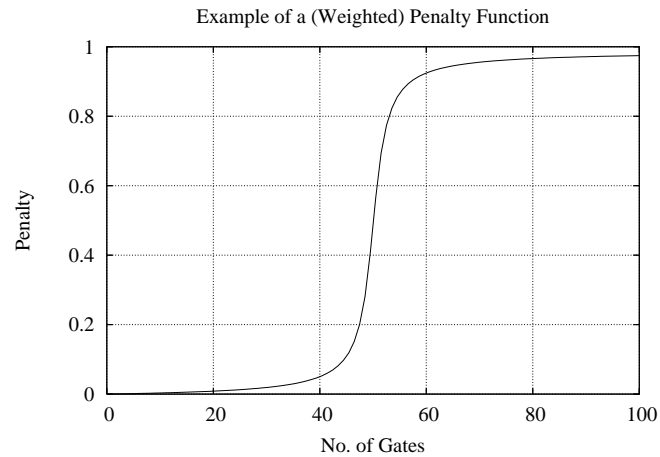


Figure 5.10: Example for a penalty function already weighted with 0.1 to illustrate the real penalty. The system parameter  $ip$  defines the position of the inflection point. The plot shows  $0.1 \cdot p(num)$  for  $ip = 50$ .

The fitness function then reads as follows:

$$f = M/c \cdot 0.4 + \epsilon_{max} \cdot 0.3 + \epsilon_{acc}/c \cdot 0.2 + p(num) \cdot 0.1 \quad (5.1)$$

where the values of  $M$ ,  $\epsilon_{max}$ ,  $\epsilon_{acc}$  and  $num$  relate to a given individual and  $c$  is the number of fitness cases. The fitness function is standardized and normalized.

The idea of using fitness components is adopted from [140]. Instead of combining these components in lexicographic ordering as recommended in [140], the fitness value is calculated as the sum of the weighted components. The weighting is developed by experience. Although other values might be suitable, it is a useful prioritization of the components. The components *misses*, *maximum error* and *total error* are not independent of each other. The three goals in the optimization are to some extent complementary. A necessary criterion for an individual to be a solution is that the number of misses is zero. Therefore, the number of misses gets the highest weight. The maximum error is weighted higher than the total error. In doing so, solutions with smaller maximum error are favored above those with a smaller total error. Since too much emphasis on minimal solutions is counterproductive to evolution, the weight of the number of quantum gates is rather small.

### Assessing Classification

Actually, the Deutsch-Jozsa problem is a classification problem, that is, a solution has to distinguish inputs of several different classes. In the case of the Deutsch-Jozsa problem, these are the class of constant Boolean functions ( $A$ ) and the class of balanced Boolean functions ( $B$ ).

A fitness case for a Deutsch-Jozsa problem instance then consists of a Boolean function  $f$ , represented by a suitable oracle matrix  $U_f$  (cf. Section 2.4.5), and a tag determining the property of the function and therefore the class this fitness case belongs to. The fitness of a quantum program is then calculated according to the program's quality to identify the class of every input (for DJ: the property of a given function  $f$ ).

To obtain a fitness value, a quantum circuit is evaluated for each input matrix. For this purpose, the sequence of gates is applied to the initial basis state resulting in a final quantum state. The initial basis state is  $|0\dots 0\rangle$  by default, but this can also be changed from fitness case to fitness case. For the Deutsch-Jozsa problem, there is no need for other initial bases states. After circuit evaluation, the results, i. e., the probability vectors corresponding to fitness cases of each class are compared with the probability vectors corresponding to the fitness cases of every other class. In case of the Deutsch-Jozsa problem, the outputs corresponding to balanced functions are compared with those corresponding to the two constant functions. On the basis of the measurement probabilities (for every basis state) the determinability of the classification by the quantum algorithm is quantified by the fitness value.<sup>6</sup>

---

<sup>6</sup>A similar fitness approach is implemented by R. Stadelhofer in his GP system for NMR-based quantum circuit design [141].

Going into detail, the probability vector coefficients  $p_1[x]$  and  $p_2[x]$  corresponding to base state  $|x\rangle$  are compared to  $p = 1/2^n$ , which is the probability for measuring any  $n$ -qubit basis state of an equally weighted superposition. If both coefficients are larger or equal to  $p$ , this is interpreted as a *misclassification*, since for both fitness cases there is a reasonable probability of measuring the same result  $|x\rangle$ , which would not make both cases distinguishable. Misclassifications are counted only once for a pair of fitness cases. However, the probabilities are also added to the *undecidability* value of the corresponding fitness case. It is also rated as an undecidability when both probabilities  $p_1[x]$  and  $p_2[x]$  are smaller than  $p$ . Added up over all  $x$ , the larger amount of the two undecidability values for the respective fitness cases is added to an overall undecidability value. If one probability is larger and the other smaller than  $p$ , the outcome  $|x\rangle$  might be a distinctive characteristic of the two fitness cases. But there might be still an error, which is summed up in a separate variable. The whole calculation is summarized in the following algorithm.

### Algorithm

(\*  $Cl$  is the set of all classes;  $c_1$  and  $c_2$  are fitness cases from two of these classes ( $C_1$  and  $C_2$ );  $p_1$  and  $p_2$  are the  $2^n$ -dimensional probability vectors resulting from the evaluation of the given quantum circuit for the two fitness cases  $c_1$  and  $c_2$ ;  $p$  is a constant representing a threshold value for misclassification and undecidability; misclassifications are notified in *misclass* and summed up in the variable *miss*; *undec<sub>1</sub>*, *undec<sub>2</sub>* are the undecidabilities for every two compared fitness cases; *undec* is the sum of the maximum of undecidabilities over all pairs of fitness cases of different classes; *err* stores the error, summed up over all pairs of fitness cases of different classes; \*)

1.  $p \leftarrow 1/2^n$
2. **for all**  $\{C_1, C_2\}$  with  $C_1, C_2 \in Cl, C_1 \neq C_2$
3.     **do** (\* for all two different classes \*)
4.         **for all**  $(c_1, c_2) \in C_1 \times C_2$
5.             **do** (\* for all pairs of fitness cases from  $C_1$  and  $C_2$  \*)
6.                 *undec<sub>1</sub>*  $\leftarrow 0$
7.                 *undec<sub>2</sub>*  $\leftarrow 0$
8.                 *misclass*  $\leftarrow false$
9.                 **for**  $x \leftarrow 0$  **to**  $2^n - 1$
10.                     **do** (\* for all possible basis states \*)
11.                         **if**  $(p_1[x] \geq p)$  **and**  $(p_2[x] \geq p)$
12.                             **then** *misclass*  $\leftarrow true$
13.                                 *undec<sub>1</sub>*  $\leftarrow undec_1 + p_1[x]$
14.                                 *undec<sub>2</sub>*  $\leftarrow undec_2 + p_2[x]$
15.                         **if**  $(p_1[x] < p)$  **and**  $(p_2[x] < p)$
16.                             **then** *undec<sub>1</sub>*  $\leftarrow undec_1 + p_1[x]$
17.                                 *undec<sub>2</sub>*  $\leftarrow undec_2 + p_2[x]$

```

18.           if ( $p_1[x] < p$ ) and ( $p_2[x] \geq p$ )
19.             then  $err \leftarrow err + p_1[x]$ 
20.           if ( $p_1[x] \geq p$ ) and ( $p_2[x] < p$ )
21.             then  $err \leftarrow err + p_2[x]$ 
22.         if ( $undec_1 > undec_2$ )
23.           then  $undec \leftarrow undec + undec_1$ 
24.           else  $undec \leftarrow undec + undec_2$ 
25.         if ( $misclass$ )
26.           then  $miss ++$ 

```

Finally, concerning a certain individual, the number of pairs of fitness cases where misclassification is detected (*miss*), the undecidability value (*undec*), the total error (*err*) and the number of gates (*num*) enter the fitness function. The first three components are normalized, i. e., divided by the number of fitness case comparisons *c*. The number of gates enters the penalty function  $p(num)$ , and all four resulting values are weighted. The fitness function reads then as follows:

$$f = miss/c \cdot 0.4 + undec/c \cdot 0.3 + err/c \cdot 0.2 + p(num) \cdot 0.1 \quad (5.2)$$

Here, the weighting is chosen as it is already used in the fitness function previously introduced (Equation 5.1). A higher number of quantum gates in a genotype leads to a slightly increased fitness value due to the size penalty. The fitness function is standardized. The values of the fitness components are normalized, as is the fitness function. However, some fitness components depend on each other, excluding that they both get their highest possible value at the same time. This is the reason why for the “classification assessment” 0.7 is the highest possible value (neglecting the size penalty): The number of misclassifications is at maximum equal to *c* and each misclassification contributes an undecidability of 1. In this case, there are no other errors adding to the overall fitness value.

The threshold *p*, the detection of misclassifications is based on, is debatable. Consider for example the case that  $p_1[0] = p_2[0] = p = 1/2^n$ ,  $p_1[1] = 1 - p$ ,  $p_2[2] = 1 - p$  for a pair of fitness cases from different classes. Then, this would be counted as a misclassification although for large *n* the probability for a correct classification or differentiation of these two fitness cases is close to 1. However, for small numbers of *n* as they are used in the experiments described below ( $n \leq 4$ ), the value of *p* is large enough to not exclude problem solutions from being evolved. In the experiments concerning the evolution of quantum circuits for the Deutsch-Jozsa problem this fitness function was successfully applied. Yet, for further experiments (with larger *n*) the fitness function should define misclassifications in a different way or even omit them and assess the quality of classification only on the basis of undecidability.

### Training Sets and Early Termination

Usually, a GP-system works on a training set of fitness cases to evolve a proper solution of a given problem. To appraise the true quality of the best evolved solutions, they are applied to a test set of fitness cases not included in the training set. The GP systems used for the investigations in the course of this thesis do not distinguish between training and test sets of fitness cases, i. e., there is no assessment step other than fitness calculation based on the training set.

It should be the aim to find solutions for any possible inputs and not only for some subsets. This applies especially to those problems where proper quantum algorithms already exist. Anything else than finding true solutions would be very unsatisfactory. Selecting all possible fitness cases as the training set ensures, unfortunately at great expense, that an evolved quantum circuit is a solution of the problem and not only for certain fitness cases.<sup>7</sup>

Some evolutionary runs made for DJ and 1-SAT using small random subsets of the entire set of fitness cases did not lead to quantum circuits properly solving *all* possible fitness cases. Yet, these experiments do not have statistical evidence due to the small number of runs. However, they indicate that evolution of good solutions is difficult when using only small subsets of fitness cases for certain problems and that the selection of training sets is a demanding task. Conversely, for some other problems like quantum teleportation it is possible to evolve universally valid solutions (quantum circuits) using only a rather limited number of fitness cases even though the set of all fitness cases is infinite [168]. It is apparently a characteristic of the problem whether and how fitness cases build “groups of representatives”, comparable to equivalence classes, where for each group one fitness case is as “good” as any other to be chosen for the training set, but where also a selection of representatives of each class is necessary for a successful evolution of a problem solution.

Entailing huge costs, large numbers of fitness case evaluations per individual for fitness calculation are a general problem in GP. Consequently, different *subset selection* methods were developed allowing an individual’s fitness to be calculated only on a small subset of all fitness cases, namely *historical subset selection* [56], *random subset selection* [56], *stochastic sampling* [112, 8], *stochastic subset sampling* [95, 96] (based on a fitness case topology), *dynamic subset selection* (DSS) [56] and *active data selection* [173].

In the following paragraph, DSS is explained briefly, since it is already implemented as an optional enhancement in the linear GP system. A small number of experiments was already performed. The evolutionary runs were not promising, however, for a detailed analysis more runs are necessary.

DSS assigns to every fitness case  $i$  a difficulty  $D_i$  and an age  $A_i$ . The difficulty is increased every time an individual fails on that particular fitness case, while the age is increased every time the fitness case is not part of the training (sub-)set. Both, difficulty

---

<sup>7</sup>Of course, using this approach there is no need for a test set as there are no fitness cases left.

and age are reset to zero if the fitness case is selected to be in the subset. The update is done in every generation<sup>8</sup>. The sum of  $A_i$  and  $D_i$ , each taken to the power of certain parameters  $a$  and  $d$ , works as a weight  $W_i = A_i^a + D_i^d$ . This weight determines the probability  $P_i$  of fitness case  $i$  of being selected for a subset of fixed size  $S$  from  $T$  fitness cases in total:  $P_i = W_i \cdot S / \sum_{j=1}^T W_j$ . This method assumes that it is of benefit to focus on those cases, which are frequently “miscalculated” and which have not been selected for several generations or tournaments.

On one hand for some problems it seems to be necessary to evaluate individuals for almost all fitness cases to evolve proper quantum circuits, on the other hand, as already discussed in Section 5.2.2, if individuals are evaluated for all fitness cases, the huge number of fitness cases makes evolution very time consuming. Therefore, it is very important to reduce the number of fitness case evaluations for individuals for which an exhaustive evaluation is not necessary, as they do not represent solutions. This is done using the *early termination* method, which means that after a certain number of fitness cases on which the individual fails a further evaluation of the individual is abandoned. The individual’s fitness is then set to a value dependent on the number of fitness cases. This saves time, wasted in the evaluation of “bad” individuals. Both fitness calculation methods described above can use early termination, however, when using the classification approach it should be guaranteed that constantly all classes are represented in the subset selection. Concerning this, up to now there are not any experiments done yet.

Other methods for subset selection and reduction of fitness case evaluations are not implemented. Combined with the comparison of these approaches this should be a further step. There seems to be a large scope for speeding up evolution, which is highly important for evolutionary quantum circuit design.

---

<sup>8</sup>For steady-state GP this might happen after a certain number of tournaments.





## 6 Results and Analyses

*Making intelligent mistakes is a great art.*

---

Federico Fellini (1920–1993)

This chapter comprises the most important results of evolutionary runs performed in the course of this thesis and further experiments concerning different aspects of quantum circuit evolution using genetic programming:

Quantum circuits have a natural linear structure. If one takes intermediate measurements (which compare to conditional branchings) into consideration, this structure becomes linear-tree. Whether the use of intermediate measurements is useful or even permissible<sup>1</sup> seems to be still under debate [65]. In Section 6.1 the linear and linear-tree GP system are applied to instances of 1-SAT and the Deutsch-Jozsa problem. Aside from the question of whether intermediate measurements help evolve quantum algorithms, this section essentially deals with the scalability of quantum algorithms. Evolution often starts with random populations. However, to evolve scalable algorithms, the iterative approach using evolved solutions for smaller problem instances to generate the initial population for the next larger instance turns out to be very effective. Another aspect of evolution is the choice of genetic operators. In many GP systems, the crossover operator is important for a rapid evolution of good solutions. As shown in this section (and Section 6.2), there seems to be no benefit of crossover for the evolution of quantum algorithms.

Section 6.2 presents a study of search spaces and fitness landscapes in the context of the evolution of quantum programs. The relationship between landscape characteristics and quantum algorithm evolution can be useful for improving the efficiency of the search process. Problem instances of the DJ and 1-SAT are considered as a starting point for the exploration of search spaces of quantum algorithms. The structure of fitness landscapes is analyzed using autocorrelation functions and information measures.

In Section 6.3 different selection strategies are analyzed, such as tournament selection,  $(\mu, \lambda)$  and  $(\mu + \lambda)$  ES selection, which are applied to the Deutsch-Jozsa problem and the 1-SAT problem using the linear GP system. Moreover, it is demonstrated that additional randomness added to the selection mechanism of a  $(1,10)$  selection strategy can boost the evolution of quantum algorithms on particular problems.

---

<sup>1</sup>Intermediate measurements mostly change pure states into mixed states [65].

There is an abundance of adjustable parameters in the GP systems which are not investigated. In this sense, the following explorations are only a part of what could be analyzed in the context of evolutionary quantum circuit design and return mostly “experiences” acquired in dealing with the implemented GP systems. In many cases, parameter values are adjusted due to experience gathered in evolutionary runs done in the past. As a result of a continual development of the GP systems, “incompatibilities” arose. For example, from the beginning, the progress in evolution (using tournament selection) was measured according to the number of tournaments. To achieve a more accurate measure and to make the resulting plots comparable to the outcomes of evolutionary runs using other selection methods, progress in evolution was measured later on according to the number of individual evaluations. However, this does not affect the results and their interpretation at all.

## 6.1 Evolution and Scalability

A quantum algorithm is *scalable* if the size of the problem can be increased while the efficiency of the algorithm is maintained. In general, a scalable quantum algorithm can be regarded as a “scheme” specifying quantum circuits which solve instances of a certain problem of any (but fixed) size. Following this scheme, a quantum algorithm for a small problem instance can be easily upgraded (with only slight modifications) to solve a larger problem instance. However, in scalable algorithms these changes affect the efficiency of the algorithm only insignificantly, that is, the complexity of the scalable algorithm is independent of the problem instance.

Two examples of a scalable algorithm are the Deutsch-Jozsa algorithm (cf. Section 2.8.1) and Hogg’s algorithm (cf. Section 4.2.5). A scalable quantum algorithm (its scheme) for DJ is shown in Figure 2.14 in Section 2.8.1. This algorithm does not only work on  $n$ - but also on  $n+1$ -bit functions  $f$ . The overall structure of the algorithm is still the same independent of the problem instance. Only another qubit and two additional Hadamard-gates are necessary to build the quantum circuit for the increased problem instance.

Scalability and detecting scalability of quantum algorithms is important, because as discussed in the Sections 5.1 and 5.2 evolution of quantum circuits is basically possible only for very few qubits, or small problem instances respectively. Hence, scalability can compensate for the restricted simulation of quantum circuits on conventional computers and the limited use of GP-based quantum circuit evolution on small quantum systems. Even if non-scalable quantum algorithms might be of interest, scalable quantum algorithms are more relevant and favored.

In this section results of the evolution of quantum circuits for the Deutsch-Jozsa problem ( $n = 1 \dots 3$ ) and 1-SAT ( $n = 2 \dots 4$ ) are presented. For each problem instance, quantum algorithms can be evolved using the linear-tree and linear GP system. It should

be emphasized that in contrast to the GP systems developed by Spector et al. (cf. Section 5.3.2) the linear and linear-tree GP system were not designed to *directly* evolve scalable quantum circuits. Nevertheless, evolution of quantum algorithms for problem instances of varying size may lead to “visibly” scalable quantum algorithms, as it is shown for the 1-SAT problem. In this context, it is also found that the mixing matrix in Hogg’s algorithm can be implemented more efficiently by using simple  $R_x$  gates. Moreover, using the best solutions of a problem instance of size  $n$  to generate the initial population for evolutionary runs for problem size  $n + 1$  boosts the evolution and helps to find scalable quantum algorithms. The positive effect of this kind of *pre-evolved* initial population is demonstrated for both 1-SAT and Deutsch-Jozsa using the linear GP system.

The experiments described in the following are done without any previous optimization of relevant GP parameters. For instance, the population size was often chosen wrongly for a good performance of the evolutionary runs (cf. Section 6.3). In addition, later investigations reveal that crossover affects the evolutionary process adversely and therefore should be omitted. Evolutionary runs supplying evidence are provided in the last part of this section. The impact of crossover is also addressed in Section 6.2 in terms of crossover landscapes. Statements as to the choice of the population size in tournament selection are made in Section 6.3.

Independent of the GP system, the selection strategy, and other aspects of the evolutionary process, the GP systems always stores the fitness of the best individual found so far. These data form the basis of all empirical results.

### 6.1.1 Evolving Quantum Circuits for 1-SAT

The 1-SAT problem for  $n$  variables is already described in Section 4.2.5. It is also discussed in Section 5.2.1 as a possible suitable problem for GP-based quantum circuit design.

The linear-tree GP system was applied to 1-SAT with  $n = 2 \dots 4$  variables. Each of the  $\sum_{k=1}^n \binom{n}{k} 2^k$  fitness cases consists of an input state (always  $|0\rangle^{\otimes n}$ ), an input matrix for the formula and the desired output, which is described in a certain basis state representation (cf. Section 5.4.3). For example,

$$(|00\rangle, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{pmatrix}, |-0\rangle)$$

is the fitness case for the 1-SAT formula  $\bar{v}_2$  in two variables  $v_1, v_2$ . The desired output state is described as  $|-0\rangle$ , since only the rightmost qubit is essential to the solutions  $\{v_1 = \text{true/false}, v_2 = \text{false}\}$ . That means, an equally weighted superposition of all solutions is not required.

GP Structure	linear-tree
Selection	tournament, elitist
Tournament Size	2
Population Size	5000
Basic Gate Types	$H, Rx, Ry, Rz, C^k NOT, INP, M$
Max. No. of Gates	10
Max. No. of Measurements	$n / 0^*$ )
Number of Input Gates	1
Mutation Rate	1
Crossover (XO) Rate	0.1
Linear XO Probability	$0.9 / 1^*$ )

Table 6.1: Parameter settings for the 1-SAT problem with  $n = 2 \dots 4$ . \*) indicates special settings for  $n = 4$ .

Table 6.1 gives some parameter settings for GP runs applied to the 1-SAT problem. However, after evolving solutions for  $n = 2$  and  $n = 3$ , intermediate measurements seemed to be irrelevant for searching 1-SAT quantum algorithms since all evolved solutions did not use them. Therefore, the evolutionary runs for  $n = 4$  did without using intermediate measurements. Without intermediate measurements (gate type  $M$ ), which constitute the tree structure of quantum circuits, tree crossover is not applicable. In GP runs for  $n = 2, 3$  the maximum number of intermediate measurements was limited by the number of qubits.

For the two-, three- and four-variable 1-SAT problem, 100 GP runs were done recording the best evolved quantum algorithm of each run. Finally, the overall best quantum algorithm was determined. For each problem instance, the linear-tree GP system evolved solutions (Figures 6.1 and 6.2) that are essentially identical to Hogg's algorithm. This can be seen at a glance when noting that  $U = Rx[3/4\pi]^{\otimes n}$ , which can be proven as follows:

$$Rx[3/4\pi]^{\otimes n} = \frac{1}{\sqrt{2}^n} \sum_{x,y} i^{|x \otimes y|} (-1)^{n-|x \otimes y|} |x\rangle \langle y| = \frac{(-1)^n}{\sqrt{2}^n} \sum_{x,y} (-i)^{|x \otimes y|} |x\rangle \langle y|$$

where  $|x \otimes y|$  is the number of bits that  $x$  and  $y$  differ in. Thus,

$$U_{x,y} := \frac{1}{\sqrt{2}^n} e^{i\pi(n-m)/4} (-i)^{|x \otimes y|} = e^{i\pi(n-m)/4} (-1)^n (Rx[3/4\pi]^{\otimes n})_{x,y}$$

and  $U$  is equal to  $Rx[3/4\pi]^{\otimes n}$  up to a global phase factor which of course has no influence on the final measurement results.

