Chapter 1

Genetic Algorithm for Logic Synthesis of Combinatorial Quantum Circuits

1.1 Introduction

A Genetic Algorithm (GA) is an evolutionary model of search and learning based on observed and simplified principles from natural phenomena; the evolution of species in Nature. It was originally proposed by Holland [Hol75] but it was later extended to various alternatives allowing the evolutionary approach to formulate and possibly solve problems from various domains [Hol75, MT78, Gol89, Rai96, LPMP02, Yen05]. The whole research area of evolutionary computation can be separated into three main subareas: Genetic Programming (GP), Evolutionary Strategies (ES) and Genetic Algorithms. The main differences reside in the problem representation (GP - represents the problem as logical trees like structure (Figure 1.1(a)), GA as a string of bits (Figure 1.1(b)), ES - string of real numbers (Figure 1.1(d))), in the employed evolutionary operators (Mutation, Cross-Over, Adaptive Mutation) and finally in the way of generating and selecting solutions.

From the computational point of view, the class of GA's is well suited to solve the constraint-satisfaction problems (CSP) because it offers certain advantages in problems with a high number of local maxima and when the problem definition is noisy. Some instances of the CSP problems that GA was applied to are the Traveling-Salesman Problem, the Eight-Queen problem, the Satisfiability problem, the Map Coloring, etc. Also, quantum logic synthesis is a CSP problem when viewed as a method to synthesize a quantum circuit for a specified function given some structural constraints such as the width of the circuit (number of qubits), the possible types

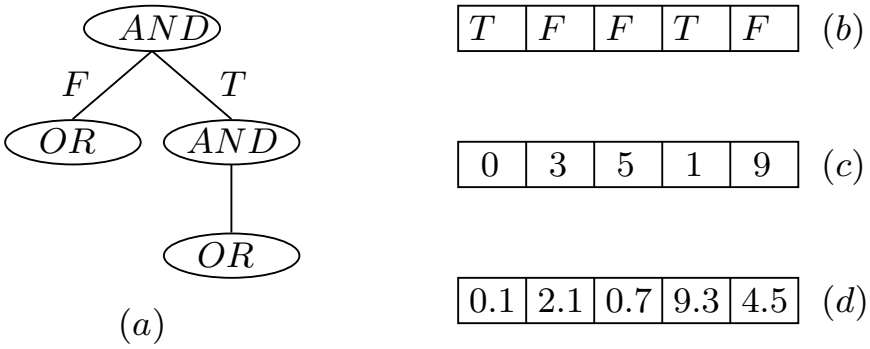


Figure 1.1: solution representation in Evolutionary Algorithms: (a) Logical Tree, (b) Binary String, (c) Integer String, (d) String of Floats

of gates to be used, or the total number of gates to use.

The reasons to use a GA in this book are multiple. The most pertinent are:

- GA is very well situated to explore large problem spaces with small amount of solutions
- GA is well situated to solve problems where only small amount or none information is available about the structure of the problem space
- GA is easily adapted to various rquirements and computational models
- GA allows to synthesize and optimize quantum circuits algorithms, games and automata and is thus a versatile synthesis tool.
- GA concepts can be realized both in a classical and quantum computer [], thus leading to a new concept of Quantum Evolutionary Hardware.

Thus the GA used in this work is merely a tool to obtain examples or results, explore the possibilities of QLS and demonstrate the introduced principles.

The approach adopted by the evolutionary computation represents a metaphore to the evolutionary process in Nature described at best by a natural selection, sexual reproduction and random mutation seen as a process of computation.