Differences in fitness values of the best algorithms of each GP run are negligible, though they differ in length and structure, i. e., in the arrangement of gate-types. Details of the

```

Misses:                0
Max. Error:            8.7062e-05
Total Error:           0.0015671
Oracle Number:         1
Gate Number:           10
Fitness Value:         0.00025009

Individual:
H      0
H      1
H      2
INP
RX     6.1083    0
RX     2.6001    0
RX     3.0818    0
RX     2.3577    1
RX     2.3562    2
RZ     0.4019    1

```

Figure 6.1: Extracted from the GP system output: After 100 runs this individual was the best evolved solution to 1-SAT with three variables. Here, INP denotes the specific input matrix  $R$ .

```

H 0
H 1
H 2
H 3
INP
Rx[3/4 Pi] 0
Rx[3/4 Pi] 1
Rx[3/4 Pi] 2
Rx[3/4 Pi] 3

```

Figure 6.2: The three best, slightly hand-tuned quantum algorithms to 1-SAT with  $n = 2, 3, 4$  (from left to right) after 100 evolutionary runs each. Manual postprocessing was used to eliminate introns, i.e. gates which have no influence on the quantum algorithm or the final measurement results respectively, and to combine two or more rotation gates of the same sort into one single gate. Here, the angle parameters are stated more precisely in fractions of  $\pi$ . INP denotes the input gate  $R$  as specified in the text. Without knowledge of Hogg's quantum algorithm, there would be strong evidence for the scalability of this evolved algorithm.

performance and convergence of the averaged fitness values over all GP runs can be seen in the three graphs of Figure 6.3 and 6.4.

Further GP runs with different parameter settings hint at other strong parameter dependencies. For example, the adequate limitation of the maximum number of gates leads more rapidly to good quantum algorithms. In contrast, stronger limitations (somewhat above the length of the best evolved quantum algorithm) make convergence of the evolutionary process more difficult. Some experiments addressed different gate sets. Unfortunately, for larger gate sets “visible” scalability is not detectable. Due to the small number of these runs with other parameter settings the results do not have any statistical validity and are therefore not described here.

### 6.1.2 Evolving Quantum Circuits for Deutsch-Jozsa

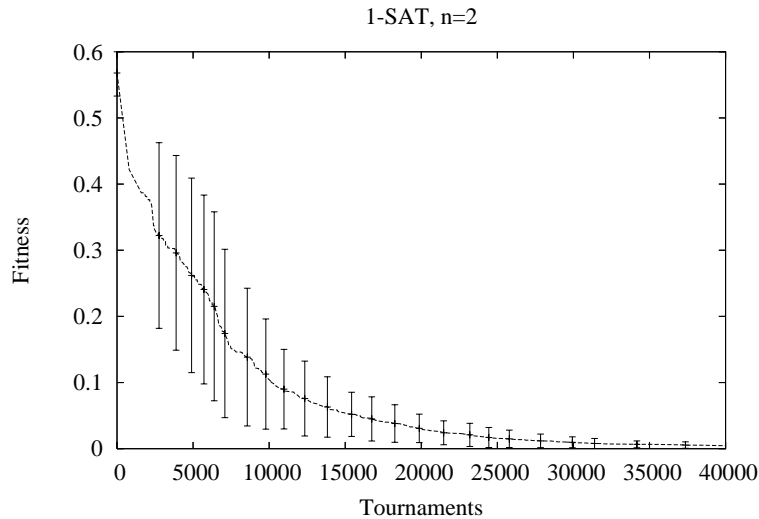
Also for the second problem (DJ), the use of intermediate measurements has no noticeable positive effect. Evolutionary runs are performed using a fitness calculation based on desired outputs in basis state representation (no classification assessment). For example,

$$(|000\rangle, f(x_1, x_2) = x_1 \text{ XOR } x_2, | - -1\rangle)$$

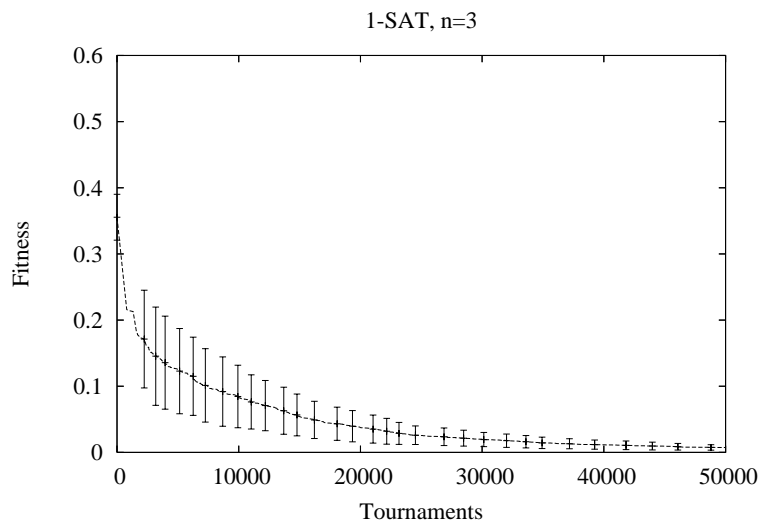
is the fitness case for the *XOR* function of two variables  $x_1, x_2$  ( $n = 2$ ). The desired output state for balanced functions is described as  $| - -1\rangle$ , that is, measuring the zeroth qubit as ‘1’ signals that the function is balanced and not constant. Any superposition of base states with second qubit ‘1’ is acceptable. The desired output state for the two constant functions is  $| - -0\rangle$ , that is, the final measurement of the zeroth qubit has to yield ‘0’ to make the function’s property identifiable. The choice of the decisive qubit is arbitrary, as is the measurement outcome for balanced and constant functions. Defining a certain qubit to carry the property tag reduces the number of possible quantum circuits solving DJ. The quantum algorithm presented in Section 2.8.1 classifies the Boolean functions (balanced or constant) in a different way. Thus, this algorithm is not evolvable due to the precondition given by the fitness cases (strictly speaking, by the definition of desired outputs). A side-effect of this definition is that scalable quantum algorithms are harder to find.

Figures 6.5 and 6.6 show the progress of evolution for the Deutsch-Jozsa problem instances with  $n = 1 \dots 3$  averaged over 100 runs for  $n = 1, 2$  and 30 runs for  $n = 3$  respectively. The most important parameter settings for these runs are given in Table 6.2. Many of them are comparatively chosen arbitrarily, such as the population size. Yet, it was not the focus of these experiments to optimize the parameter setting but to check whether the use of intermediate measurements has any advantage.

Not many of the evolved quantum circuits solving DJ did use intermediate measurements. For the problem instance  $n = 3$  there were no evolved solutions using a single intermediate measurement. In most circuits (for  $n = 1, 2$ ), using intermediate measurements had no effect at all, i. e., they could be removed without changing the final prob-



(a)



(b)

Figure 6.3: Two graphs illustrating the course of 100 evolutionary runs for quantum algorithms for the two- and three-variable 1-SAT problem. Errorbars show the standard deviation for the averaged fitness values of the 100 best evolved quantum algorithms after a certain number of tournaments. The dotted line marks averaged fitness values. Convergence of the evolution is obvious.

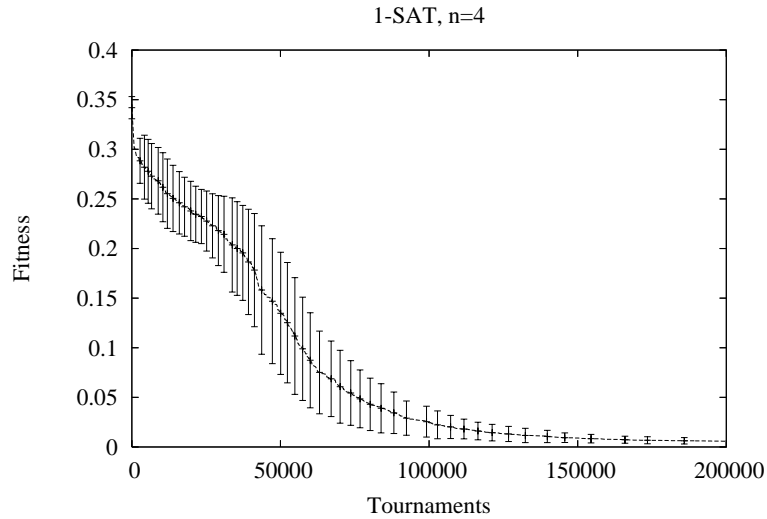
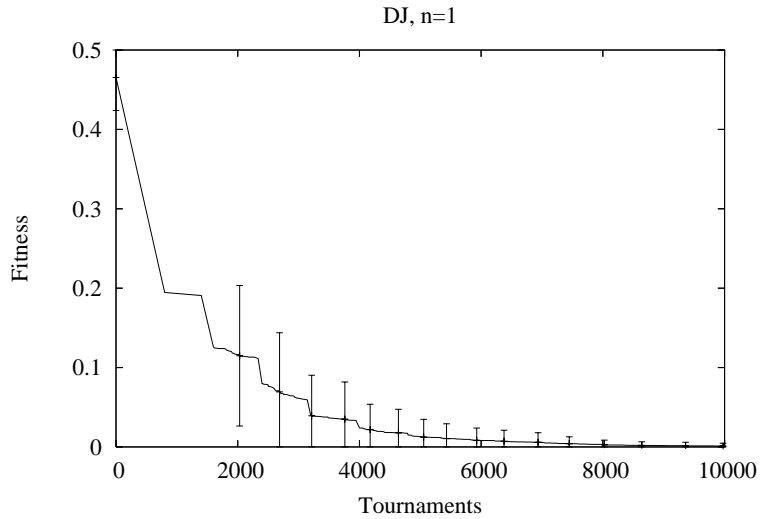


Figure 6.4: The course of 100 evolutionary runs for quantum algorithms for the four-variable 1-SAT problem. Errorbars show the standard deviation for the averaged fitness values of the 100 best evolved quantum algorithms after a certain number of tournaments. The dotted line marks averaged fitness values. Convergence of the evolution is obvious.

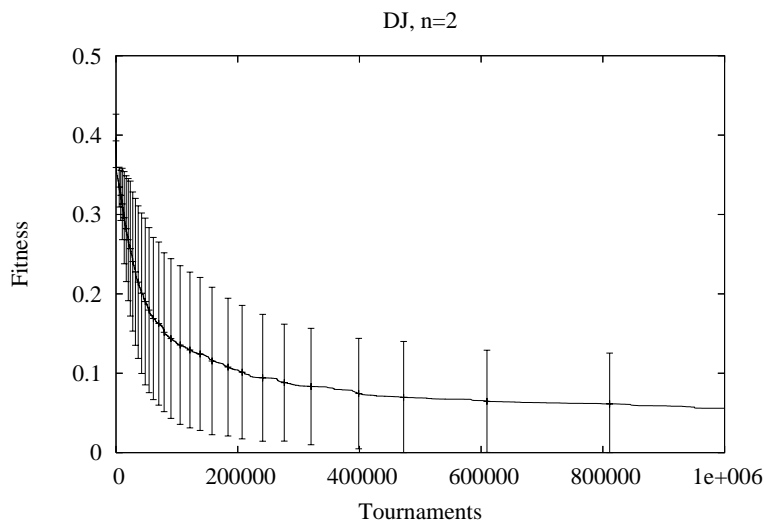
GP Structure	linear-tree
Selection	tournament, elitist
Tournament Size	2
Population Size	500
Basic Gate Types	$H, Rx, Ry, Rz, C^k NOT, INP, M$
Max. No. of Gates	10 (for $n = 1, 2$ ) / 15 (for $n = 3$ )
Max. No. of Measurements	$n + 1$ (no. of qubits)
Number of Input Gates	1
Mutation Rate	1
Crossover (XO) Rate	0.1
Linear XO Probability	0.9

Table 6.2: Parameter settings for evolutionary runs for the Deutsch-Jozsa problem with  $n = 1 \dots 3$ .





(a)



(b)

Figure 6.5: Two graphs illustrating the course of 100 evolutionary runs for quantum algorithms for the Deutsch-Jozsa problem with  $n = 1, 2$ . Errorbars show the standard deviation for the averaged fitness values of the 100 best evolved quantum algorithms after a certain number of tournaments. The line marks averaged fitness values.

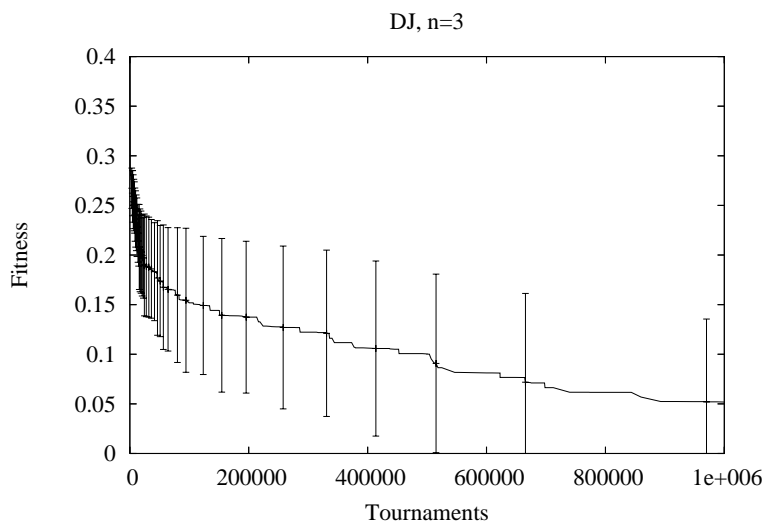


Figure 6.6: The course of 100 evolutionary runs for quantum algorithms for the Deutsch-Jozsa problem with  $n = 3$ . Errorbars show the standard deviation for the averaged fitness values of the 100 best evolved quantum algorithms after a certain number of tournaments. The line marks averaged fitness values.

ability measures. For example, for  $n = 2$  in about 80% of the evolved quantum circuits with intermediate measurements the measurements were obviously useless. Moreover, for  $n = 2, 3$  the evolution frequently failed to find a proper solution in the given maximum number of tournaments. For  $n = 2$  ( $n = 3$ ) about 40% (10%) of the evolutionary runs did not lead to a proper quantum circuit.

These results were decisive for stopping evolving quantum algorithms using intermediate measurements and programming a separate linear GP system. However, this does not mean that intermediate measurements are dispensable in general. Rather, both 1-SAT and DJ have scalable solutions of such a “simple” structure that any intermediate measurement can only interfere with this.

Scalability is not (or not easily) detectable, not least because of the design of the fitness cases. A mere visual comparison of solutions for the problem instances was unsuccessful. However, the structure of the original Deutsch-Jozsa algorithm “shines through” most of the evolved quantum algorithms.

### 6.1.3 Pre-Evolved Initial Populations

The benefit of using “pre-evolved” initial populations for the evolution of scalable quantum algorithms is now demonstrated for the linear GP system.

GP Structure		linear
Fitness Assessment Method		classification, with gate number penalty
Selection		tournament, elitist
Tournament Size		2
Population Size		1500
Basic Gate Types	$H, Rx, Ry, Rz, NOT, CNOT, INP$	
Max. No. of Gates		10 ( $n = 1$ ) / 15 ( $n = 2, 3$ )
Number of Input Gates		1
Genetic Operator		mutation
Early Termination		no
Gate No. penalty		50

Table 6.3: Parameter settings for the Deutsch-Jozsa problem with  $n = 1 \dots 3$ . Evolutionary runs with random initial population and those with pre-evolved initial population have the same parameter settings.

The GP system reads the individuals, evaluated solutions of a smaller problem instance, from a file and feeds with them the initial population. If there are still individuals missing in the population it is filled up with individuals from the file, which are mutated to a maximum of 25% of their length before adding them to the population. That is, for a quantum circuit of length 20, five mutations are performed at the most, adding random gates or replacing randomly chosen gates by random gates. This is of course arbitrary, and there might be other approaches which promote evolution even better. However, the aim of these experiments is just to show that the use of pre-evolved start populations can effect an accelerated evolution.

For the Deutsch-Jozsa problem with  $n = 1$  (2 qubits) 100 solutions are evolved. Each of them is mutated 14 times to generate the initial population of 1500 individuals for the Deutsch-Jozsa problem with  $n = 2$  (3 qubits). The best solutions of 100 runs are again mutated 14 times. Mutations and original individuals build the initial population of size 1500 for DJ with  $n = 3$  (4 qubits). For this problem only 20 evolutionary runs are performed. The fitness of each newly found best individual is stored along with the number of individual evaluations done, until termination criteria are met. The most important parameters are shown in Table 6.3.

The plots in Figure 6.7 and 6.8 illustrate the course of fitness averaged over all evolutionary runs with randomly created and with pre-evolved initial population. Standard deviations are plotted as errorbars.

The same approach is applied to 1-SAT. For  $n = 2$  (2 qubits) 100 solutions are evolved. Each of them is mutated four times to generate the initial population of 500 individuals

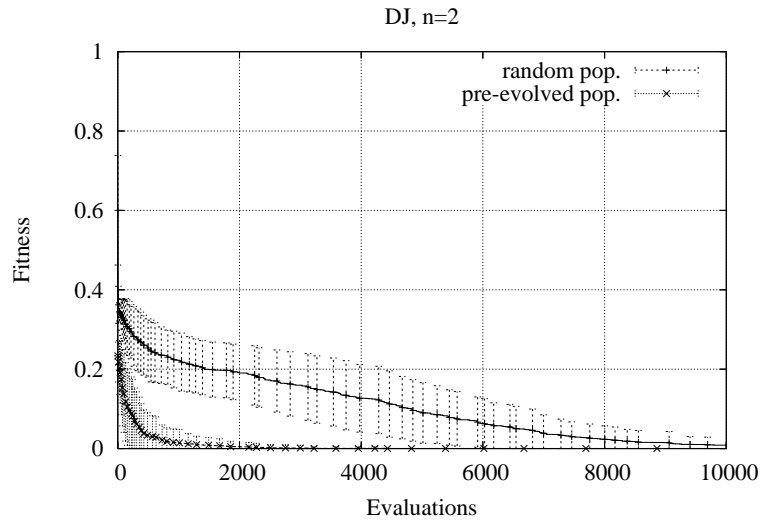


Figure 6.7: The average course of 100 evolutionary runs for the Deutsch-Jozsa problem with  $n = 2$  using a random and a pre-evolved initial population. The latter is based on the results of 100 evolutionary runs for the Deutsch-Jozsa problem with  $n = 1$ .

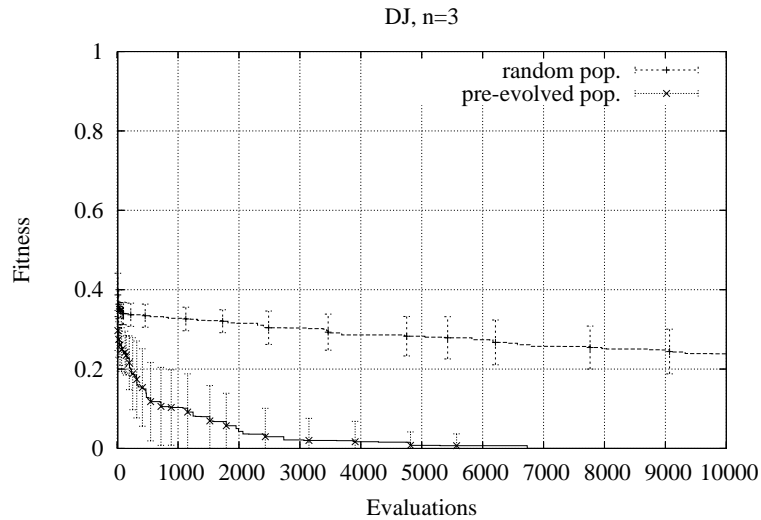


Figure 6.8: The two plots show the averaged fitness courses of the 20 best evolved quantum algorithms for the Deutsch-Jozsa problem with  $n = 3$  using random and pre-evolved initial population. The plots contain also the standard deviations. The pre-evolved initial population is based on the results of 100 evolutionary runs for the Deutsch-Jozsa problem with  $n = 2$ .

GP Structure		linear
Fitness Assessment Method		basis state, with gate number penalty
Selection		tournament, elitist
Tournament Size		2
Population Size		500
Basic Gate Types	$H, R_x, R_y, R_z, NOT, CNOT, INP$	
Max. No. of Gates		10 ( $n = 1$ ) / 15 ( $n = 2, 3$ )
Number of Input Gates		1
Genetic Operator		mutation
Early Termination		no
Gate No. penalty		50

Table 6.4: Parameter settings for the 1-SAT problem with  $n = 2 \dots 4$ . Evolutionary runs with random initial population and those with pre-evolved initial population have the same parameter settings.

for 1-SAT with  $n = 3$  (3 qubits). The best solutions of 100 runs are again mutated four times. Mutations and original individuals build the initial population of size 500 for 1-SAT with  $n = 4$  (4 qubits). For this problem again 100 evolutionary runs are performed. The most important parameters are shown in Table 6.4.

The plots in Figure 6.9 and 6.10 illustrate the course of fitness averaged over all evolutionary runs with randomly created and with pre-evolved initial population. Standard deviations are plotted as errorbars.

For both problems the positive effect on the evolution using pre-evolved initialized populations is obvious. This might not be a surprise: if the evolved algorithms for the smaller problem instance are scalable algorithms (in the sense that they are based on a general algorithm solving the problem of arbitrary size), then the individuals in the initial population should be close to the solutions. For example, the 1-SAT solutions of problem sizes  $n = 2$  and  $n = 3$  (or  $n = 3$  and  $n = 4$ ) presented in Figure 6.2 have only a “distance” of two mutation steps. It remains to find out whether this approach is still of value for more difficult problems which have a more complex structure and a larger “distance” between the (scalable) algorithms of two consecutive problem instances (of size  $n$  and  $n + 1$ ).