First, these concepts mean that a GA uses chromosomes (strings of elements) to represent the problem; each chromosome can be a possible solution to the problem. For a population of such *individuals*, a large problem space can be covered by the evolutionary search as well as multiple local minima can be explored. Second, a population of individuals together with the fitness function represents the information about the problem that is available to the evolutionary algorithm. Third, the computation is represented by a computational cycle that consists of: random mutations

3	2	1	7	5	8	4	6	9	0

Figure 1.2: Example of an individual in a GA solving the TSP problem. the numbers correspond to towns in order from left to right. Each individual is a permutation on the set of all possible towns to visit

(introducing noise and novelty into the system - randomly altering the individuals), information exchange between individuals (local solutions) represented in the cross-over operation and a simulated *survival-of-the-fittest* mechanism of selection and replication. These are the only operations allowed on the set of individuals. Fourth, the method of evaluation of each individual contains all of the knowledge required to determine whether or not a solution was found.

This brief description can be illustrated by the following points describing the process of designing a GA for a given problem.

- Assuming the problem is formalized as a CSP, select the appropriate information encoding. For instance, a n-vertices Traveling Salesman Problem (TSP) can be encoded as strings of length n, representing by an integer number each vertex from the graph. This can be seen in Figure 1.2
- Select appropriate evolutionary operators, in general the standard mutation modified for the problem is enough to introduce noise, however self-adaptive mutation operator is also a well known tool in ES approach. In particular for the TSP problem one must preserve the property of chromosomes such that each chromosome represents a valid path in a graph (each vertex must be present in the path and it can appear only once). Thus the mutation operation can be a 2-vertices random position swap.
- Select a cross-over operator such as single-point cross-over, two-points cross-over or multi-parent crossover [EvKK95]. For certain problems, when the chromosome is ordered, the crossover must be able to preserve the overall validity of the individual similarly as in the case of mutation. Again, in the case of TSP, Figure 1.3 explains the crossover.
- Select or create a fitness function - the function that will evaluate each individual solution and generate fitness value for each solution. This means that for the TSP problem, such a function evaluates the total path length that each individual's chromosome encodes for.
- Select a selection scheme such as Roulette Wheel (RW) or Stochastic Universal Sampling (SUS). This process simulates the *survival-of-the-fittest* rule, as it is

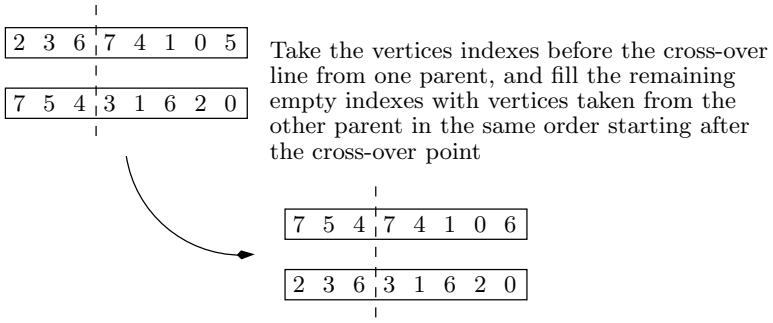


Figure 1.3: Example of cross-over operation for the TSP problem.

biased to select preferably chromosomes/individuals with a higher value of fitness. In this step it is also possible to tune the selection process to various forms of elitism (These concepts are explained in details below).

In this chapter a GA is presented and its mechanisms are explained in details. The various functions and objects are specified with respect to the problem of quantum logic synthesis (synthesis of quantum circuits). More general ideas are also introduced below in order to cover the potentials of the evolutionary computation in quantum circuit (automata, games, algorithms, etc) design. The description also extends to the mechanism of the combined approach called the GAEX genetic algorithm for quantum circuit synthesis [LPG+03, LP02, LP05b].

1.2 Genetic algorithm

A Genetic algorithm is a set of directed random processes that make probabilistic decisions - simulated evolution. Table 1.1 shows the general structure of a GA algorithm and this section follows this structure with each step explained in individual sub-section.

1.2.1 Encoding/Representation

For quantum logic synthesis the representation that we use is based on the encoding introduced in [LPMP02]. This representation allows to describe any Quantum or Reversible circuit [LPG+03, LP02]. All individuals in the GA are strings of ordered characters (each character representing a quantum gate) partitioned into parallel blocks. This partitioning of the circuit was in our case induced from the representation of any QC such as one in Figure 1.4. Each block has as many inputs

Table 1.1: Structure of a Genetic Algorithm

```
01: t ← 0;
02: initialize(P(t));                          /* initial population */
03: evaluate(P(t));                            /* evaluate fitness */
04: while (not termination-condition) do
05: t ← t + 1;
06: Q_s(t) ← select(P(t ← 1));                 /* selection operator */
07: Q_r(t) ← recombine(Qs(t));                 /* crossover operator */
08: P(t) ← mutate(Q_r(t));                     /* mutation operator */
09: evaluate(P(t));                            /* evaluate fitness */
10: end while
```

and outputs as the width of the quantum array (five in the case of Figure 1.4). The chromosome of each individual is a string of characters with two types of tags. First a group of characters is used to represent the set of possible gates that can be used in the individual string representation. Second, a single character 'p' is used as a separator between parallel blocks of quantum gates. An example of a chromosome can be seen in Figure 1.4. In this particular encoding each space (empty wire or a gate) is represented by a character with appropriate decoding shown. Our problem-specific encoding was applied to allow the construction of as simple genetic operators as possible. The advantage of these strings is that they allow encoding of an arbitrary QL or RL circuit without any additional parameters. Several such parameters were used in previous research [LPG+03, LP05a] and using them made the genetic algorithm more complicated. Please note that only the possibility to move gate characters, remove and add them to the chromosome consequently make it possible to construct an arbitrary circuit and also to modify this circuit in order to optimize it.

1.2.2 Initialization steps of GA

The GA requires an input file (c.f. Pseudo-Code 1.1 and Pseudo-Code 1.2) which specifies all input parameters and required settings.

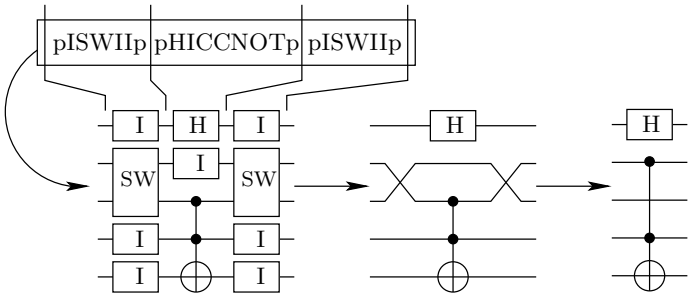


Figure 1.4: Transformation of a QC from the chromosome (on the top) encoded string, to a final quantum circuit notation representation of this QC (on the right). Here SW is a Swap gate, H is a Hadamard gate and I is a Identity. In the middle there is one $CCNOT$ (Toffoli) gate.

(1.1)

01 :Population Size:80
02 :Segment MaxNumber:20
03 :Segment MinNumber:2
04 :Local Search generational switch:100000
05 :Mutation Probability:10
06 :Crossover Probability:85
07 :Alpha:0.99
08 :Beta:0.01
09 :Cost divider:10
10 :Type of GA (normal=0; Baldwinian=1) : 0
11 :Type of Mutation(normal=0; bitwise=1) : 1
12 :Type of Crossover(1-point = 0;2-point=1) : 0
13 :Replication($0-$ Roulette Wheel, $1-$ Stochastic Universal Sampling, $2-$ Tournament) : 1
14 :Type of Fitness($0-$ simple linear only error, $1-$ simple square only error, $2-$ complex linear, 3 - complex) :0
15 :Type of fitness calculation (0 - individual, 1 - grouped) :0
16 :Pareto optimization (0 - no, 1 - yes) :0
17 :Threshold replication (0 - no, other - threshold) :0
18 :Elitism (0-disabled, 1-enabled) :0
19 :Tournament parameter(amount of individuals chosen randomly) :5
20 :Measurement :1
21 :Measured qubits indexes:0
22 :$(1,0)(0,1)(1,0)(0,0)(1,0)(0,0)(0,0)(0,0)$
23 :$(0,0)(0,1)(0,0)(1,0)(0,0)(1,0)(1,0)(1,0)$

The lines (01-23) within the file specifies the parameters defining the overall behavior of the GA. The number of individuals in the population is given first in line 01. The size (length) of the circuit specified by a maximal (t_{max}) and minimal number of segments (t_{min}) in each individual (chromosome) is given in lines 02 and 03 respectively.