Another benefit can be derived from the use of pre-evolved individuals in the initial population: The evolution is not only faster, evolved scalable solutions are also easier to identify. Figure 6.11 gives an example for three evolved quantum circuits for DJ with  $n = 1, 2, 3$  which obviously hint at a general scalable algorithm.

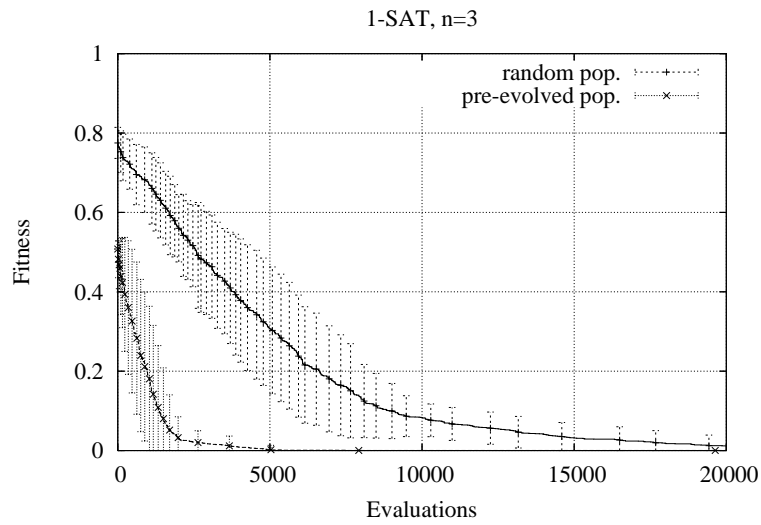


Figure 6.9: This graph illustrates the average course of 100 evolutionary runs for the three-variable 1-SAT problem using a randomly initialized population and a pre-evolved initial population. The pre-evolved initial population is based on the results of 100 GP runs for the two-variable 1-SAT problem. Errorbars show the standard deviation.

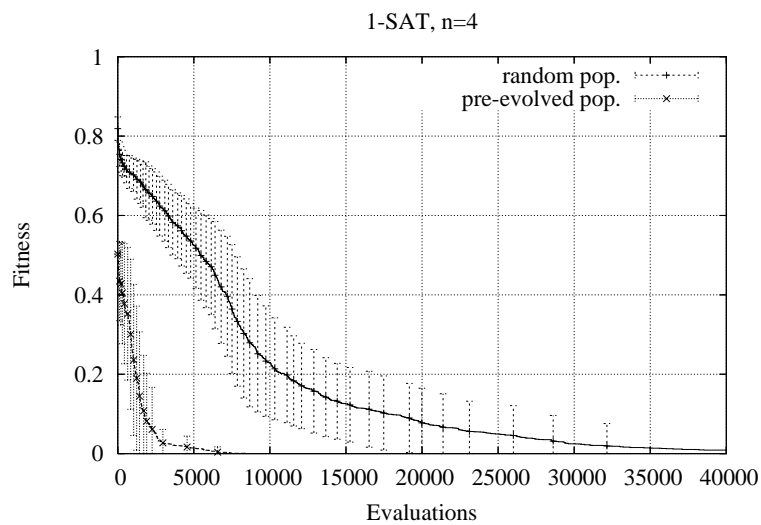


Figure 6.10: Average course of 100 evolutionary runs for the four-variable 1-SAT problem using a randomly initialized population and a pre-evolved initial population. The pre-evolved initial population is based on the results of 100 evolutionary runs for the three-variable 1-SAT problem. Errorbars show the standard deviation.

		Ry[5/4 Pi] 0
	Ry[5/4 Pi] 0	H 1
Ry[5/4 Pi] 0	H 1	H 2
H 1	H 2	H 3
INP	INP	INP
H 1	H 1	H 1
	H 2	H 2
		H 3

Figure 6.11: Hand-tuning was not necessary: these three quantum algorithms for Deutsch-Jozsa with  $n = 1 \dots 3$  are evolved according to the method described above (using pre-evolved initial populations for  $n = 2, 3$ ). The “scheme” of the scalable quantum algorithm for arbitrary  $n$  is obvious. The angle parameter is stated more precisely in fractions of  $\pi$ .

To sum it up, it can be said that the use of pre-evolved initial populations provides a cost-effective alternative to GP systems directly evolving scalable quantum algorithms. This approach cannot only be used to speed up evolution but also to improve detection of scalability.

#### 6.1.4 Disruptive Crossover

In many GP systems crossover is the predominant search operator. The idea behind this is that of good “building blocks”: subprograms improving the fitness which are worth being multiplied and spread over the population while evolution proceeds. When crossover is of benefit to evolution, then it has the function of preserving good schemata (*GP schema theorem* [8]). Unfortunately, on quantum circuit evolution crossover does not seem to have any positive effect as the following experiments for the Deutsch-Jozsa problem with  $n = 2$  and 1-SAT with  $n = 3$  demonstrate.

As a result of tournament selection, winners replace losers and are modified by genetic operators. In the experiments, a single-step mutation is always applied to the winners and with a certain rate (linear) crossover is also performed between the two winners of two parallel tournaments. The crossover operator is modified in a way that should favor its effect on evolution: applied to quantum algorithms with a single input gate, it either exchanges subcircuits before or after the single input gate. Thus, there is no swap of subcircuits including the input gate. Crossover rates and other system parameters are listed in Table 6.5.

Figure 6.12 shows the plots of evolutionary progress averaged over 100 runs for different crossover rates. It is obvious that using the crossover operator adversely affects the evolution of quantum circuits for both problems with different fitness functions with increasing crossover rate. The analysis of crossover landscapes in Section 6.2 confirms

	1-SAT / DJ
GP Structure	linear
Fitness Assessment Method	basis state / classification with gate number penalty
Selection	tournament, elitist
Tournament Size	2
Population Size	500 / 1500
Basic Gate Types	$H, Rx, Ry, Rz, NOT, CNOT, INP$
Max. No. of Gates	10 / 15
Number of Input Gates	1
Genetic Operator	mutation, crossover
Mutation Rate	1.0
Crossover Rate	0.0, 0.3, 0.5, 0.7, 1.0
Early Termination	no
Gate No. penalty	50

Table 6.5: Parameter settings for 1-SAT and Deutsch-Jozsa for evolutionary runs examining the effect of crossover.

that using the crossover operator (at least for these two problems) damages rather than benefits evolution.

## 6.2 Search Space Analysis

In evolutionary search methods, the population moves on the so-called *fitness landscape* [143, 157] by means of evolutionary operators. Roughly speaking, a fitness landscape in evolutionary computation is a search space – the space of all phenotypes, represented by their genotypes – extended by the corresponding genotype’s fitness values as the “altitude” and (optionally) a neighborhood structure. The fitness (dependent on the problem) decisively influences the contour and surface of the landscape. The neighborhood of genotypes is determined by an evolutionary (genetic) operator, typically an elementary mutation or a one-point crossover. Mathematically, in finite (discrete) spaces a fitness landscape can be considered to be a graph whose vertices are genotypes including an assigned fitness value and whose edges are defined by the evolutionary operator, obeying the so-called “one-operator, one-landscape”-concept [157].

Here, the considered mutation operator consists of random *deletion*, *insertion* and *replacement* of a single quantum gate. The mutation is randomly selected (equally distributed) from the set of all possible mutations. The linear crossover operator used has already been described in Section 6.1.4. The experiments analyzing the fitness landscapes for 1-SAT and Deutsch-Jozsa problem instances use the linear GP system.



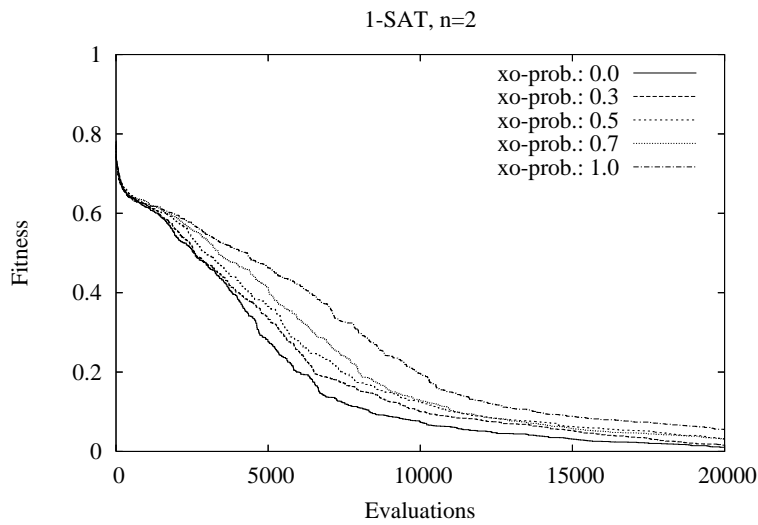
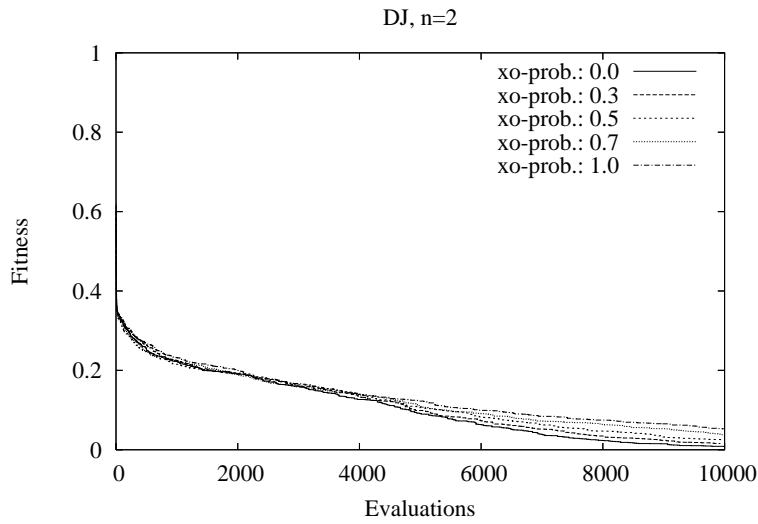


Figure 6.12: These graphs illustrate the unfavorable impact of crossover on the evolution of quantum circuits for (a) the Deutsch-Jozsa problem with  $n = 2$  and (b) 1-SAT with  $n = 3$ .

The fitness of a quantum program for DJ is assessed according to its quality of classification as described in Section 5.4.3. In the case of 1-SAT, the fitness is calculated as stated previously in Section 5.4.3 and 6.1.1. There is no size penalty for large quantum circuits. Experiments are conducted using the universal gate set  $\{H, NOT, CNOT, CCNOT, Rx, (\phi), Ry, (\phi), Rz, (\phi), INP\}$ .

### 6.2.1 Fitness Landscapes and Analysis Methods

A landscape can be characterized by its ruggedness and neutrality. Ruggedness refers to the average correlation between neighboring genotypes, neutrality refers to the size of flat landscape areas [125]. Since the study of landscape structures helps to understand the ability of evolutionary algorithms to perform efficiently, techniques for revealing these characteristics are important. Rugged and neutral fitness landscapes are usually considered to be difficult to evolve.

The structure of fitness landscapes regarding DJ and 1-SAT is analyzed using two approaches: the autocorrelation analysis by Weinberger [163] and the information analysis by Vassilev et al. [155, 156, 157]. Both techniques are based on time series  $\{f_t\}_{t=0}^n$  obtained by random walks on the mutation and crossover landscape.  $f_t$  is the fitness value of the genotype reached after  $t$  steps from the starting point.

The autocorrelation function of a time series is defined by

$$\rho(s) = \frac{E[f_t f_{t+s}] - E[f_t]E[f_{t+s}]}{V[f_t]}$$

where  $E[f_t]$  is the expectation value and  $V[f_t]$  the variance. It is a measure for the correlation between points on the landscape that are separated by  $s$  steps, i. e.  $s$  applications of the mutation operator. The correlation length  $\tau = -\frac{1}{\ln \rho(1)}$  is an important measure for the ruggedness of a landscape.

Information analysis is used to investigate the structure in more detail, especially features related to flat areas (fitness plains), isolated points (locally isolated maxima or minima in the search space), and landscape structures having neither characteristic. The information measures being considered are

- *information content*  $H(\epsilon)$ , to estimate the diversity of landscape shapes,
- *partial information content*  $M(\epsilon)$ , to measure the modality of the landscape path,

The information content is obtained by transforming the time series  $\{f_t\}_{t=0}^n$  into a string  $S(\epsilon) = s_1 s_2 \dots s_n$  of symbols  $s_i \in \{\bar{1}, 0, 1\}$ , where

$$s_i = \begin{cases} \bar{1}, & \text{if } f_{i-1} - f_i < -\epsilon \\ 1, & \text{if } f_{i-1} - f_i > \epsilon \\ 0, & \text{otherwise} \end{cases}$$

for  $\epsilon \in [0, l]$  where  $l$  is the maximum difference between two fitness values (the highest elevation in the fitness landscape). Then, the information content of the time series is  $H(\epsilon) = -\sum_{p \neq q} P_{[pq]} \log_6 P_{[pq]}$ , where  $P_{[pq]} = n_{[pq]}/n$  and  $n_{[pq]}$  is the number of occurrences of  $pq$ ,  $p, q \in \{\bar{1}, 0, 1\}$ , in  $S(\epsilon)$ . Thus,  $P_{[pq]}$  is the share of  $pq$ -blocks in  $S(\epsilon)$ . Increasing  $\epsilon$  is like zooming out from the landscape, from a closer inspection to a rough overview.

The partial information content is determined from  $S(\epsilon)$  by counting the number of slopes  $\mu(\epsilon)$  in the landscape path which is equivalent to the number of changes in the fitness trend (plus 1). To evaluate  $\mu(\epsilon)$  consider the string  $S(\epsilon)$ . Remove all symbols 0 and replace all sequences (runs) of equal symbols by a single symbol. The resulting string  $S'(\epsilon)$  is of the form  $1\bar{1}1\bar{1}1\bar{1} \dots$  or  $\bar{1}\bar{1}\bar{1}\bar{1}\bar{1} \dots$  and  $\mu(\epsilon)$  is just the length of  $S'(\epsilon)$ . Thus,  $\mu(\epsilon)$  indicates the modality of the landscape path. With it, the partial information content is given by  $M(\epsilon) = \mu(\epsilon)/n$ . For example, for  $S(\epsilon) = 01\bar{1}1\bar{1}11100\bar{1}\bar{1}01\bar{1}0$  it is  $\mu(\epsilon) = 8$  and  $M = 0.5$  since  $S'(\epsilon) = 1\bar{1}1\bar{1}1\bar{1}\bar{1}$ . The number of optima in the landscape path is  $\lfloor \frac{1}{2}\mu(\epsilon) \rfloor$ .

Another information measure is the *density-basin information*  $h(\epsilon)$ , used to estimate the variety of flat and smooth areas. It is also based on the string  $S(\epsilon)$ :

$$h(\epsilon) = - \sum_{p \in \{\bar{1}, 0, 1\}} P_{[pp]} \log_3 P_{[pp]}$$

. However, this information measure is not used here, as it does not really contribute new insights.

For further information on the information landscape analysis, refer to [157].

### 6.2.2 Landscape Analysis

For this analysis, random walks on problem-specific mutation and crossover landscapes are performed for quantum circuits from 2 up to 4 qubits ( $n = 1 \dots 3$  for DJ and  $n = 2 \dots 4$  for 1-SAT) and maximum circuit sizes  $L = 10, 15, 20, 25, 30$ .

Random walks on a *mutation* landscape start with a random genotype and continue successively by random (one-point) mutation from one genotype to a neighbor. Random walks on a *crossover* landscape [160] start with two random genotypes to which the crossover operator is applied generating two offspring. From those, one is chosen randomly to mate with another random genotype. For each landscape defined by  $L$  and  $n$ , 100 walks are performed, each consisting of 100,000 steps. Both methods, autocorrelation analysis and information analysis are applied to the same time series.

#### Autocorrelation Analysis

The averaged autocorrelation functions of random walks on the mutation landscapes for different  $L$  and  $n$  for the Deutsch-Jozsa problem and the corresponding standard

deviations are depicted in Figure 6.13. Figure 6.14 represents the averaged autocorrelation functions and standard deviations for random walks on crossover landscapes. For 1-SAT, Figure 6.15 and 6.16 show the autocorrelation functions and standard deviations of mutation and crossover landscapes. The correlation length can also be defined as the displacement for which  $\rho(s)$  is equal to  $1/e$  ( $\approx 0.3678$ ). In the autocorrelation plots the horizontal line marks this value making the correlation length directly readable.

Problem independent, the plots show that correlations of (one-point) mutation landscapes are extremely low compared, for instances, to the autocorrelation of mutation landscapes for digital circuit evolution [156]. However, for crossover landscapes the autocorrelations are even lower. The correlation lengths are below two, that is, the points on the crossover landscapes are almost entirely uncorrelated. This confirms the experimental results presented in Section 6.1.4 which were decisive for refraining from using a crossover operator in subsequent evolutionary runs.

For the autocorrelation functions based on random walks on mutation landscapes for 1-SAT: With increasing  $n$  (or number of qubits respectively) the autocorrelations increase, but only slightly, while on the contrary with increasing circuit length  $L$  the correlations of the landscapes decrease. This is similar to the autocorrelation functions for DJ, although the plots are here clearly closer together. Any effect of a larger circuit length  $L$  on the autocorrelation is hard to detect here. The correlation lengths show that beyond three steps, the points on the mutation landscape path become almost uncorrelated.

From these observations it can be concluded that the fitness landscapes are rather rugged and seem to become only slightly smoother for larger  $n$ . Rugged landscapes make evolutionary search very difficult, though not impossible. The existence of crossover operators with better landscape characteristics cannot be ruled out.

### Information Analysis

The following has focused exclusively on mutation landscapes as the crossover operator is apparently of no importance for efficiency improvements of the evolutionary search process.

The information characteristics  $H(\epsilon)$  and  $M(\epsilon)$  for DJ are depicted in Figure 6.19 ( $n = 2$ ) and 6.20 ( $n = 3$ ). Those for 1-SAT are represented in Figure 6.17 ( $n = 3$ ) and 6.18 ( $n = 4$ ). Since the plots for DJ with  $n = 1$  and 1-SAT with  $n = 2$  do not contribute to further conclusions they are skipped. For calculating the information characteristics the value of  $\epsilon$  is varied with a stepsize of 0.001 between 0 and 1. In the following, important characteristics of the plots are explained and analyzed.

All plots have in common that they become increasingly smoother with increasing maximum circuit size  $L$ . Since the plots are step functions, one can conclude an increasing number of altitude levels, that is, an increasing number of different fitness values, and consequently an increasing number of differences of fitness values of neighboring genotypes. This number, also referred to as the *degree of (ir)regularity* [157], can be interpreted as a measure of the complexity of the random walks. Accordingly, the figures reveal an increased complexity for increasing  $L$ .

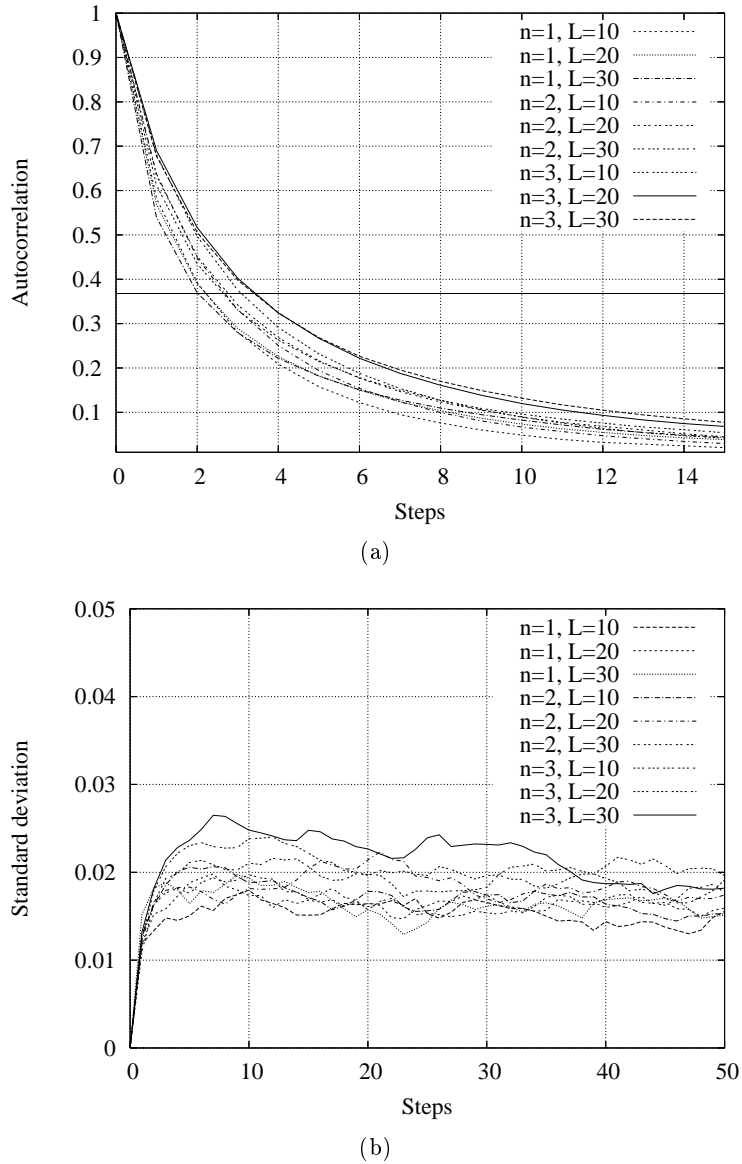
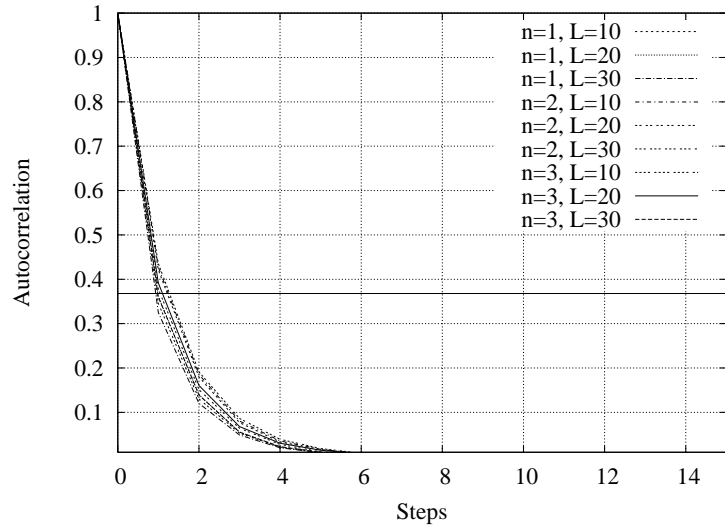
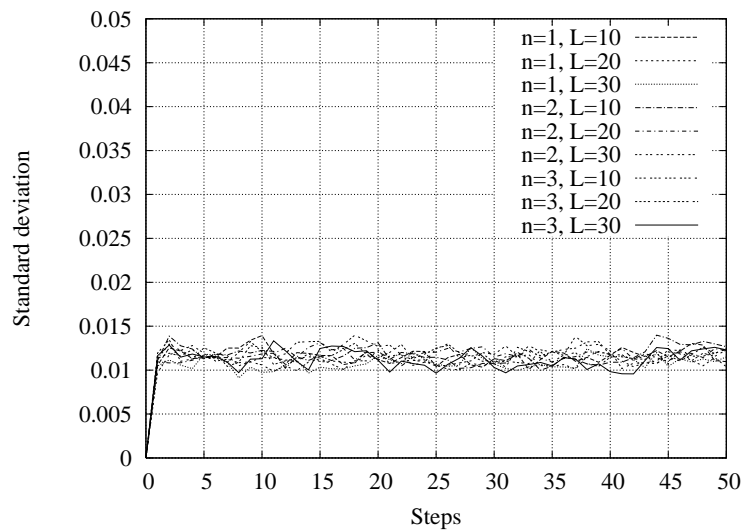


Figure 6.13: (a) Autocorrelation function of mutation landscapes for Deutsch-Jozsa problems with  $n = 1 \dots 3$  and  $L = 10, 20, 30$  averaged over 100 walks. The correlation length is between two and four. (b) The standard deviation is always  $< 0.03$ .