The initial circuits are created with a random size within the interval specified by these two parameters. The size of the chromosome is not limited during the lifetime of an individual. Rather, two other parameters allow to specify the minimal and maximal sizes of every circuit. Each individual has a dynamically changing genome and the GA is a subclass of the Messy GA [GKD89].

Line 04 specifies the number of generations of the evolutionary search after which the GA will switch to a local search. Lines 05-06 specify the probability of mutation and crossover, and lines 07 and 08 specify the parameters α and β when fitness function from eq. 1.16 and 1.17 is used. Line 09 represents the minimal cost $MinCost$[1] used in the cost function described in section 1.4.2.

Line 10 specifies the type of GA that is run; two possibilities are available: a standard GA and a Baldwinian model, line 11 specifies whether bitwise mutation or standard mutation is used, line 12 determines if single-point or two-point crossover is used and line 13 allows to select the replication mechanism: either the Stochastic Universal Sampling (SUS), the Roulette Wheel (RW) or the Tournament Selection can be used. Line 14 specifies the type of the fitness function (Section 1.4) and the line 15 allows to share the fitness among individuals (fitness scaling). Line 16 allows to turn the GA into a Pareto-optimizing (multi-objective) evolutionary search and line 17 allows to force the GA to limit the selection and replication process by a threshold. Line 18 allows to use the Elitism and line 19 allows to choose the number of individuals used in the tournament replication procedure. Finally lines 20 to 23 specify if the measurement is used, how many qubits are measured as well as the expectation values of the measurement.

[1] In this book this parameter will be referred to as either *minimal* cost, *desired* cost or *optimal* cost during the process of synthesis

$$
\begin{array}{l}
23:21 \\
24:1 \\
25:1 \\
26:\text{wire} \\
27:(1,0)(0,0) \\
28:(0,0)(1,0) \\
\quad\vdots \\
44:3 \\
45:1 \\
46:\text{Controlled\_wire\_V} \\
46:(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0) \\
47:(0,0)(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0) \\
48:(0,0)(0,0)(1,0)(0,0)(0,0)(0,0)(0,0)(0,0) \\
49:(0,0)(0,0)(0,0)(1,0)(0,0)(0,0)(0,0)(0,0) \\
50:(0,0)(0,0)(0,0)(0,0)(0.5,0.5)(0.5,-0.5)(0,0)(0,0) \\
51:(0,0)(0,0)(0,0)(0,0)(0.5,-0.5)(0.5,0.5)(0,0)(0,0) \\
52:(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0.5,0.5)(0.5,-0.5) \\
53:(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0.5,-0.5)(0.5,0.5)
\end{array}
\tag{1.2}
$$

The input file also lists the elementary quantum gates to be used as components, like the single qubit H, X, Y, Z or V gates and two qubit operations such as $CNOT$ or CV, which are the building blocks of the quantum circuits to be found. The quantum gates are represented as quantum unitary (and Hermitian) matrices with the cost specified for each gate.

On line 23 the total number of component input gates is given. Then the unitary matrices of the gates, their number of input/output and their cost are given. From lines 23 to 27 the truth table of the single qubit operation "Wire" is presented. Another quantum gate is depicted as a quantum truth table (in a form of a unitary matrix) from lines 44 to 53. This gate is a 3*3 Controlled-V (CV) gate [BBC$^+$95] on qubits 0 and 2. Observe that each input gate is specified by a unitary matrix that describes each complex coefficient by the real and imaginary components. For instance $(1,0)$ represents the real state, while $(0.5,0.5)$ represents a complex state with coefficient $\frac{1+i}{2}$.