(a)



(b)

Figure 6.14: (a) Autocorrelation function  $\rho(s)$  of crossover landscapes for Deutsch-Jozsa problems with  $n = 1 \dots 3$  and  $L = 10, 20, 30$ . The correlation length is close to one. (b) The standard deviation is  $< 0.015$  considering  $\rho(s)$  averaged over 100 walks.

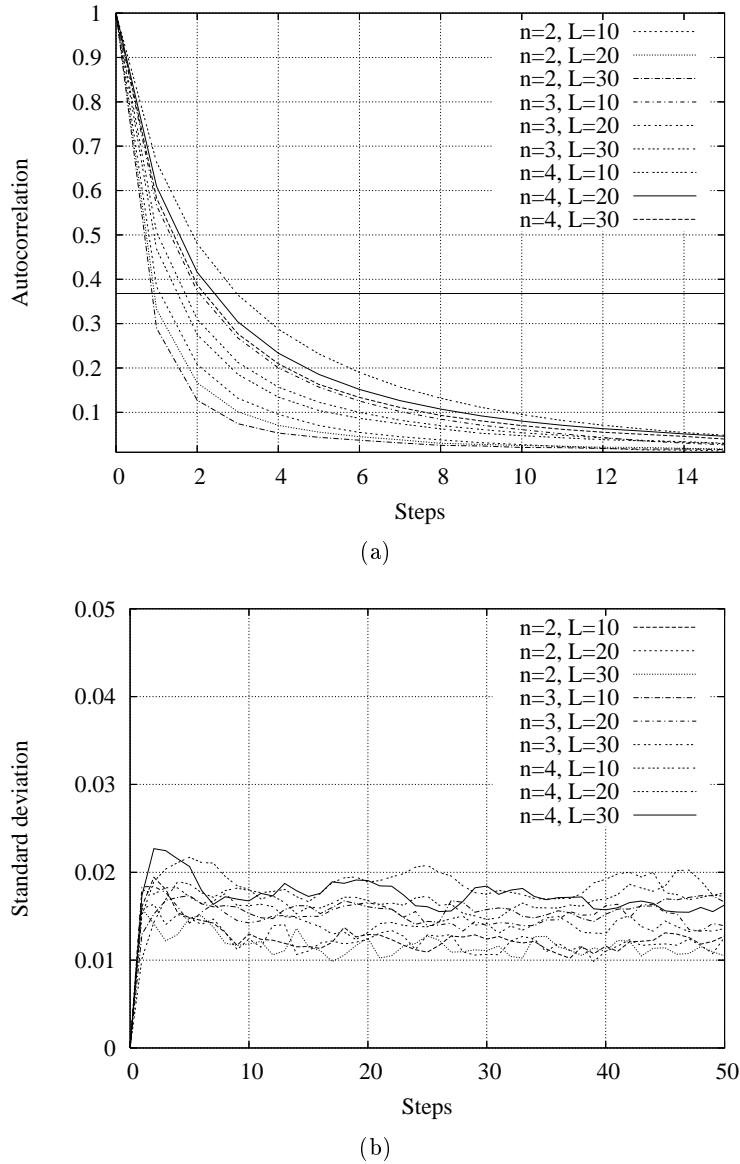
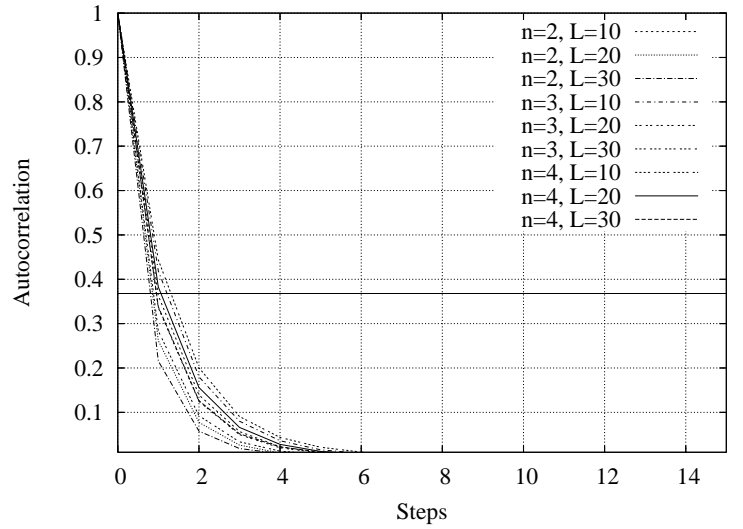
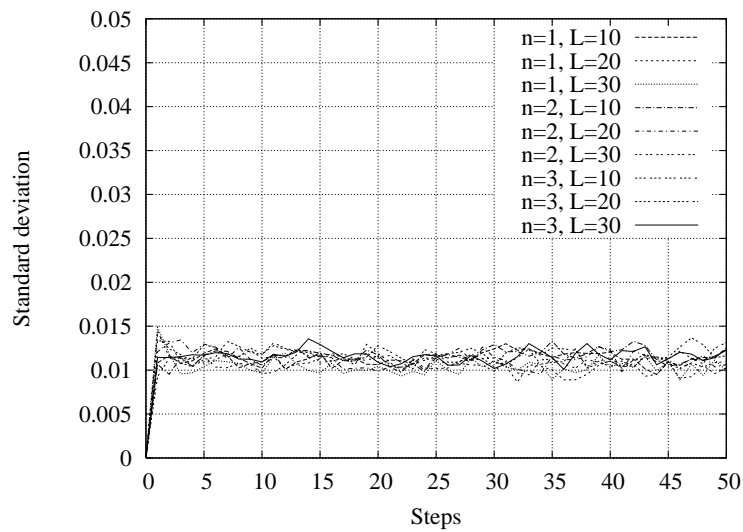


Figure 6.15: (a) Autocorrelation function of mutation landscapes for 1-SAT with  $n = 2 \dots 4$  and  $L = 10, 20, 30$  averaged over 100 walks. The correlation length is between two and four. (b) The standard deviation is always  $< 0.03$ .



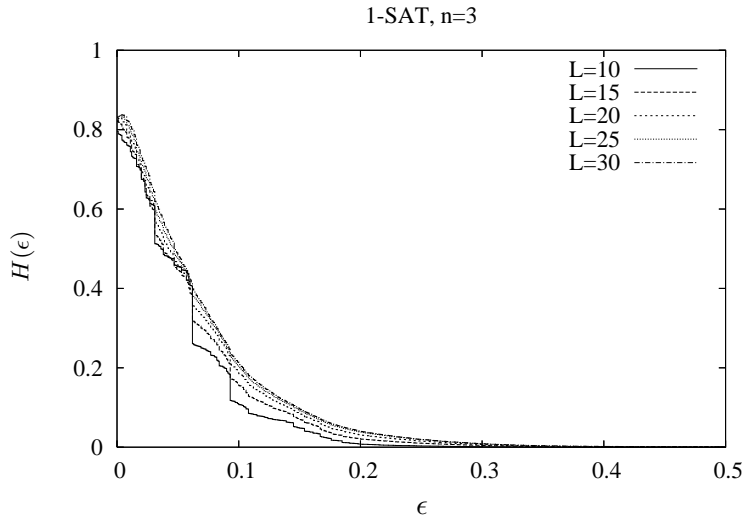
(a)



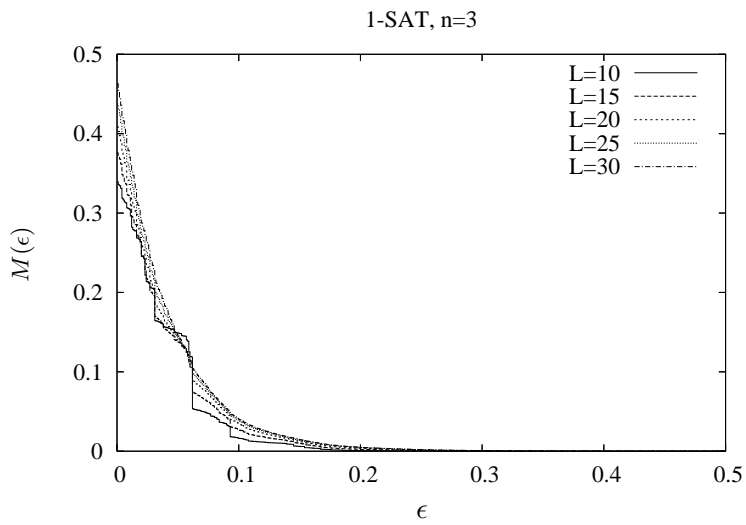
(b)

Figure 6.16: (a) Autocorrelation function of crossover landscapes for 1-SAT with  $n = 2 \dots 4$  and  $L = 10, 20, 30$ . The correlation length is close to one. (b) The standard deviation is  $< 0.015$  considering  $\rho(s)$  averaged over 100 walks.





(a)



(b)

Figure 6.17: Information characteristics for 1-SAT with  $n = 3$ . (a) Averaged information content  $H(\epsilon)$ . (b) Averaged partial information content  $M(\epsilon)$ .

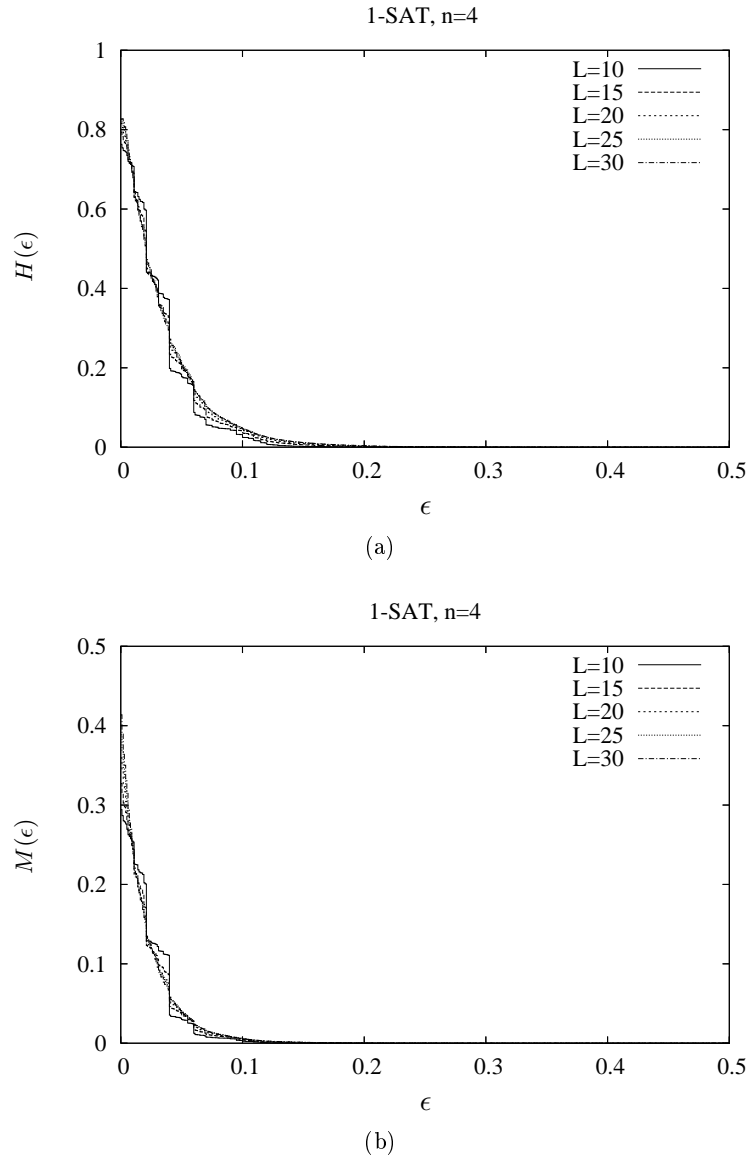
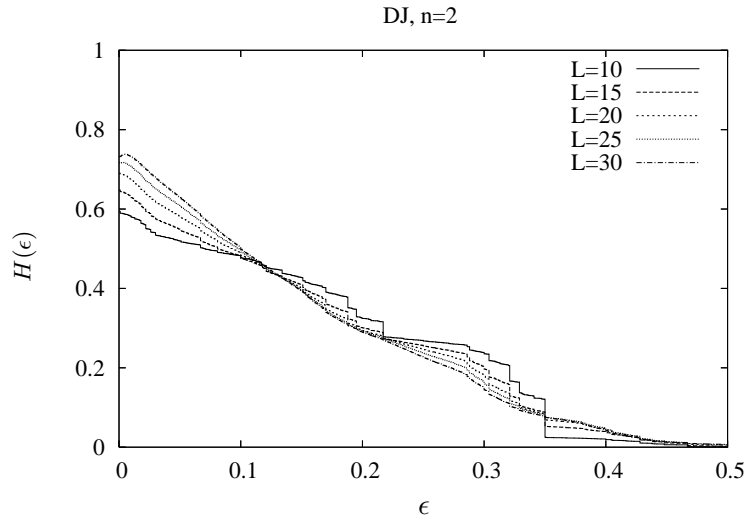
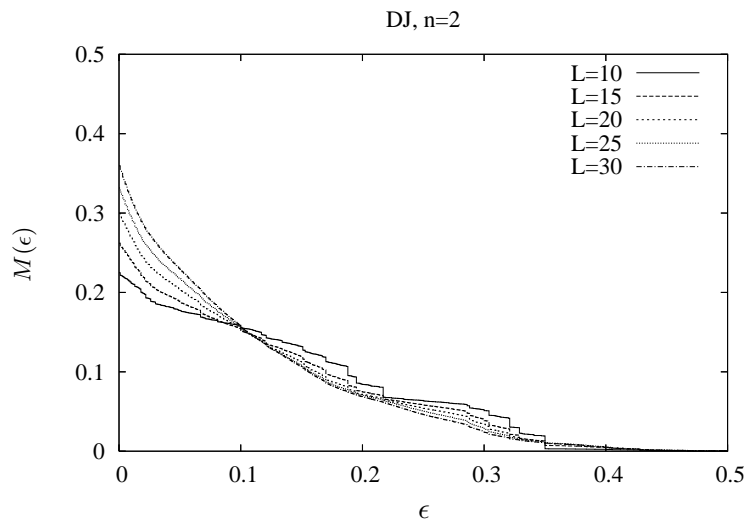


Figure 6.18: Information characteristics for 1-SAT with  $n = 4$ . (a) Averaged information content  $H(\epsilon)$ . (b) Averaged partial information content  $M(\epsilon)$ .



(a)



(b)

Figure 6.19: Information characteristics for Deutsch-Jozsa with  $n = 2$ . (a) Averaged information content  $H(\epsilon)$ . (b) Averaged partial information content  $M(\epsilon)$ .

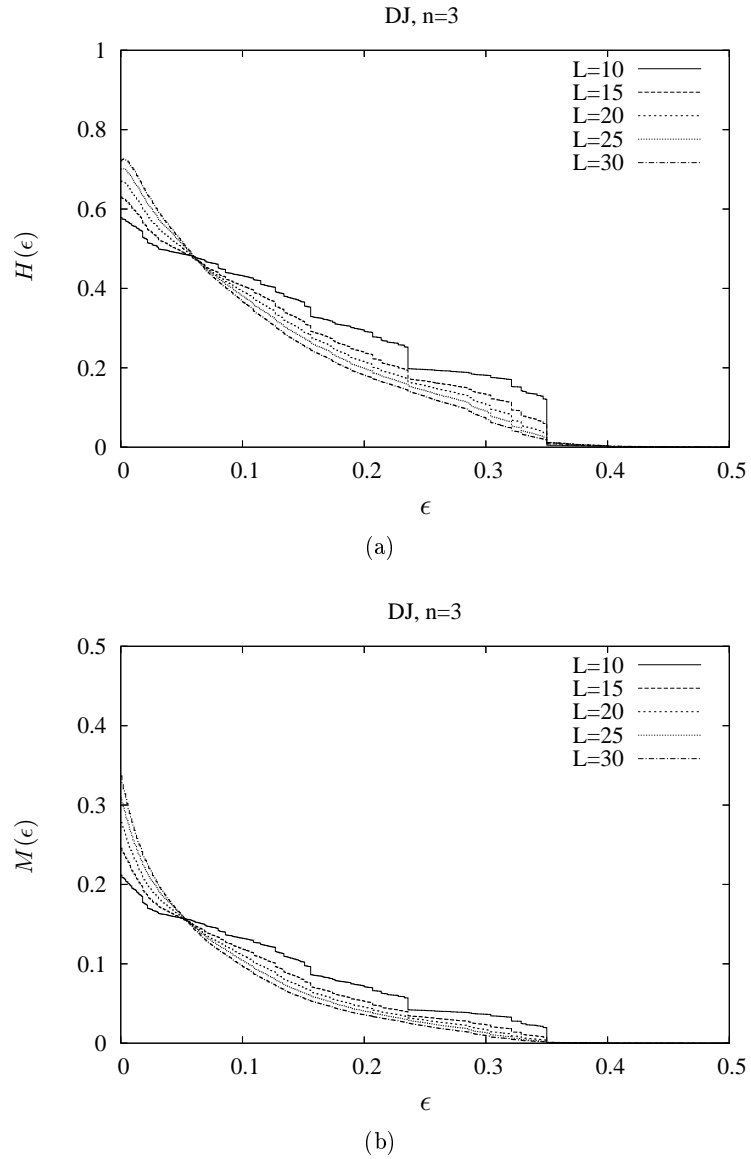


Figure 6.20: Information characteristics for Deutsch-Jozsa with  $n = 3$ . (a) Averaged information content  $H(\epsilon)$ . (b) Averaged partial information content  $M(\epsilon)$ .

Along with this, it can be seen for both problems and values of  $n$  that the averaged information content  $H(0)$  and the averaged partial information content  $M(0)$  increase as  $L$  increases. This indicates a higher diversity of the landscape for larger  $L$  which together with the increased modality (number of local optima) implies more rugged landscapes. However, the increase of  $H(0)$  is much more visible for the Deutsch-Jozsa problem than for 1-SAT, where  $H(0)$  only increases over a small interval. This suggests, that for DJ the landscape type changes more distinctly with the maximum length of quantum circuits than for 1-SAT. Moreover, the value of  $H(0)$  is in all four cases (for both problems and two different values of  $n$ ) significantly larger than  $\log_6 2$  which can be seen as an indication for some flat landscape areas [157]. For larger  $L$  the averaged information content  $H(\epsilon)$  tends to increase for increasing  $\epsilon$  from 0 up to small values, e. g. 0.006 for DJ with  $n = 3$  and  $L = 30$  (Figure 6.20a). This is another hint for differences in the diversity of landscape shapes for different  $L$ .

For both problems it can be observed that the information functions are steeper for the higher value of  $n$ . Since steeper information functions characterize smoother landscapes while less steep functions correspond to more rugged landscapes, the fitness landscapes become smoother for higher values of  $n$ . This corresponds to the results in the autocorrelation analysis. It is difficult to establish this for increasing  $L$ . However, for the Deutsch-Jozsa problem with  $n = 2$  and  $n = 3$ , the plots seem to become steeper as well. A higher steepness of  $M(\epsilon)$  indicates in other words a minor diversity of optima in terms of their magnitude.

Comparing the graphs for 1-SAT with those for DJ it is obvious that the former are far steeper than the latter and thus the mutation landscapes for 1-SAT are smoother than those for DJ. This might be the reason why evolutionary search with ES selection strategies (especially plus strategies) performs better on the 1-SAT landscapes, whereas on DJ landscapes supplied additional randomness for ES selection or tournament selection is more successful, as shown in the following section. The number of local optima given by the partial information content  $M(\epsilon)$  is a little larger for 1-SAT than for DJ but, as already stated above, in the case of 1-SAT it decreases much faster as  $\epsilon$  increases.

To sum up, it can be said that the information analysis hints at rugged landscapes for both problems. In comparison to DJ landscapes, 1-SAT landscapes are a little bit smoother. The modality is high but the magnitude of peaks is mostly very low which might benefit certain evolutionary approaches. In contrast, in DJ landscapes the diversity of optima is very high and multi-peak configurations seem to prevail over flat plains or plateaus. Evolution runs a higher risk here of getting stuck in a local optimum. For a larger number of qubits (a larger  $n$ ) the landscapes tend to become smoother making evolutionary search at least in principle easier. Information analysis is a useful tool as it provides additional information about fitness landscapes which autocorrelation analysis cannot and can help to adjust the search method. However, the interpretation of its plots is not a simple task and requires more experience in dealing with fitness landscapes.

## 6.3 Comparison of Selection Strategies

At present one must be content to *evolve* essentially already existing quantum algorithms and to analyze the search space of these quantum circuits in order to improve the efficiency of evolutionary search. If simulation of quantum circuits cannot be sped up, then the only way to novel, complex quantum circuits is via accelerated evolution.

This is the motivation for examining and comparing different selection strategies with respect to their effectiveness. Using the linear GP system, in addition to the tournament selection as part of the steady state GP the  $(\mu, \lambda)$  and  $(\mu + \lambda)$  ES selection as part of a generational GP approach are implemented. Moreover, the (1,10) ES selection strategy is combined with additional randomness and tested independently with step size adaptation. The experiments are exemplarily performed on the Deutsch-Jozsa problem with  $n = 2$  (three qubits) and  $n = 3$  (four qubits) and the 1-SAT problem with  $n = 3$  (three qubits) and  $n = 4$  (four qubits).

### 6.3.1 Selection Strategies

In general, selection is the result of a competition between individuals in a population. The better the individual's fitness in comparison with all other individuals in the population, the higher its selection probability. Some popular generational selection algorithms are the  $(\mu + \lambda)$  and  $(\mu, \lambda)$  ES selection as well as tournament selection. A coarse explanation of the comma and plus selection strategy is already given in Section 3.4. An algorithmic description of both selections  $(\mu + \lambda)$  and  $(\mu, \lambda)$  follows:

---

1. Generate the initial population; population size is:
    - $\mu + \lambda$  (for  $(\mu + \lambda)$  selection)
    - $\lambda$  (for  $(\mu, \lambda)$  selection)
  2. Evaluate the fitness of each individual;
  3. Select the winners:
    - the best  $\mu$  individuals in the entire population (for  $(\mu + \lambda)$  selection)
    - the best  $\mu$  newly created individuals or offspring respectively (for  $(\mu, \lambda)$  selection)
  4. Perform one-point mutation on each winner to generate  $\lambda$  offspring (about  $\lambda/\mu$  offspring per winner);
  5. Evaluate the fitness of the offspring;
  6. Go to step 3 unless termination criteria are met.
-

For further experiments, the (1,10) selection is extended by adding randomness. After fitness evaluation of the offspring, instead of the best offspring, individuals are chosen randomly with a given probability  $p$ . In this case, step two of the algorithm above is as follows:

2. Let  $u = \text{unif}([0, 1])$  uniformly distributed; if  $u < p$ , then choose the winner randomly from the set of offspring and go ahead to step 4, otherwise evaluate the fitness of each individual;

Furthermore, the (1,10) selection is used with *self-adaptation* [131] of step-sizes. The self-adaptation routine is due to [128] and is as follows:

```

u = unif([0, 1])
if u < 0.5
then m* = 1.3
else m/ = 1.3
nmut = geo(u, m)

```

The parameter  $n_{mut}$  determines the step size, i. e. the number of mutations. It is drawn from a geometric distribution. In more detail,

$$\text{geo}(u, m) = \left\lceil \frac{\ln(1-u)}{\ln(1-p)} \right\rceil, \quad \text{where } p = 1 - \frac{m}{1 + \sqrt{1+m^2}}.$$

The variable  $m$  denotes the mutation strength. Its value determines the average distance of an offspring individual to its parent individual after a mutation. By mutating the mutation strength before the mutation is carried out it is possible to control the mutation strength by a process which is termed self adaptation. Due to Schwefel [131] and others, the proposed mutation rule for the step size enables this self-adaptation for a broad range of problems provided that a surplus of offspring is generated in each generation, i. e.,  $\mu \ll \lambda$ . The initial value of  $m$  has to be chosen appropriately.

### 6.3.2 Experiments and Empirical Results

For each problem instance 20 evolutionary runs were performed for each selection method. The results were averaged over the runs. A GP run terminated when the number of single individual (quantum program) evaluations exceeded  $10^7$  or the fitness of a new best individual under-ran a given threshold, which is close to zero. The length of the quantum circuits was limited to 15 gates. The initial population was randomly created.

The selection strategies and parameter settings used in the linear GP system are:

- Comma- and Plus-strategies: (1,10), (5,20), (1+10), (5+20);
- (1,10) with step-size adaptation. The value of  $n_{mut}$  (the step size) was bounded to 5 mutations. The start value of  $m$  is set to 3. This might seem arbitrary, but experiments with larger values showed no or virtually no effect. A more detailed analysis is for future work.
- (1,10), combined with additional random selection.
- Tournament selection with different population sizes (100, 500, 1000, 1500, 2000). Tournament size is 2.

Figures 6.21a and 6.24a show four graphs each, illustrating the course of the evolutionary runs for the Deutsch-Jozsa problem for  $n = 2$  and  $n = 3$  using pure comma- and plus-selection strategies. The (1,10) strategy achieved the best results. In the case of  $n = 3$ , for this strategy 25% of the runs did not lead to an acceptable result and for other strategies the failure rate was even larger.

Tournament selection can outperform these runs when applied with a suitable population size of about 500 to 1000 individuals for  $n = 2$  and 1500 to 2000 individuals for  $n = 3$ , as shown in Figure 6.21b ( $n = 2$ ) and Figure 6.24b ( $n = 3$ ). For larger populations the performance decreases more and more. Yet, even for tournament selection not every run was successful within the required number of evaluations.

Performance of the (1,10) selection can be visibly boosted using step size adaptation which was, all in all, the best selection strategy. In Figure 6.23 and 6.26 the plot essentially runs below the plots relating to all other selection strategies. Curiously enough, the figures also reveal a comparable good performance achieved by using the (1,10) strategy combined with 10% random selection. Further experiments demonstrated that even higher percentages of random selection predominate over the pure comma- and plus-strategies as exemplarily shown in Figure 6.23 and 6.26 for (1,10) ES with additional 20% randomness. In all runs the resulting quantum circuit was a solution of the test problem. Thus, considering the number of successful runs, this strategy behaves even better than tournament selection on the given problem.

The corresponding standard deviations are depicted in the Figures 6.22, 6.23b, 6.25, and 6.26b. The number of evolutionary runs might be too small, especially, since the standard deviations are comparatively large. Further runs are necessary to verify the conclusions.

This result confirms the analysis of the mutation landscape of the Deutsch-Jozsa problem. Because of the high ruggedness, the high modality and only a few paths to global optima the evolution is often caught in local optima. Larger steps or - what seems to help as much - additional randomness appears to compensate for this.

Applied to the 1-SAT problem instance a completely different result is obtained. Here, the best strategies are tournament selection with a population size of about 500 individuals, as shown in Figure 6.29a, and (1+10) selection, as shown in Figure 6.29b.

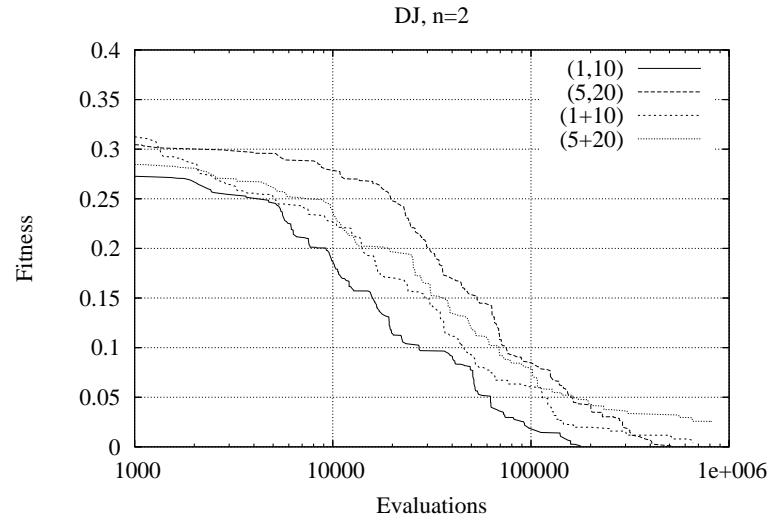


Furthermore, Figure 6.29b illustrates, that plus-strategies perform far better than the comma-strategies. Step size adaptation does not improve evolution and additional randomness rather deteriorates the evolutionary search (Figure 6.29a). Moreover, for each of the tested selection strategies all 20 evolutionary runs were successful on this problem. Standard deviations are illustrated in the Figures 6.28 and 6.30. The values are between 0 and 0.25. This shows that the fluctuation in evolution is very high. Here, further evolutionary runs have to be done to confirm the results.

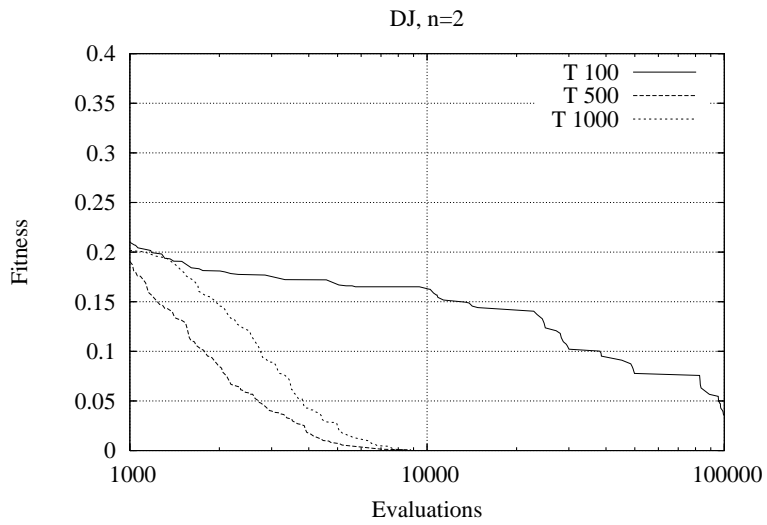
The experiments reveal that depending on the population size, tournament selection can be very effective on either test problem. Self-adaptation is similarly effective and enables comma-selection to compete against tournament selection.

The comma-strategy with random selection seems to be useful on complex problems with rugged fitness landscapes that are difficult for standard evolutionary selection mechanisms. Under those circumstances it can be effective or for smaller populations even better than tournament selection.

Further experiments on the influence of adding randomness for very difficult fitness landscapes would help to judge, whether this is more than just a pleasant side effect. Unfortunately there is only a small set of test problems available and evolution of quantum circuits on larger state spaces is prohibitively time consuming.

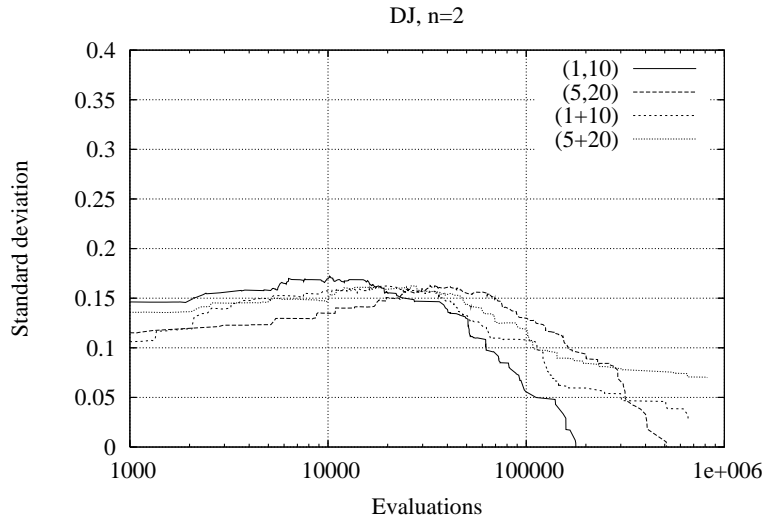


(a)

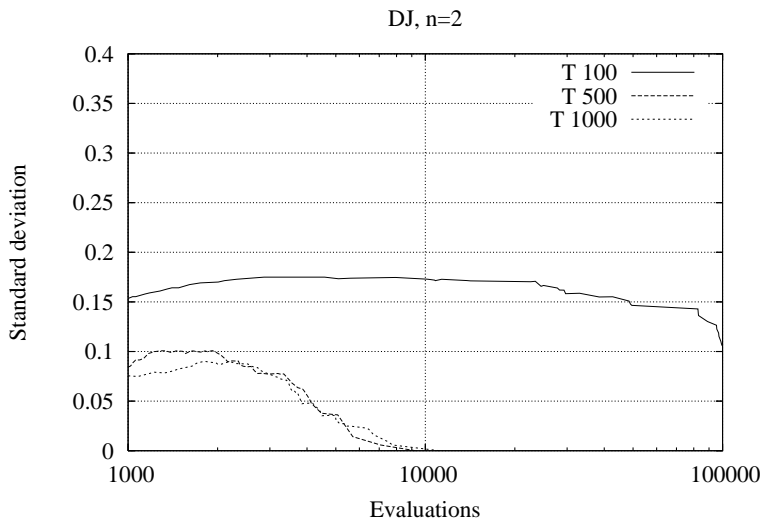


(b)

Figure 6.21: Impact of different selection strategies on the evolution of quantum circuits for the Deutsch-Jozsa problem with  $n=2$ : (a) comma- and plus-strategies; (b) tournament selection for different population sizes; Note the logarithmic scaling on the x-axis.



(a)



(b)

Figure 6.22: Standard deviation for the plots in Figure 6.21.

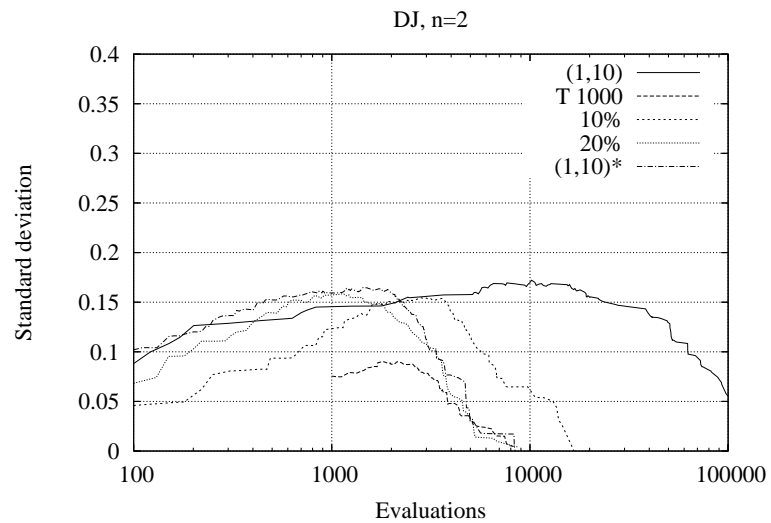
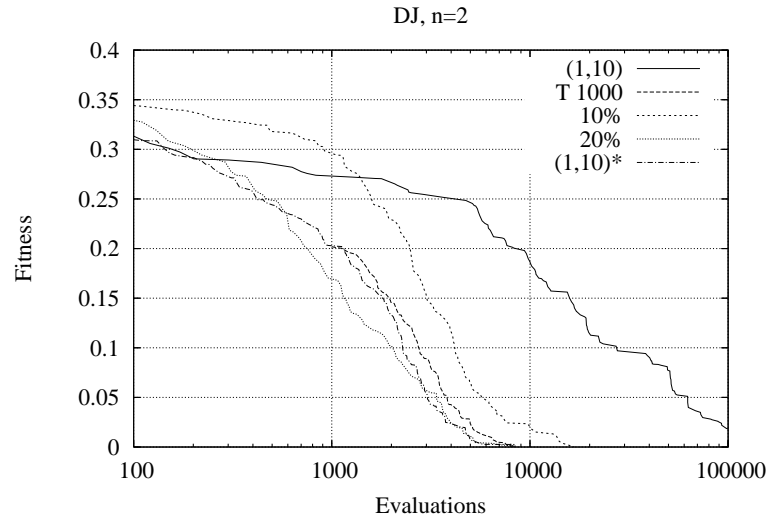
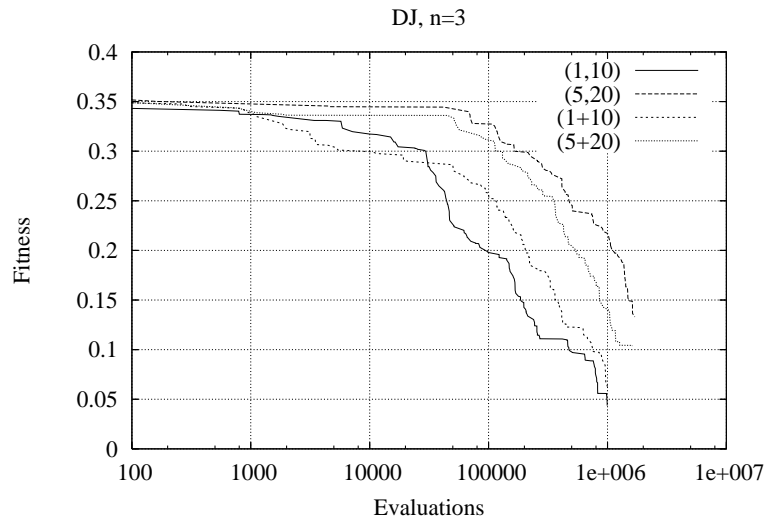
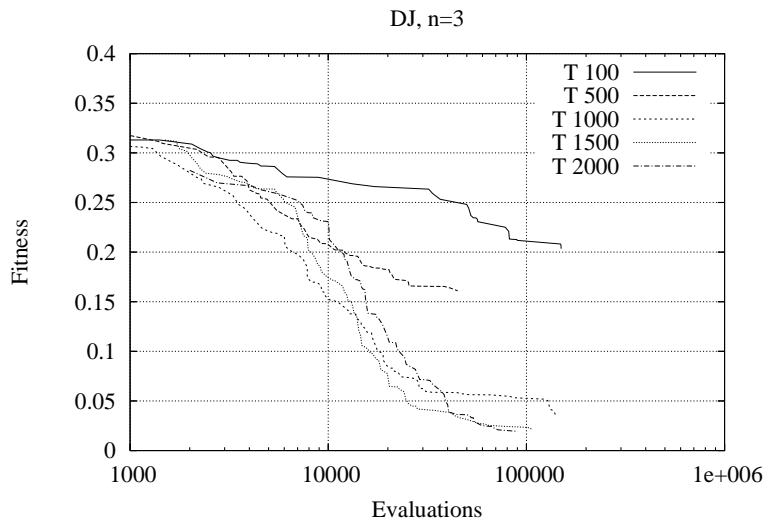


Figure 6.23: a) Different selection strategies applied to the Deutsch-Jozsa problem with  $n=2$ : tournament selection, pure  $(1,10)$  selection,  $(1,10)$  selection with step size adaptation (marked with  $*$ ) and  $(1,10)$  selection plus 10% or 20% random selection respectively. Note the logarithmic scaling on the x-axis. b) Standard deviation.



(a)



(b)

Figure 6.24: Plots of averaged evolutionary runs for the Deutsch-Jozsa problem with  $n=3$  using different selection strategies: (a) comma- and plus-strategies; (b) tournament selection for different population sizes; Note the logarithmic scaling on the x-axis.

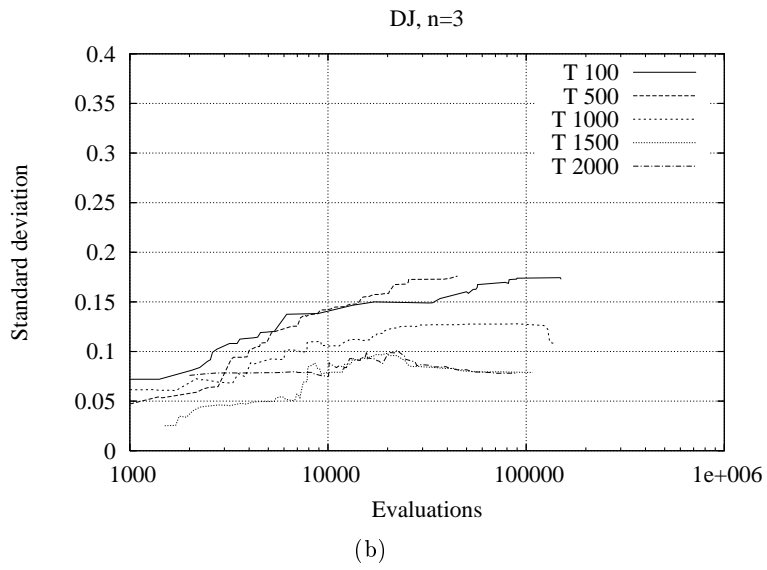
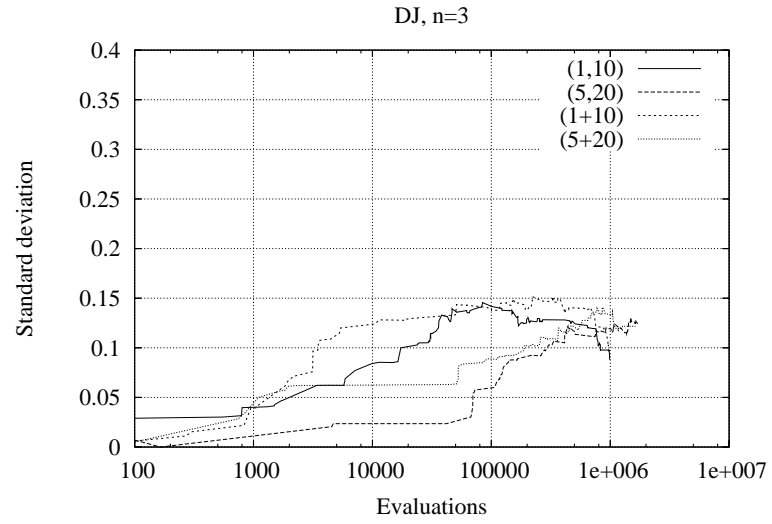


Figure 6.25: Standard deviation for the plots in Figure 6.24.