The above described features of the GA are described later on in this chapter and in the next chapter.

1.3 Evaluation of Synthesis Errors

The GA used in this book has two possible evaluation methods for designed circuits that have been developed in order to accommodate both completely and incompletely specified quantum-reversible functions. These methods are called ME and EE.

1.3.1 Element Error Evaluation method (EE)

To represent a completely specified functions (in particular, the deterministic permutative reversible functions such as the universal gates Fredkin or Toffoli) a matrix is used. It can be seen that as long as the function is easily specified in this manner, it is possible to evaluate the Unitary matrix of the synthesized circuit directly. This matrix represents the desired matrix that the circuit must satisfy by comparison of each matrix coefficient. For instance, to specify the Fredkin gate, the number of the qubits of the result and the matrix specifying the target circuit can be added at the end of the input file (eq. 1.3). Observe that each coefficient in the matrix is represented by a pair of floats. For instance (1,0) represents the number 1 + 0i and (0,0) represents 0 + 0i.

$$
\begin{aligned}
&54 : 3\\
&55 : (1,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)\\
&56 : (0,0)(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)\\
&57 : (0,0)(0,0)(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)\\
&58 : (0,0)(0,0)(0,0)(1,0)(0,0)(0,0)(0,0)(0,0)\\
&59 : (0,0)(0,0)(0,0)(0,0)(1,0)(0,0)(0,0)(0,0)\\
&60 : (0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(1,0)(0,0)\\
&61 : (0,0)(0,0)(0,0)(0,0)(0,0)(1,0)(0,0)(0,0)\\
&62 : (0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(1,0)
\end{aligned}
\tag{1.3}
$$

This method is computationally intensive. However, it allows to directly alter and analyze the circuit matrix representation; changes on individual gates can be directly observed on the unitary matrix. The evaluation of the error is done in this case by formula $e_k = \sum_j (o_j - s_j)^2$ (i.e. by comparing the resulting and the desired matrices coefficient-by-coefficient) in order to obtain the overall error (before measurement). This evaluation method is referred to as the Elements Error (EE) method.

1.3.2 Measurement Evaluation method (ME)

When the desired circuit is to represent an incompletely specified function represented as $f = [00, --, --, 10]$ or as $f = [0-, -1, --, 0-]$ (Definition **??**), the matrix representation is not convenient and designing a unitary quantum-realizable incompletely specified function might not even be possible for large functions. Also, representing such circuit as a unitary matrix would require to specify all elements either as cares or as don't cares. In general, the number of don't cares for machine learning will be much higher than the number of cares. Thus representing machine learning problems as matrices would be wasteful.

Thus, in cases when the function is incompletely specified (or the output function is defined on less bits than the input), it is better to represent the solution only by the obtainable information. This is done by using the after-measurement evaluation of the circuit. The GA generates the set of measurement operators for each desired qubit. For each output state of the circuit under evaluation the algorithm measures the state for the desired imulti-qubit measured state (Section **??**). In the case of a don't care the GA skips the given measurement and the value of the output remains unknown. This method is referred to as the Measurement Evaluation (ME) method.

1.3.2.1 Measurement Evaluation Input-Data Specification

The specification of the problem for ME method is shown in lines 19 to 22. In this case, there is one qubit that is going to be measured (line 19) on the first wire (indexed as 0: line 20). The measurement expectation values are in lines 21 for the expected state $|0\rangle$ and line 22 for the expected state $|1\rangle$. Note that the measurement for state $|001\rangle$ (line 21 and line 22) has expected complex value $0 + i1$ represented as $(0, 1)$ for both possible outcomes $|0\rangle$ and $|1\rangle$. This artificial notation means that the outcome is considered as a don't care. The error evaluation becomes $e_k = \sum_j (o_j - s_j), \forall j \in O$, with O the set of all defined expected values.