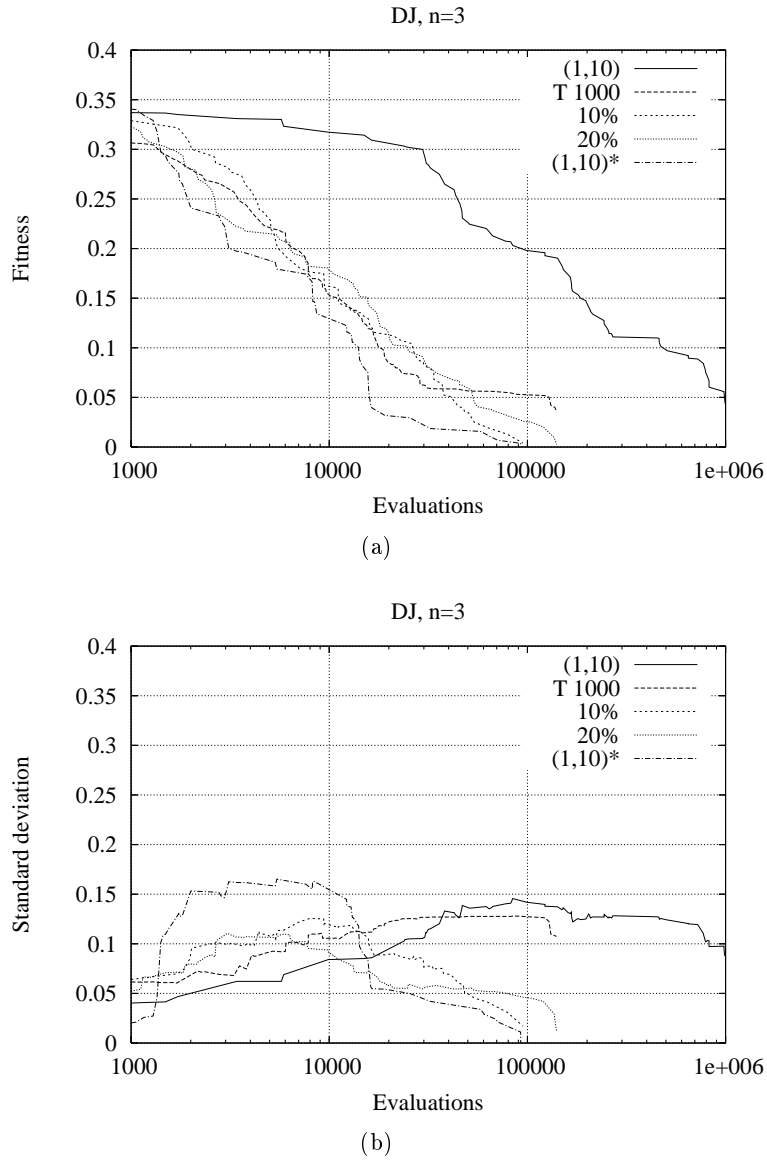


Figure 6.26: a) Behavior of evolution of quantum circuits for the Deutsch-Jozsa problem with  $n=3$  using tournament selection, pure (1,10) selection, (1,10) selection with step size adaptation (marked with \*) and (1,10) selection plus 10% or 20% random selection respectively. Note the logarithmic scaling on the x-axis. b) Standard deviation.

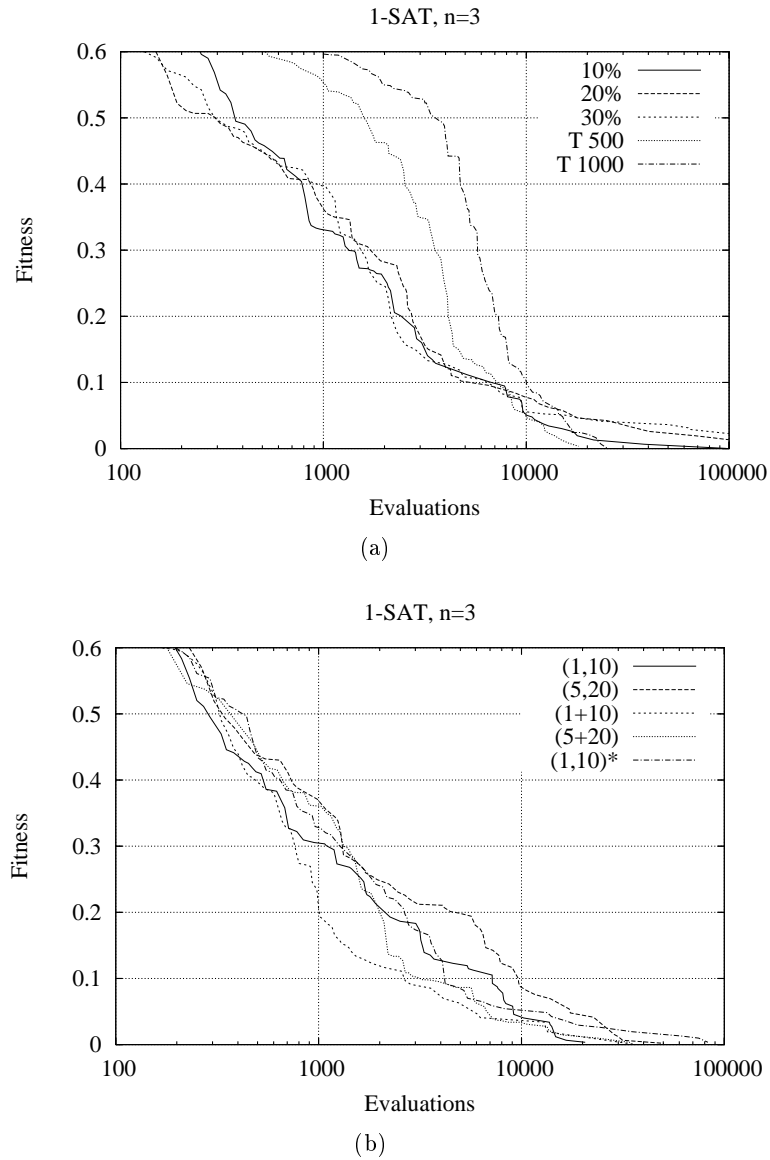
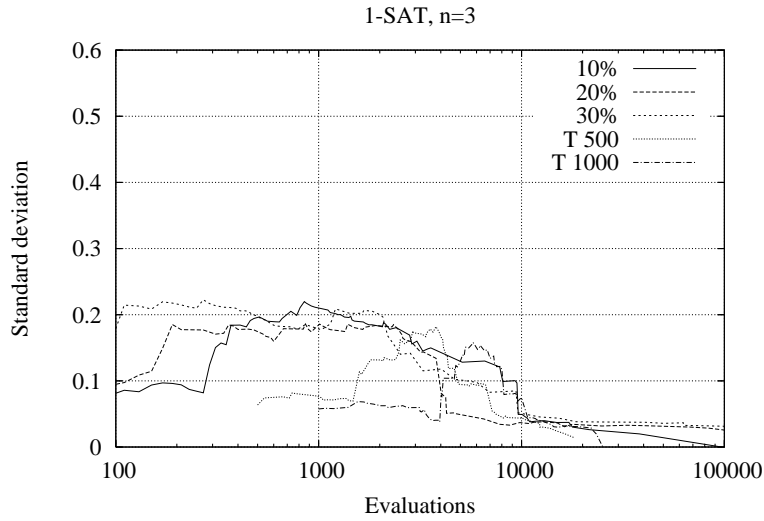
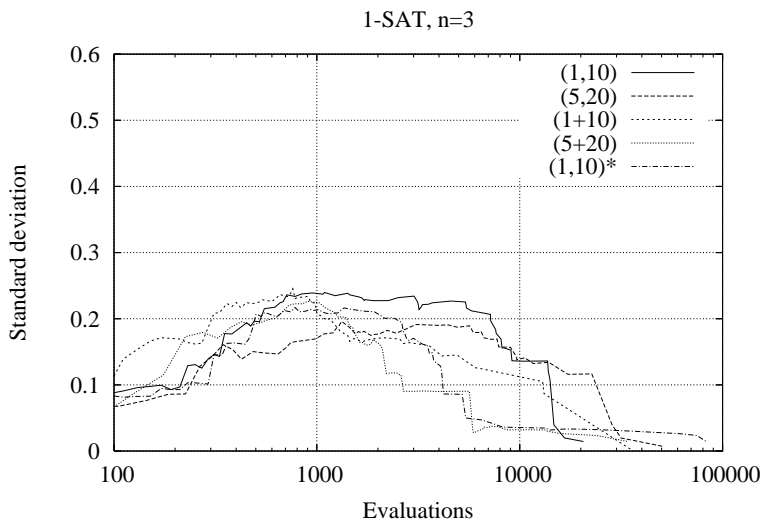


Figure 6.27: Different selection strategies for the 1-SAT problem with  $n=3$ : (a) Tournament selection for different population sizes and (1,10) selection strategy with 10% or 20% randomness, respectively. (b) (1+10), (5+20), (1,10) and (5,20) ES selection plus (1,10) selection with step size adaptation (marked with \*). Note the logarithmic scaling on the x-axis.



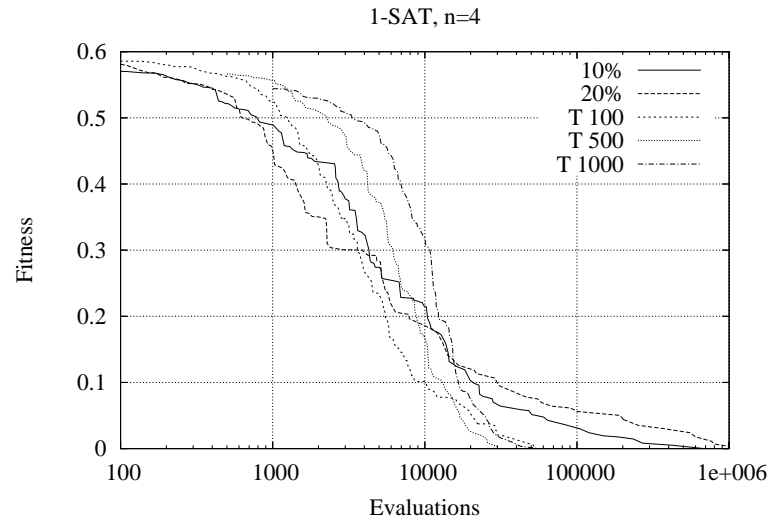


(a)

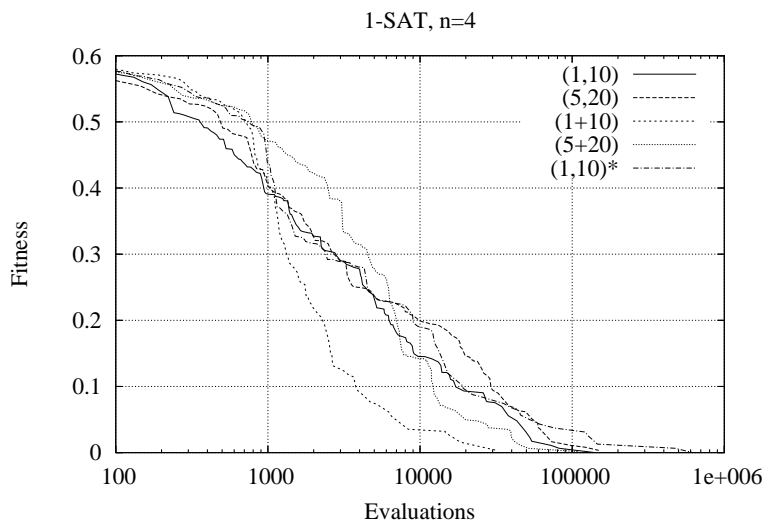


(b)

Figure 6.28: Standard deviation for the plots in Figure 6.27. (a) Tournament selection for different population sizes and  $(1,10)$  selection strategy with 10% or 20% randomness, respectively. (b)  $(1+10)$ ,  $(5+20)$ ,  $(1,10)$  and  $(5,20)$  ES selection plus  $(1,10)$  selection with step size adaptation (marked with \*).

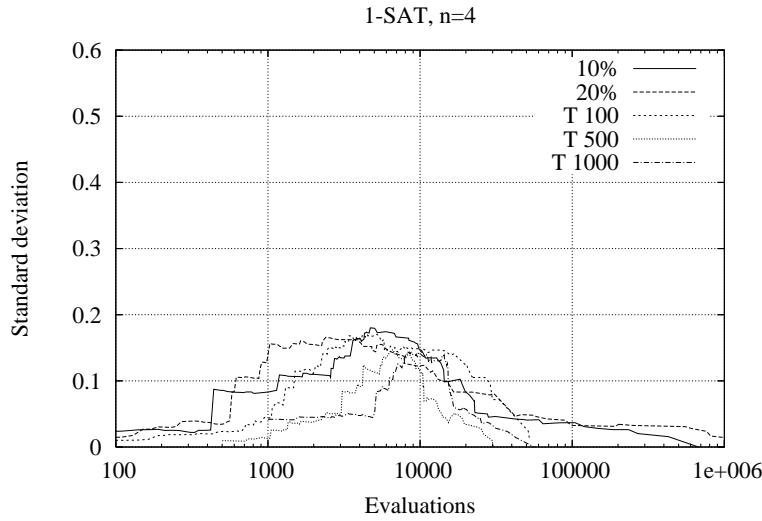


(a)

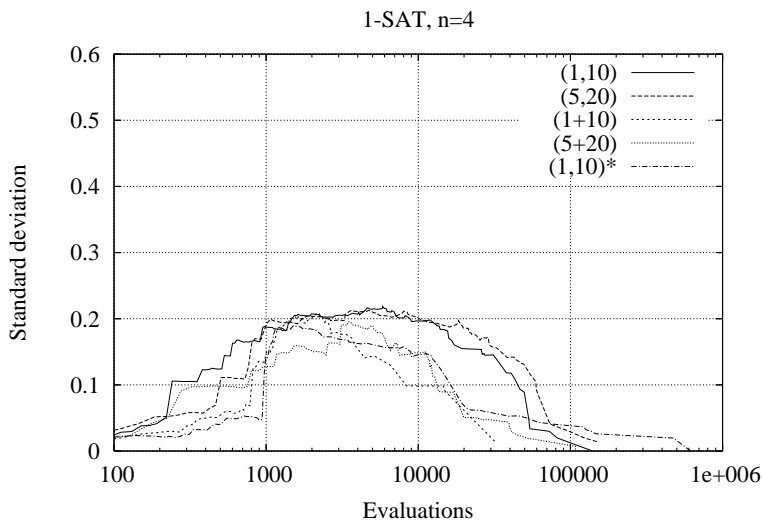


(b)

Figure 6.29: Different selection strategies for the 1-SAT problem with  $n=4$ : (a) Tournament selection for different population sizes and (1,10) selection strategy with 10% or 20% randomness, respectively. (b) (1+10), (5+20), (1,10) and (5,20) ES selection plus (1,10) selection with step size adaptation (marked with \*). Note the logarithmic scaling on the x-axis.



(a)



(b)

Figure 6.30: Standard deviation for the plots in Figure 6.29. (a) Tournament selection for different population sizes and (1,10) selection strategy with 10% or 20% randomness, respectively. (b) (1+10), (5+20), (1,10) and (5,20) ES selection plus (1,10) selection with step size adaptation (marked with \*).



## 7 Discussion and Outlook

*In order to attain the impossible  
one must attempt the absurd.*

---

Miguel de Unamuno y Jugo  
(spanish philosopher, 1864–1936)

*Difficulties seem to exist only in  
order to be overcome.*

---

E. T. A. Hoffmann (1776–1822)

The *evolution of quantum algorithms using genetic programming* causes difficulties which are not easy to overcome. Originally, starting with the challenging objective of evolving *novel* quantum algorithms which solve relevant problems such as *approximate pattern matching*<sup>1</sup> better than any classical algorithm, it did not take long to realize that this aim is for many reasons not within reach — at least not yet. The conclusion is disillusioning: With only a few exceptions, the evolutionary search approach did not lead to new realizations so far. In contrast to this, manual quantum computer programming had much more success though further breakthrough solutions comparable to Shor’s factorization algorithm or Grover’s search algorithm did not follow. Of course, this raises the question whether other “killer” applications of quantum computing exist at all — a question which could not be answered in this thesis.

### Summary

In the course of this thesis two GP-systems were developed: one based on purely linear genome structures, the other based on linear-tree genome structures allowing the use of intermediate measurements. The latter was built to achieve more “degrees of freedom” in the construction and evolution of quantum circuits compared to stricter linear GP schemes. However, this turned out to be of no benefit for the evolutionary process, probably because of the simplicity of the problem solutions. Yet, it was possible to evolve quantum circuits. Apart from the result that evolution of (small) quantum algorithms is possible using the linear and linear-tree GP system (it was demonstrated already by Lee Spector that it is in principle possible), a simpler implementation for Hogg’s

---

<sup>1</sup>For this problem, both automatic programming and manual quantum circuit design failed.

mixing matrix was found. Moreover, although not explicitly designed to evolve scalable quantum algorithms, scalability of the evolved quantum algorithms was evident and visible. In addition, it could be shown that by using pre-evolved initial populations, evolution is accelerated and (even more) forced to produce scalable quantum algorithms. Focusing on the performance of quantum circuit evaluation, the quantum simulator was optimized based on considerations summarized in Section 5.1. The next step aimed at improving the efficiency of evolutionary search. Due to major obstacles identified as the exponential growth of computational costs, the huge search spaces, and complex problem landscapes making evolutionary search very difficult, evolution has to perform effectively. The investigations lead to the exclusion of the crossover operator, which was proved to be detrimental to the evolution of quantum circuits for the Deutsch-Jozsa problem and 1-SAT. For both problems, the structure of mutation landscapes was analyzed using autocorrelation functions and information measures for characterization. The relationship between landscape characteristics and quantum algorithm evolution can be useful for improving the efficiency of the search process. Here, it is shown to be useful to explain different behaviors of evolution resulting from a comparison of different selection strategies for evolutionary quantum circuit design.

### Future Work

The experiments performed so far convey an “impression” of the evolvability of quantum circuits using genetic programming. However, it seems unlikely that the search spaces and fitness landscapes of DJ and 1-SAT are in any way representative of other problem search spaces. For the lack of existing test problems this is hard to decide.

A big problem is to find suitable test problems having instances which only need a few qubits, including ancillary qubits, and/or have only a small number of fitness cases. A few suggestions are already made in Section 5.2.1, however, this does not seem to be sufficient to obtain further insights and to help quantum computing find relevant applications.

There are still further ways to improve and to modify the GP-systems. For example, not considered up to now are suitable error models since an error-free circuit model of computation was assumed. Decoherence and faulty quantum operations must be considered if a more realistic quantum computational model is demanded. Unfortunately, this will again slow down the evolution. The quantum circuit model is based on pure quantum states. Evolution on the basis of density matrices is another approach which already showed promising results [142]. Another consideration refers to the role of entanglement and Vidal’s method simulating slightly entangled quantum computations efficiently on a classical computer. Is this also a suitable approach to boost the simulation and quantum circuit evaluation?

Some predictions of trends and developments of technology made in the past and disproven today give reference that it is difficult and dangerous to predict what the future has in store for certain technologies. However, a major breakthrough in quantum soft-

---

and hardware is not to be realistically expected (at least) in the near future. A special case of quantum computing which seems to be more feasible technologically and of more practical relevance is *reversible computing* “quantum computing’s practical cousin” [55]. The benefits and differences of reversible computing compared to quantum computing are explained at great length in [55]. Briefly, in the physical sense, reversible computing means ideally computing without (hardly any) energy consumption. That is, considering computing as a conversion of information reversible gates operate without loss of information. Instead, irreversible operations always lead to lost or erased information. That is the reason why reversible computing will require less energy and will emit less heat, both of which will reduce limiting factors of conventional irreversible computing. It remains to be seen whether “reversible computing will be the foundation for most 21st-century computer engineering”, as predicted in [55].

### **GP & QC: A Common Future?**

Genetic Programming is suitable for circuit analysis (the decomposition of quantum circuits or matrices into elementary gates) as shown by Williams and Gray (cf. Section 5.3.1) and confirmed by experiments on the decomposition of the QFT matrix. Beyond this, the synthesis of quantum algorithms is possible but only for rather small quantum systems. In isolated cases this might be possible for more than seven or eight qubits, in most cases with problems having huge numbers of fitness cases this is only possible for smaller quantum systems. And with only such a few quantum bits being available, it is difficult to find problems with sufficiently small instances. The evolution of quantum circuits for in some ways “artificial” problems such as Deutsch’s or Simon’s might help to gain solutions for more relevant problems.

Perhaps, genetic programming and quantum computing will have a brighter common future, as soon as quantum programs do not have to be simulated on classical computers, but can be tested on true quantum computers.

Alex Hamilton on a plenary debate session on quantum computing, June 2003 [1]:

*“We don’t have many quantum algorithms, but that’s okay, we don’t have that many quantum computers to run them on just yet.”*





## About the author

From 1993 to 1999 studied computer science at the University of Dortmund, Germany, attaining a diploma in computer science (Dipl.-Inform.). The study focused on systems analysis, chaos theory and technical optimization. The diploma thesis dealt with a subject in biocomputing. In parallel, from 1994 to 1996, studied mathematics finishing with a pre-degree. Following these studies, from 2000 to 2003, was a scientific associate in the group of Prof. Dr. Wolfgang Banzhaf at the chair of systems analysis, University of Dortmund. Scientific activities in the field of non-standard computing: at first biocomputing, later on, quantum computing in connection with genetic programming. From 2001 to 2003 scholarship holder within the Ph.D. program *Materials and Concepts for Quantum Information Processing*. Early in 2004, was a visiting research student for three month at the Memorial University of Newfoundland, Canada.

## Publications

A. Leier and C. Richter and W. Banzhaf and H. Rauhe, *Cryptography with DNA binary strands*, BioSystems, 57, pp. 13–22, 2000.

A. Leier and W. Banzhaf, *Evolving Hogg's Quantum Algorithm Using Linear-Tree GP*, in GECCO-03: Proceedings of the Genetic and Evolutionary Computation Conference, Part I, E. Cantú-Paz et al., eds., vol. 2723 of LNCS, pp. 390–400, Springer, 2003.

A. Leier and W. Banzhaf, *Exploring the Search Space of Quantum Programs*, in Proceedings of the 2003 Congress on Evolutionary Computation, R. Sarker et al., eds., vol. I, pp. 170–177, IEEE Computer Society Press, 2003.

A. Leier and W. Banzhaf, *Comparison of Selection Strategies for Evolutionary Quantum Circuit Design*, in GECCO-04: Proceedings of the Genetic and Evolutionary Computation Conference, LNCS, Springer, 2004.



# Bibliography

- [1] D. Abbott, *Dreams versus Reality: Plenary Debate Session on Quantum Computing*, Jun. 2003, LANL e-preprint quant-ph/0310130. Part of SPIE's First International Symposium on Fluctuations and Noise (FaN'03).
- [2] D. Abrams and S. Lloyd, *Simulation of many-body Fermi systems on a universal quantum computer*, Phys. Rev. Lett., 79 (13), pp. 2586–2589, Sep. 1997, LANL e-preprint quant-ph/9703054.
- [3] ACM, ed., *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, Heraklion, Crete, Greece, ACM Press, New York, Jul. 2001.
- [4] D. Aharonov, *Quantum Computing*, in Annual Reviews of Computational Physics VI, D. Stauffer, ed., ch. 4, pp. 259–346, World Scientific, Singapore, 1999, LANL e-preprint quant-ph/9812037.
- [5] A. Aho and K. Svore, *Compiling quantum circuits using the palindrome transform*, Nov. 2003, LANL e-preprint quant-ph/0311008.
- [6] P. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzal, eds., *Proceedings of the 1999 Congress on Evolutionary Computation*, Washington DC, USA, IEEE Computer Society Press, Silver Spring, MD, USA, Jul. 1999.
- [7] W. Banzhaf, *Genetic programming for pedestrians*, in Proceedings of the 5th International Conference on Genetic Algorithms (ICGA), S. Forrest, ed., p. 628, Morgan Kaufmann Publishers, San Francisco, 1993.
- [8] W. Banzhaf, P. Nordin, R. Keller, and F. Francone, *Genetic Programming - An Introduction*, dpunkt Heidelberg and Morgan Kaufmann Publishers, San Francisco, 1998.
- [9] H. Barnum, H. Bernstein, and L. Spector, *Better-than-classical circuits for OR and AND/OR found using genetic programming*, Jul. 1999, LANL e-preprint quant-ph/9907056.
- [10] H. Barnum, H. Bernstein, and L. Spector, *Quantum circuits for OR and AND of ORs*, J. Phys. A: Math. Gen., 33 (45), pp. 8047–8057, Nov. 2000.

- [11] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf, *Quantum lower bounds by polynomials*, in IEEE [77], pp. 352–361, LANL e-preprint quant-ph/9802049.
- [12] P. Benioff, *The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines*, J. Stat. Phys., 22, pp. 563–591, 1980.
- [13] C. Bennett, *Logical reversibility of computation*, IBM Journal of Research and Development, 17 (6), pp. 525–530, 1973.
- [14] C. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, *Strengths and weaknesses of quantum computing*, SIAM J. Comput., 26 (5), pp. 1510–1523, Oct. 1997, LANL e-preprint quant-ph/9701001.
- [15] C. Bennett, G. Brassard, C. Crepeau, R. Jozsa, A. Peres, and W. Wootters, *Teleporting an unknown quantum state via dual classical and Einstein-Prodolsky-Rosen channels*, Phys. Rev. Lett., 70 (13), pp. 1895–1899, Mar. 1993.
- [16] E. Bernstein and U. Vazirani, *Quantum complexity theory*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC), ACM, ed., pp. 11–20, ACM Press, New York, May 1993.
- [17] E. Bernstein and U. Vazirani, *Quantum complexity theory*, SIAM J. Comput., 26 (5), pp. 1411–1473, Oct. 1997. Preliminary version appeared in Proceedings of 25th Annual ACM Symposium on Theory of Computing (STOC), May 1993.
- [18] E. Biham, O. Biham, D. Biron, M. Grassl, D. Lidar, and D. Shapira, *Analysis of generalized Grover quantum search algorithms using recursion equations*, Phys. Rev. A, 63 (1), Jan. 2001. Article 012310.
- [19] D. Boneh and R. Lipton, *Quantum cryptanalysis of hidden linear functions*, in Advances in Cryptology – CRYPTO’95, Proceedings of the 15th Annual International Cryptology Conference, D. Coppersmith, ed., vol. 963 of LNCS, pp. 424–437, Springer, New York, Aug. 1995.
- [20] J. Boye and J. Małuszyński, *Rewriting systems*, 2003. Lecture notes taught at Linköping University, <http://www.ida.liu.se/~TDDB40/>.
- [21] M. Boyer, G. Brassard, P. Høyer, and A. Tapp, *Tight bounds on quantum searching*, in Toffoli et al. [149], pp. 36–43, LANL e-preprint quant-ph/9605034.
- [22] G. Brassard, *Teleportation as a quantum computation*, in Toffoli et al. [149], pp. 48–50.

- [23] G. Brassard and P. Høyer, *An exact quantum polynomial-time algorithm for Simon's problem*, in Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems (ISTCS), IEEE, ed., pp. 12–23, IEEE Computer Society Press, Silver Spring, MD, USA, Jun. 1997.
- [24] G. Brassard, P. Høyer, and A. Tapp, *Quantum counting*, May 1998, LANL e-preprint quant-ph/9805082.
- [25] P. Bürgisser, *Algebraische Komplexitätstheorie II - Schnelle Matrixmultiplikation und Kombinatorik*, Séminaire Lotharingien De Combinatoire, 36 (B36b), 1996.
- [26] H. Buhrman and R. de Wolf, *Complexity measures and decision tree complexity: A survey*, Theoretical Computer Science, 288, pp. 21–43, 2002.
- [27] H. Buhrman, C. Dürr, M. Heiligman, P. Høyer, F. Magniez, M. Santha, and R. de Wolf, *Quantum algorithms for element distinctness*, Jul. 2000, LANL e-preprint quant-ph/0007016.
- [28] J. Busch, J. Ziegler, W. Banzhaf, A. Ross, D. Sawitzki, and C. Aue, *Automatic Generation of Control Programs for Walking Robots using Genetic Programming*, in Lutton et al. [101], pp. 259–268.
- [29] V. Bužek, S. Braunstein, M. Hillery, and D. Bruß, *Quantum copying: a network*, Phys. Rev. A, 56 (5), pp. 3446–3452, 1997, LANL e-preprint quant-ph/9703046.
- [30] K. Cheung and M. Mosca, *Decomposing finite Abelian groups*, Jan. 2001, LANL e-preprint cs.DS/0101004.
- [31] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca, *Quantum algorithms revisited*, Proc. R. Soc. London A, 454 (1969), pp. 339–354, 1998, LANL e-preprint quant-ph/9708016.
- [32] H. Cohen, *A Course in Computational Algebraic Number Theory*, vol. 138 of Graduate Texts in Mathematics, Springer-Verlag, 1993.
- [33] J. Cohen and M. Roth, *On the implementation of Strassen's fast multiplication algorithm*, Acta Informatica, 6, pp. 341–355, 1976.
- [34] D. Collins, K. Kim, and W. Holton, *Deutsch-Jozsa algorithm as a test of quantum computation*, Phys. Rev. A, 58 (3), pp. R1633–R1636, Sep. 1998.
- [35] D. Coppersmith, *An approximate Fourier transform useful in quantum factoring*, Tech. Rep. RC 19642, IBM, 1994.

- [36] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC), ACM, ed., pp. 1–6, ACM Press, New York, May 1987.
- [37] G. Cybenko, *Reducing quantum computations to elementary unitary operations*, Computing in Science and Engineering, 3 (2), pp. 27–32, 2001.
- [38] J. Daida, T. Bersano-Begey, S. Ross, and J. Vesecky, *Computer-assisted design of image classification algorithms: Dynamic and static fitness evaluations in a scaffolded genetic programming environment*, in Koza et al. [89], pp. 279–284.
- [39] C. Darwin, *On the Origin of Species by Means of Natural Selection or the Preservation of Favoured Races in the Struggle for Life*, Murray, London, UK, 1859.
- [40] Y. Davidor, H.-P. Schwefel, and R. Männer, eds., *Parallel Problem Solving from Nature – PPSN III, International Conference on Evolutionary Computation*, vol. 866 of Lecture Notes in Computer Science, Jerusalem, Israel, Springer, Berlin, Oct. 1994.
- [41] N. Dershowitz, *A taste of rewrite systems*, in Functional Programming, Concurrency, Simulation and Automated Reasoning, P. Lauer, ed., vol. 693 of LNCS, pp. 199–228, Springer, Berlin, 1993.
- [42] N. Dershowitz and J.-P. Jouannaud, *Rewrite systems*, in Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, J. van Leeuwen, ed., Elsevier, 1990, pp. 243–320.
- [43] D. Deutsch, *Quantum theory, the Church-Turing principle and the universal quantum computer*, Proc. R. Soc. London A, 400, pp. 97–117, 1985.
- [44] D. Deutsch, *Quantum computational networks*, Proc. R. Soc. London A, 425, pp. 73–90, 1989.
- [45] D. Deutsch, *The Fabric of Reality*, Penguin Books, 1997.
- [46] D. Deutsch and R. Jozsa, *Rapid solution of problems by quantum computation*, Proc. R. Soc. London A, 439, pp. 553–558, 1992.
- [47] P. A. M. Dirac, *The Principles of Quantum Mechanics*, Oxford, 1958.
- [48] C. Dürr and P. Høyer, *A quantum algorithm for finding the minimum*, Jul. 1996, LANL e-preprint quant-ph/9607014.
- [49] M. Ettinger and P. Høyer, *On quantum algorithms for noncommutative hidden subgroups*, in Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS), C. Meinel and S. Tison, eds., vol. 1563 of LNCS, pp. 478–487, Springer, Berlin, Mar. 1999, LANL e-preprint quant-ph/9807029.

- [50] M. Ettinger, P. Høyer, and E. Knill, *Hidden subgroup states are almost orthogonal*, Jan. 1999, LANL e-preprint quant-ph/9901034.
- [51] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, *A limit on the speed of quantum computation for insertion into an ordered list*, Dec. 1998, LANL e-preprint quant-ph/9812051.
- [52] A. Fijany and C. Williams, *Quantum Wavelet Transforms: Fast Algorithms and Complete Circuits*, in Williams [165], pp. 10–33.
- [53] L. Fogel, A. Owens, and M. Walsh, *Artificial Intelligence through a Simulation of Evolution*, in Biophysics and Cybernetic Systems: Proceedings of the 2nd Cybernetic Sciences Symposium, M. Maxfield, A. Callahan, and L. Fogel, eds., pp. 131–155, Spartan Books, Washington, D.C., 1965.
- [54] L. Fogel, A. Owens, and M. Walsh, *Artificial Intelligence through Simulated Evolution*, John Wiley & Sons, New York, USA, 1966.
- [55] M. Frank, *Reversible Computing: Quantum Computing's Practical Cousin*, May 2003. Invited talk, J.H. Simons Conference on Quantum and Reversible Computation, Stony Brook, NY.
- [56] C. Gathercole and P. Ross, *Dynamic training subset selection for supervised learning in genetic programming*, in Davidor et al. [40], pp. 312–321.
- [57] Y. Ge, L. Watson, and E. Collins, *Genetic algorithms for optimization on a quantum computer*, in Proceedings of the 1st International Conference on Unconventional Models of Computation (UMC), C. Calude, J. Casti, and M. Dinneen, eds., DMTCS, pp. 218–227, Springer, Singapur, Jan. 1998.
- [58] T. Gramß, S. Bornholdt, M. Groß, M. Mitchell, and T. Pellizzari, eds., *Non-Standard Computation - Molecular Computation, Cellular Automata, Evolutionary Algorithms, Quantum Computers*, Wiley-VCH, 1998.
- [59] M. Grigni, L. Schulman, M. Vazirani, and U. Vazirani, *Quantum Mechanical Algorithms for the Nonabelian Hidden Subgroup Problem*, in ACM [3], pp. 68–74.
- [60] L. Gritz and J. Hahn, *Genetic programming for articulated figure motion*, Journal of Visualization and Computer Animation, 6, pp. 129–142, 1995.
- [61] L. Grover, *A fast quantum mechanical algorithm for database search*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC), ACM, ed., pp. 212–219, ACM Press, New York, May 1996, LANL e-preprint quant-ph/9605043.
- [62] L. Grover, *Quantum mechanics helps in searching for a needle in a haystack*, Phys. Rev. Lett., 79, pp. 325–328, Jul. 1997, LANL e-preprint quant-ph/9706033.

- [63] L. Grover, *Quantum search on structured problems*, Chaos, Solitons, and Fractals, Special Issue on Quantum Computing, 10 (10), pp. 1695–1705, Jun. 1999, LANL e-preprint quant-ph/9802035. Earlier version in Proceedings of the First NASA QCQC Conference.
- [64] L. Grover, *An improved quantum scheduling algorithm*, Feb. 2002, LANL e-preprint quant-ph/0202033.
- [65] J. Gruska, *Quantum Computing*, McGraw-Hill, London, 1999.
- [66] L. Hales and S. Hallgren, *An improved quantum Fourier transform algorithm and applications*, in Proceedings of the 41st Annual IEEE Symp. on Foundations of Computer Science (FOCS), IEEE, ed., pp. 515–525, IEEE Computer Society Press, Silver Spring, MD, USA, Nov. 2000.
- [67] S. Hallgren, *Quantum Fourier Sampling, the Hidden Subgroup Problem and Beyond*, PhD thesis, University of California, Berkeley, 2000.
- [68] S. Hallgren, *Polynomial-time quantum algorithms for Pell’s equation and the principal ideal problem*, in Proceedings of the 34rd Annual ACM Symposium on Theory of Computing (STOC), ACM, ed., ACM Press, New York, May 2002.
- [69] S. Hallgren, A. Russell, and A. Ta-Shma, *The hidden subgroup problem and quantum computation using group representations*, Aug. 2001. Preliminary version appeared in Proceedings of 32nd Annual ACM Symposium on Theory of Computing (STOC), May 2000.
- [70] M. Heiligman, *Finding matches between two databases on a quantum computer*, Jun. 2000. LANL e-preprint quant-ph/0006136.
- [71] M. Hirvensalo, *Quantum Computing*, Natural Computing Series, Springer, Berlin, 2001.
- [72] T. Hogg, *Highly structured searches with quantum computers*, Phys. Rev. Lett., 80 (11), pp. 2473–2476, Mar. 1998.
- [73] T. Hogg, *Solving highly constrained search problems with quantum computers*, J. Artificial Intelligence Res., 10, pp. 39–66, Feb. 1999.
- [74] J. Holland, *Adaption in natural and artificial systems*, MIT Press, Cambridge, MA, USA, 1992.
- [75] P. Høyer, J. Neerbek, and Y. Shi, *Quantum complexities of ordered searching, sorting, and element distinctness*, Feb. 2001, LANL e-preprint quant-ph/0102078.



- [76] IEEE, ed., *Proceedings of the 35th Annual IEEE Symp. on Foundations of Computer Science (FOCS)*, Santa Fee, New Mexico, USA, IEEE Computer Society Press, Silver Spring, MD, USA, Nov. 1994.
- [77] IEEE, ed., *Proceedings of the 39th Annual IEEE Symp. on Foundations of Computer Science (FOCS)*, Palo Alto, CA, USA, IEEE Computer Society Press, Silver Spring, MD, USA, Nov. 1998.
- [78] ITRS, *International Technology Roadmap for Semiconductors, 2003 Edition, Executive Summary*, 2003. <http://public.itrs.net/Files/2003ITRS/Home2003.htm>.
- [79] G. Ivanyos, F. Magniez, and M. Santha, *Efficient algorithms for some instances of the non-abelian hidden subgroup problem*, Feb. 2001, LANL e-preprint quant-ph/0102014.
- [80] R. Jozsa, *Quantum algorithms and the Fourier transform*, Proc. R. Soc. London A, 454, pp. 323–337, 1998, LANL e-preprint quant-ph/9707033.
- [81] R. Jozsa and N. Linden, *On the role of entanglement in quantum computational speed-up*, Mar. 2002, LANL e-preprint quant-ph/0201143.
- [82] W. Kantschik and W. Banzhaf, *Linear-tree GP and its comparison with other GP structures*, in Miller et al. [102], pp. 302–312.
- [83] W. Kantschik and W. Banzhaf, *Linear-Graph GP – A new GP Structure*, in Lutton et al. [101], pp. 83–92.
- [84] C. Kirchner and H. Kirchner, *Rewriting solving proving*, 2001. Preliminary version, <http://www.loria.fr/~ckirchne/rsp.pdf>.
- [85] A. Kitaev, *Quantum measurements and the Abelian stabilizer problem*, Tech. rep., L.D. Landau Institute for Theoretical Physics, Moscow, 1995, LANL e-preprint quant-ph/9511026.
- [86] A. Klappenecker and M. Rötteler, *Discrete cosine transforms on quantum computers*, Nov. 2001, LANL e-preprint quant-ph/0111038.
- [87] J. Koza, *Genetic Programming: On the Programming of Computers by Natural Selection*, MIT Press, Cambridge, MA, USA, 1992.
- [88] J. Koza, *Genetic Programming II*, MIT Press, Cambridge, MA, USA, 1994.
- [89] J. Koza, D. Goldberg, D. Fogel, and R. Riolo, eds., *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, USA, MIT Press, Cambridge, MA, USA, 1996.

- [90] J. Koza, F. B. III, D. Andre, and M. Keane, *Automated WYSIWYG design of both the topology and component values of electrical circuits using genetic programming*, in Koza et al. [89], pp. 123–131.
- [91] J. Koza, F. B. III, D. Andre, and M. Keane, *Four problems for which a computer program evolved by genetic programming is competitive with human performance*, in *Proceedings of the 1996 Congress on Evolutionary Computation*, pp. 1–10, IEEE Computer Society Press, New York, 1996.
- [92] R. Ladner, *On the structure of polynomial time reducibility*, *Journal of the ACM*, 22, pp. 155–171, 1975.
- [93] R. Landauer, *Information is physical*, *Physics Today*, 44, pp. 23–29, 1991.
- [94] W. Langdon, *The Distribution of Reversible Functions is Normal*, in *Genetic Programming - Theory and Practice*, R. Riolo and B. Worzel, eds., vol. 6 of *Genetic Programming*, pp. 173–188, Kluwer Academic Publishers, Boston, 2003.
- [95] C. Lasarczyk, *Trainingsmengenselektion auf der Grundlage einer Fitnesscase-Topologie*, Diploma thesis, University of Dortmund, Germany, 2002.
- [96] C. Lasarczyk, P. Dittrich, and W. Banzhaf, *Dynamic subset selection based on a fitness case topology*, *Evolutionary Computation*, 12 (2), 2004.
- [97] A. Leier and W. Banzhaf, *Evolving Hogg’s quantum algorithm using linear-tree GP*, in *GECCO-03: Proceedings of the Genetic and Evolutionary Computation Conference, Part I*, E. Cantú-Paz, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. Potter, A. Schultz, N. Jonoska, K. Dowsland, J. Miller, J. Foster, K. Deb, D. Lawrence, R. Roy, U.-M. O’Reilly, H.-G. Beyer, R. Standish, and G. Kendall, eds., vol. 2723 of *LNCS*, pp. 390–400, Springer, Jul. 2003.
- [98] A. Leier and W. Banzhaf, *Exploring the search space of quantum programs*, in *Proceedings of the 2003 Congress on Evolutionary Computation*, R. Sarker, R. Reynolds, H. Abbass, K. Tan, B. McKay, D. Essam, and T. Gedeon, eds., vol. I, pp. 170–177, IEEE Computer Society Press, Piscataway, NJ, USA, Dec. 2003.
- [99] A. Leier and W. Banzhaf, *Comparison of selection strategies for evolutionary quantum circuit design*, in *GECCO-04: Proceedings of the Genetic and Evolutionary Computation Conference*, *LNCS*, Springer, Jun. 2004.
- [100] S. Lloyd, *Quantum search without entanglement*, *Phys. Rev. A*, 61 (1), Jan. 2000. Article 010301.
- [101] E. Lutton, J. Foster, J. Miller, C. Ryan, and A. Tettamanzi, eds., *Proceedings of the 5th European Conference on Genetic Programming (EUROGP)*, vol. 2278 of *LNCS*, Kinsale, Ireland, Springer, Berlin, Apr. 2002.

- [102] J. Miller, M. Tomassini, P. Lanzi, C. Ryan, A. Tettamanzi, and W. Langdon, eds., *Proceedings of the 4th European Conference on Genetic Programming (EUROGP)*, vol. 2038 of LNCS, Lake Como, Italy, Springer, Berlin, Apr. 2001.
- [103] T. Mitchell, *Machine Learning*, Mc-Graw-Hill, New York, USA, 1996.
- [104] M. Mosca, *Quantum searching, counting and amplification by eigenvector analysis*, in Proceedings of International Workshop on Randomized Algorithms, R. Freivalds, ed., pp. 90–100, Aachen University Press, Aug. 1998.
- [105] M. Mosca and A. Ekert, *The hidden subgroup problem and eigenvalue estimation on a quantum computer*, in Williams [165], pp. 174–188, LANL e-preprint quant-ph/9903071.
- [106] A. Narayanan and J. Wallace, *A quantum algorithm for route finding*, in Proceedings of the 15th European Meeting on Cybernetics and Systems Research (EMCSR), R. Trappl, ed., vol. 1, pp. 140–143, Austrian Society for Cybernetic Studies, Apr. 2000.
- [107] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.
- [108] H. Nishimura and M. Ozawa, *Computational complexity of uniform quantum circuit families and quantum Turing machines*, Theoretical Computer Science, 276 (1-2), pp. 147–181, Apr. 2002.
- [109] P. Nordin, *A compiling genetic programming system that directly manipulates the machine code*, in *Advances in Genetic Programming*, ch. 14, pp. 311–331, MIT Press, Cambridge, MA, USA, 1994.
- [110] P. Nordin, *Evolutionary Program Induction of Binary Machine Code and its Application*, Krehl-Verlag, Münster, Germany, 1997.
- [111] P. Nordin and W. Banzhaf, *Genetic programming controlling a miniature robot*, in Working Notes for the AAAI Symposium on Genetic Programming, E. Siegel and J. Koza, eds., pp. 61–67, AAAI, Menlo Park, CA, 1995.
- [112] P. Nordin and W. Banzhaf, *An on-line method to evolve behaviour and to control a miniature robot in real time with genetic programming*, Adaptive Behaviour, 5 (2), pp. 107–140, 1997.
- [113] P. Nordin and W. Banzhaf, *Real time control of a khepera robot using genetic programming*, Cybernetics and Control, 26 (3), pp. 533–561, 1997.