For example, assume the output state from the circuit under evaluation is $|\psi\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}$ and the expected result is the state $|00\rangle$. The error with respect to the measured qubit (assume indexed at 0) will now be $e_0 = 0$ and for the second qubit $e_1 = 0.5$. This means that if the output is taking into account only the zero-th qubit, the state $|\psi\rangle$ is a valid solution. However, if also the second qubit is used, the state $|\psi\rangle$ is not a solution, because the expectation value for the output state $|00\rangle$ is 0.5. In general, the error is summed over all measured qubits and normalized in order to be directly used in the calculation of the fitness function. The

Table 1.2: Example of the Majority gate encoding for the GA. Observe that each input minterm with qubit $|q_3\rangle = |1\rangle$ is a don't care.

$q_0q_1q_2q_3$		m_0	m_1	$q_0q_1q_2q_3$		m_0	m_1
0000	$\xrightarrow{M}$	(1,0)	(0,0)	**1000**	$\xrightarrow{M}$	(1,0)	(0,0)
0001	$\xrightarrow{M}$	(0,1)	(0,1)	1001	$\xrightarrow{M}$	(0,1)	(0,1)
0010	$\xrightarrow{M}$	(1,0)	(0,0)	**1010**	$\xrightarrow{M}$	(0,0)	(1,0)
0011	$\xrightarrow{M}$	(0,1)	(0,1)	1011	$\xrightarrow{M}$	(0,1)	(0,1)
0100	$\xrightarrow{M}$	(1,0)	(0,0)	**1100**	$\xrightarrow{M}$	(0,0)	(1,0)
0101	$\xrightarrow{M}$	(0,1)	(0,1)	1101	$\xrightarrow{M}$	(0,1)	(0,1)
0110	$\xrightarrow{M}$	(0,0)	(1,0)	**1110**	$\xrightarrow{M}$	(0,0)	(1,0)
0111	$\xrightarrow{M}$	(0,1)	(0,1)	1111	$\xrightarrow{M}$	(0,1)	(0,1)

particular function, shown in lines (21-23) (eq. 1.1), is the majority (3x1) function with the measured output on the 0-th qubit. To synthesize circuits using an ancilla bit constant such as in [HSY+06], the input is modified to represent the whole information after measurement with using don't cares for all values of the ancilla bit/constant that are not required. For example to represent the majority function as a three-input and single-output function (using a dedicated output qubit q_0 set to $|0\rangle$), the measurement is specified in Table 1.2. This table corresponds directly to relational specifications with inputs q_0, q_1, q_2 and q_3 from the function from the K-map in Figure 1.5a.

The topmost line shows the input state $q_3q_2q_1q_0$ in the first column, columns labeled m_0 and m_1 represent the encoded probability of observing the state $|0\rangle\langle 0|$ and $|1\rangle\langle 1|$ respectively (single qubit output specified by a pair of numbers $(1,0)(0,0)$ representing the complex coefficients for the state $|0\rangle$ and $|1\rangle$). The $\xrightarrow{M}$ operation represents the whole state being measured using projective measurement on orthonormal bases $|0\rangle$ and $|1\rangle$. The don't care is represented as a complex expectation value $(0,1)$ $(0,1)$. The corresponding standard K-map with inputs q_0, q_1, q_2 and output q_3 is given in Figure 1.5b.

1.3.3 Comparison of EE and ME

The error for each individual is calculated by comparing each user-specified output care with the value obtained from the simulation of the quantum circuit behavior.