- [114] P. Nordin, F. Francone, and W. Banzhaf, *Explicitly defined introns and destructive crossover in genetic programming*, in Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications, J. Rosca, ed., pp. 6–22, Tahoe City, California, USA, Jul. 1995.
- [115] B. Ömer, *Quantum Programming in QCL*, Master thesis, Technical University of Vienna, Institute of Information Systems, Jan. 2000.
- [116] G. Ortiz, J. Gubernatis, E. Knill, and R. Laflamme, *Quantum algorithms for Fermionic simulations*, Dec. 2000, LANL e-preprint quant-ph/0012334.
- [117] X. Peng, X. Zhu, X. Fang, M. Feng, M. Liu, and K. Gao, *Experimental implementation of Hogg’s algorithm on a three-quantum-bit NMR quantum computer*, Phys. Rev. A, 65 (042315), Apr. 2002.
- [118] T. Perkis, *Stack-based genetic programming*, in Proceedings of the 1994 IEEE World Congress on Computational Intelligence, IEEE, ed., vol. 1, pp. 148–153, IEEE Computer Society Press, Jun. 1994.
- [119] D. Poulin, *Classicality of quantum information processing*, Phys. Rev. A, 65 (042319), Apr. 2002, LANL e-preprint quant-ph/0108102.
- [120] J. Preskill, *Fault-Tolerant Quantum Computation*, in Introduction to Quantum Computation and Information, H.-K. Lo, S. Popescu, and T. Spiller, eds., pp. 213–269, World Scientific, Singapur, 1998, LANL e-preprint quant-ph/9712048.
- [121] J. Preskill, *Quantum computation*, 2000-01. Lecture notes for course Physics/CS 219 (formerly Physics 229) taught at California Institute of Technology, <http://www.theory.caltech.edu/people/preskill/ph229/>.
- [122] H. Ramesh and V. Vinay, *String matching in  $\tilde{O}(\sqrt{n} + \sqrt{m})$  quantum time*, Nov. 2000, LANL e-preprint quant-ph/0011049.
- [123] M. Raymer, W. Punch, E. Goodman, and L. Kuhn, *Genetic programming for improved data mining: An application to the biochemistry of protein interactions*, in Koza et al. [89], pp. 375–380.
- [124] I. Rechenberg, *Evolutionstrategie ’93*, Frommann Verlag, Stuttgart, Germany, 1994.
- [125] C. Reidys and P. Stadler, *Neutrality in fitness landscapes*, Tech. Rep. 98-10-089, Santa Fe Institute, 1998.
- [126] M. Rötteler and T. Beth, *Polynomial-time solutions to the hidden subgroup problem for a class of non-abelian groups*, Dec. 1998, LANL e-preprint quant-ph/9812070.

- [127] B. Rubinstein, *Evolving quantum circuits using genetic programming*, in Proceedings of the 2001 Congress on Evolutionary Computation, IEEE, ed., pp. 114–151, IEEE Computer Society Press, Silver Spring, MD, USA, May 2001. The first version of this paper already appeared in 1999.
- [128] G. Rudolph, *An evolutionary algorithm for integer programming*, in Davidor et al. [40], pp. 139–148.
- [129] B. Schneier, *Applied Cryptography*, John Wiley & Sons, New York, 1996.
- [130] M. Schulz, *The end of the road for silicon?*, Nature, 399, pp. 729–730, Jun. 1999.
- [131] H.-P. Schwefel, *Evolution and Optimum Seeking*, Sixth-Generation Computer Technology Series, John-Wiley & Sons, New York, USA, 1995.
- [132] P. Shor, *Algorithms for quantum computation: discrete logarithm and factoring*, in IEEE [76], pp. 124–134.
- [133] P. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (5), pp. 1484–1509, Oct. 1997, LANL e-preprint quant-ph/9508027.
- [134] D. Simon, *On the power of quantum computation*, in IEEE [76], pp. 116–123.
- [135] D. Simon, *On the power of quantum computation*, SIAM J. Comput., 26 (5), pp. 1474–1483, Oct. 1997.
- [136] L. Spector, *Quantum computation - a tutorial*, in GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, W. Banzhaf, J. Daida, A. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. Smith, eds., pp. 170–197, Morgan Kaufmann Publishers, San Francisco, Jul. 1999.
- [137] L. Spector, *The evolution of arbitrary computational processes*, IEEE Intelligent Systems, pp. 80–83, May/June. 2000.
- [138] L. Spector, *Automatic Quantum Computer Programming - A Genetic Programming Approach*, Kluwer Academic Publishers, 2004. Not yet published.
- [139] L. Spector, H. Barnum, H. Bernstein, and N. Swamy, *Finding a better-than-classical quantum AND/OR algorithm using genetic programming*, in Angeline et al. [6], pp. 2239–2246.
- [140] L. Spector, H. Barnum, H. Bernstein, and N. Swamy, *Quantum Computing Applications of Genetic Programming*, in Advances in Genetic Programming, L. Spector, U.-M. O’Reilly, W. Langdon, and P. Angeline, eds., vol. 3, pp. 135–160, MIT Press, Cambridge, MA, USA, 1999.

- [141] R. Stadelhofer, 2003. Personal communication.
- [142] R. Stadelhofer, *Solving The Parity Problem On A Mixed State Quantum Computer*, Tech. rep., University of Dortmund, Chair of Systems Analysis, Feb. 2004. Available on request.
- [143] P. Stadler, *Towards theory of landscapes*, in Complex-Systems and Binary Networks, R. Lopéz-Pena, R. Capovilla, R. Garcia-Pelayo, H. Waelbroeck, and F. Zertuche, eds., vol. 461 of Lecture Notes in Physics, pp. 77–163, Springer, Berlin, 1995.
- [144] A. Steane, *Quantum computation*, Reports on Progress in Physics, 61, pp. 117–173, 1998, LANL e-preprint quant-ph/9708022.
- [145] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13, pp. 354–356, 1969.
- [146] Technology Experts Panel, *A quantum information science and technology roadmap*, Tech. rep., ARDA, Dec. 2002.
- [147] A. Teller and M. Veloso, *PADO: A new learning architecture for object recognition*, in Symbolic Visual Learning, K. Ikeuchi and M. Veloso, eds., pp. 81–116, Oxford University Press, Oxford, UK, 1996.
- [148] B. Terhal and D. DiVincenzo, *The problem of equilibration and the computation of correlation functions on a quantum computer*, Oct. 1998, LANL e-preprint quant-ph/9810063.
- [149] T. Toffoli, M. Biafore, and J. Leão, eds., *Proceedings of the Fourth Workshop on Physics and Computation (PhysComp)*, Boston, MA, USA, New England Complex Systems Institute, Cambridge, MA, USA, Nov. 1996.
- [150] F. Vallentin, *Zur Komplexität des “Shortest Vector Problem” und seine Anwendungen in der Kryptographie*, Diploma thesis, University of Dortmund, Dept. of Computer Science, Aug. 1999.
- [151] W. van Dam, *Quantum oracle interrogation: getting all information for half the price*, in IEEE [77], pp. 362–367, LANL e-preprint quant-ph/9805006.
- [152] W. van Dam, *Quantum algorithms for weighing matrices and quadratic residue*, Aug. 2000, LANL e-preprint quant-ph/0008059.
- [153] W. van Dam and S. Hallgren, *Efficient quantum algorithms for shifted quadratic character problems*, Nov. 2000, LANL e-preprint quant-ph/0011067.

- 
- [154] L. Vandersypen, M. Steffen, G. Breyta, C. Yannoni, M. Sherwood, and I. Chuang, *Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance*, *Nature*, 414, pp. 883–887, Dec. 2001, LANL e-preprint quant-ph/0112176.
- [155] V. Vassilev, *An information measure of landscapes*, in Proceedings of the 7th International Conference on Genetic Algorithms (ICGA), T. Bäck, ed., pp. 49–65, Morgan Kaufmann Publishers, San Francisco, 1993.
- [156] V. Vassilev, J. Miller, and T. Fogarty, *Digital circuit evolution and fitness landscapes*, in Angeline et al. [6], pp. 1299–1306.
- [157] V. Vassilev, J. Miller, and T. Fogarty, *Information characteristics and the structure of landscapes*, *Evolutionary Computation*, 8 (1), pp. 31–60, 2000.
- [158] U. Vazirani, *Quantum computation*, 2000. Lecture notes for course CS294-6, Fall 2000, taught at University of California, Berkeley, <http://www.cs.berkeley.edu/~vazirani/quantum.html>.
- [159] G. Vidal, *Efficient classical simulations of slightly entangled quantum computations*, Feb. 2003, LANL e-preprint quant-ph/0301063.
- [160] G. Wagner and P. Stadler, *Complex adaptations and the structure of recombination spaces*, Tech. Rep. 97-03-029, Santa Fe Institute, 1998.
- [161] J. Wallace, *Quantum Computer Simulators - A Review*, 1999. Version 2.1, [citeseer.nj.nec.com/wallace99quantum.html](http://citeseer.nj.nec.com/wallace99quantum.html).
- [162] J. Watrous, *Quantum algorithms for solvable groups*, in ACM [3], pp. 60–67, LANL e-preprint quant-ph/0011023.
- [163] E. Weinberger, *Correlated and uncorrelated fitness landscapes and how to tell the difference*, *Biological Cybernetics*, 63, pp. 325–336, 1990.
- [164] S. Wiesner, *Simulations of many-body quantum systems by a quantum computer*, Mar. 1996, LANL e-preprint quant-ph/9603028.
- [165] C. Williams, ed., *Proceedings of the First NASA International Conference on Quantum Computing and Quantum Communications (QCQC)*, vol. 1509 of LNCS, Palm Springs, CA, USA, Springer, New York, Feb. 1998.
- [166] C. Williams and S. Clearwater, eds., *Explorations in Quantum Computing*, Springer, New York, 1997.
- [167] C. Williams and A. Gray, *Automated design of quantum circuits*, in Williams [165], pp. 113–125.

- [168] T. Yabuki and H. Iba, *Genetic algorithms and quantum circuit design, evolving a simpler teleportation circuit*, in GECCO-00: Proceedings of the Genetic and Evolutionary Computation Conference, D. Whitley, D. Goldberg, E. Cantú-Paz, L. Spector, I. Parmee, and H.-G. Beyer, eds., pp. 421–425, Morgan Kaufmann Publishers, San Francisco, Jul. 2000. Late-breaking papers at the GECCO 2000.
- [169] A. Yao, *Quantum circuit complexity*, in Proceedings of the 34th Annual IEEE Symp. on Foundations of Computer Science (FOCS), IEEE, ed., pp. 352–361, IEEE Computer Society Press, Silver Spring, MD, USA, Nov. 1993.
- [170] T. Yu and J. Miller, *Neutrality and the evolvability of boolean function landscapes*, in Miller et al. [102], pp. 204–217.
- [171] C. Zalka, *Efficient simulation of quantum systems by quantum computers*, Mar. 1996, LANL e-preprint quant-ph/9603026.
- [172] C. Zalka, *Simulating quantum systems on a quantum computer*, Proc. R. Soc. London A, 454 (1969), pp. 313–322, Jan. 1998.
- [173] B.-T. Zhang and D.-Y. Cho, *Genetic programming with active data selection*, in Second Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'98), B. McKay, X. Yao, C. Newton, J.-H. Kim, and T. Furuhashi, eds., vol. 1585 of LNCS, pp. 146–153, Springer, Nov. 1998.



# Index

Symbols	
$\mathcal{H}^{(n)}$	10
$\oplus$	14, 23
$\otimes$	11
$U_2$	15
$U_f$	23
$\langle \cdot   \cdot \rangle$	9
$\langle \cdot  $	8
$ \cdot\rangle$	8, 10
$C^k NOT$	16
$C^k()$	18
$CCNOT$	16, 18
$CNOT$	16, 17
$CPH$	18
$H$	15
$H^{\otimes n}$	32
$ID$	81
$INP$	81
$M$	25
$NOT$	14
$PH$	15
$R_x$	14
$R_y$	14
$R_z$	14, 15
$SWAP$	18
$S$	15
$T$	15
$X$	13
$Y$	13
$Z$	13

## A

alteration	111
amplitude amplification	54, 68
ancilla qubit	24
AND/OR problem	94, 101
approximate counting algorithm	54
autocorrelation	
analysis	138, 139
function	138

## B

Bell states	12
black box	76
black-box	22, 31
Bloch sphere	10, 11, 14
BQP	30
bra	8, 9
building blocks	42

## C

circuit reduction	89
controlled- <i>NOT</i>	16
correlation length	138
crossover	47
disruptive $\sim$	135
landscape	138, 139
linear $\sim$	47, 48, 113
tree $\sim$	47, 48, 113

## D

decoherence	8
decomposition	3, 14, 20, 22, 99

two-level  $\sim$  ..... 21  
 deletion ..... 111  
 density matrix ..... 9  
 density operator ..... 9  
 Deutsch's algorithm ..... 7, 22  
 Deutsch's problem ... 31, 63, 94, 96, 100  
 Deutsch-Jozsa algorithm ..... 35, 54  
 Deutsch-Jozsa problem ... 7, 31, 96, 113  
 discrete Fourier transform ..... 55  
 discrete logarithm ..... 59, 61

**E**

early promise problem ..... 31  
 early termination ..... 99, 118  
 entanglement ..... 12, 16, 25, 31, 35, 36  
     maximum  $\sim$  ..... 94, 106  
 EPR states ..... 12  
 EQP ..... 30  
 error correcting codes ..... 8  
 evolution ..... 12, 40  
     of quantum algorithms ..... 79, 122  
 evolutionary algorithms ..... 41  
 evolutionary strategies ..... 40  
 exon ..... 51

**F**

fitness ..... 40, 41, 49  
 fitness case ..... 39, 49, 113  
     basis state representation .. 113, 123  
     classification ..... 115  
 fitness components ..... 114, 117  
 fitness function ..... 41, 49, 113  
     normalized  $\sim$  ..... 49  
     standardized  $\sim$  ..... 49  
 fitness landscape ..... 136  
     analysis ..... 139  
 Fourier sampling ..... 62

**G**

gate pool ..... 82  
 generational GP ..... 50, 52  
 genetic algorithm ..... 40, 108

genetic operator ..... 41, 47, 111  
 genetic programming .. 2, 39, 93, 99, 100  
     algorithm ..... 42  
     graph GP ..... 46  
     linear GP ..... 44, 45, 80, 110  
     linear-graph GP ..... 46  
     linear-tree GP .. 44, 45, 80, 110, 111  
     program structure ..... 42  
     schema theorem ..... 135  
     stack-based linear GP ..... 100  
     stackless linear GP ..... 100  
     tree GP ..... 42, 43  
 genotype ..... 40  
 Grover operator ..... 67  
     generalized  $\sim$  ..... 68  
 Grover's algorithm ..... 8, 22, 54, 64  
     applications ..... 69

**H**

Hadamard gate ..... 15  
 heredity ..... 40  
 hidden subgroup problem ..... 54, 61  
     Abelian  $\sim$  ..... 62, 63  
 Hilbert space ..... 8, 10, 11, 79, 89  
 Hogg's algorithm .... 54, 71, 74, 95, 123  
 hybrid algorithm ..... 53

**I**

individual ..... 39  
 information analysis ..... 138, 140  
 information content ..... 138, 139  
 inner product ..... 9  
 input gate ..... 81  
 insertion ..... 111  
 instruction ..... 81  
 intron ..... 51  
 irreversibility ..... 7

**K**

ket ..... 8, 9

- 
- L**
- Landauer's principle ..... 7
- M**
- machine learning ..... 39
- majority problem ..... 76, 101
- matrix-matrix multiplication ..... 82–86
- matrix-vector multiplication ..... 82–86
- measurement ..... 9, 11, 12, 24
- intermediate  $\sim$  ..... 27, 92, 110, 121
- partial  $\sim$  ..... 24
- principle of deferred  $\sim$  ..... 28
- projective  $\sim$  ..... 23
- single qubit  $\sim$  ..... 10
- von Neumann  $\sim$  ..... 23
- mutation ..... 40, 47
- landscape ..... 138, 139
- N**
- neutrality ..... 51, 138
- explicit  $\sim$  ..... 51
- implicit  $\sim$  ..... 51
- No-cloning theorem ..... 26
- nonseparability ..... 12
- O**
- observable ..... 24
- ON-basis ..... 9, 12, 24
- oracle ..... 22, 32, 76
- order-finding ..... 58, 59, 61
- P**
- palindrome transformation ..... 22
- palindromic optimization algorithm .. 22
- parity problem ..... 76, 95
- partial information content .... 138, 139
- Pauli matrices ..... 13, 14
- penalty function ..... 114, 117
- period-finding ..... 59, 61
- phase ..... 10, 15
- computing a function into a  $\sim$  .. 33
- gate ..... 15
- global  $\sim$  ..... 10
- phase estimation ..... 56, 58, 68
- phenotype ..... 40
- population ..... 40
- pre-evolved initial  $\sim$  .. 123, 130, 133
- projector ..... 24
- Q**
- quantum algorithm ..... 53
- classification ..... 54
- speedup limits for  $\sim$ s ..... 76
- quantum bit ..... 9, 10
- quantum circuit ..... 7, 13, 25
- analysis ..... 3
- complexity ..... 28
- evaluation ..... 82
- representation ..... 81
- simulation ..... 80
- synthesis ..... 3
- quantum computation ..... 1, 5, 12
- techniques ..... 31
- quantum computer
- simulator ..... 80
- seven qubit (NMR)  $\sim$  ..... 2
- quantum copying ..... 26
- approximate ..... 95
- quantum counting ..... 68
- quantum error ..... 8
- quantum error-correction ..... 8
- quantum Fourier transform ..... 54–56
- algorithms based on  $\sim$  ..... 65
- quantum gate ..... 13, 25
- application ..... 87
- self-inverting  $\sim$  ..... 22
- single qubit  $\sim$  ..... 13
- universal  $\sim$  ..... 18, 20
- unseparable  $\sim$  ..... 16
- quantum Givens operator ..... 21
- quantum interference ..... 31, 34
- quantum mechanics
- postulates of  $\sim$  ..... 8, 11, 12, 24

quantum multiplier . . . . . 95  
 quantum parallelism . . . . . 31, 33  
 quantum phase estimation . . . . . 58  
 quantum register . . . . . 10, 11  
 quantum search . . . . . 94  
 quantum search algorithm . . . . . 64  
 quantum search problem . . . . . 101  
 quantum simulation . . . . . 54, 74  
 quantum state . . . . . 8, 25  
     entangled  $\sim$  . . . . . 12, 25  
     separable  $\sim$  . . . . . 12  
 quantum system . . . . . 8, 10  
     closed  $\sim$  . . . . . 8  
     composite  $\sim$  . . . . . 11  
     evolution . . . . . 12  
     isolated  $\sim$  . . . . . 8  
     physical implementation . . . . . 2  
 quantum teleportation . . . 35, 94, 99, 108  
 quantum Turing machine (QTM) . . 7, 30  
 qubit . . . . . 9, 10, 25  
     control  $\sim$  . . . . . 16  
     counting of  $\sim$ s . . . . . 10  
     target  $\sim$  . . . . . 16  
 query . . . . . 22  
     complexity . . . . . 29

**R**

random walk . . . . . 138  
 readout . . . . . 9  
 recombination . . . . . 40, 47  
 reduction rule . . . . . 89  
 replacement . . . . . 111  
 reproduction . . . . . 40  
 reversibility . . . . . 7, 26  
 ruggedness . . . . . 138

**S**

SAT . . . . . 71, 94, 96  
     constrained  $\sim$  . . . . . 71  
 SBLGP . . . . . 100, 103  
 scalability . . . . . 101, 103, 122

search space . . . . . 92  
 search strategy . . . . . 40  
 second-order encoding . . . . . 101  
 selection . . . . . 40, 49  
      $(\mu + \lambda) \sim$  . . . . . 50  
      $(\mu, \lambda) \sim$  . . . . . 50  
     fitness-proportional selection  $\sim$  . . 50  
     mating  $\sim$  . . . . . 49  
     overproduction  $\sim$  . . . . . 49  
     ranking  $\sim$  . . . . . 50  
     tournament  $\sim$  . . . . . 50  
 self adaptation . . . . . 151  
 Shor's algorithm . . . . . 54, 59  
 Simon's algorithm . . . . . 7  
 Simon's problem . . . . . 64  
 SLLGP . . . . . 100  
 state  
     computational basis  $\sim$  . . . . . 9–11  
     mixed  $\sim$  . . . . . 9  
     pure  $\sim$  . . . . . 8  
     superposition  $\sim$  . . . . . 9, 11  
 state space . . . . . 8, 10, 11, 79  
 state transformation . . . . . 12  
 steady-state GP . . . . . 50, 52  
 subset selection . . . . . 118  
     active data selection . . . . . 118  
     dynamic  $\sim$  . . . . . 118  
     historical  $\sim$  . . . . . 118  
     random  $\sim$  . . . . . 118  
     stochastic sampling . . . . . 118  
     stochastic subset sampling . . . . . 118  
 superposition . . . . . 9, 11, 33  
 superpositioning . . . . . 31, 32  
 supervised learning . . . . . 40  
 swap . . . . . 111

**T**

tensor product . . . . . 11, 12  
 test set . . . . . 40  
 TGP . . . . . 100, 102  
 time series . . . . . 138

Toffoli gate ..... 16, 18  
tournament ..... 50  
tournament size ..... 50  
training set ..... 39, 40, 49, 118

**U**

unitary operator ..... 12–14, 20, 25  
  two-level  $\sim$  ..... 20  
universality ..... 18, 20  
unseparability ..... 16

**V**

variation ..... 40



## Erklärung

Hiermit erkläre ich, dass die in dieser Dissertation vorgestellten Ergebnisse von mir erarbeitet wurden. Die zur Erlangung der Ergebnisse verwendete Software wurde von mir entwickelt.

Herr Prof. Banzhaf zeichnet für das Projekt verantwortlich. Seine Beteiligung am GK 726 ermöglichte mir die Forschung zum Thema "Evolution von Quantenalgorithmen". Als Experte auf dem Gebiet des Genetischen Programmierens floss seine Erfahrung in den nicht anwendungsspezifischen Teil der Arbeit ein. Zahlreiche Diskussionen führten zu Anregungen, aus denen wesentliche Ideen für die Veröffentlichungen entstanden.

Dortmund, im Mai 2004

André Leier