If the ME model is used, the desired outputs are specified as the state of the system after the measurement. The desired output is defined by the state that has the

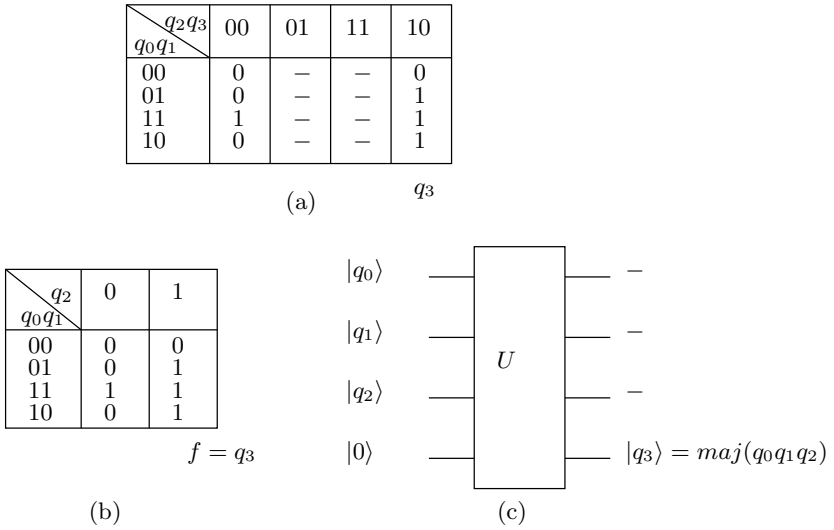


Figure 1.5: Majority gate function: (a) the relational K-map, (b) the standard K-map withou output q_3, (c) the schematic circuit representation.

highest probability of observation for a given input state. For permutative circuit this means that for each input state the GA will be evaluating the probability of observing two states: the desired and the undesired state. For instance, if the desired output state is $|001\rangle$, the GA will also evaluate the probability of observing the most undesirable state to $|001\rangle$ which is $|110\rangle$.

The state $|110\rangle$ is called the undesired state and it represents the bit-by-bit negation of the desired state. This means that a given function to be synthesized as a circuit is specified by the user using a set of desired states. The undesired states are used to provide additional information during the evaluation and the evolutionary search for the solution circuit.

Each desired output is calculated for each measured value individually. For a two valued output, the error with respect to the state after measurement of 0 and 1, can be written as $\frac{(o_k(0)-p)^2+(o_k(1)-(1-p)^2)}{2}$, with p being the desired probability of obtaining a 0 and $q = 1-p$ being the probability of obtaining a 1. For a complete set of inputs the overall error for a given individual is given by

$$\begin{aligned} error = \sum_k e_k &= \frac{1}{k}\sum_{k=1}^{2^n-1}\sum_{j=0}^{m-1}(o_k - \langle\psi'_k|M_j^*M_j|\psi'_k\rangle)^2 \\ &= \frac{1}{k}\sum_{k=1}^{2^n-1}\sum_{j=0}^{m-1}(o_k - p_k)^2 \end{aligned} \tag{1.4}$$

with $|\psi'_k\rangle = U|\psi_k\rangle$, m being the number of possible outcomes of the measurement,

$\langle\psi|M_0^* M_0|\psi\rangle = p$ and $\langle\psi|M_1^* M_1|\psi\rangle = 1 - p$.

For a $\frac{1}{2}$-spin quantum system (Boolean observable), $j = 2$, the equation 1.4 can be rewritten as

$$error = \frac{1}{k}\sum_{k} \frac{(o_k(0) - p_k)^2 + (o_k(1) - (1 - p_k)^2)}{2} \tag{1.5}$$

Thus for an incompletely specified permutative function defined on three qubits, the measurement is designed as $M_J = m_{0|1}^{\otimes} n$ and represents the Kronecker product of single qubit measurements on j qubits. Each single qubit measurement is selected according to the preference given by the GA software user. For instance the desired two-qubit measured state given as $(1,0)(0,0)(0,0)(1,0)$ corresponds to a measured state $|01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$. To obtain this state, the measurement operators from eq. 1.6 will be used during the evaluation process.

$$M_2 = m_1^1 \otimes m_0^0 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{1.6}$$

A further constraint on the ME evaluation method is that for each function the number of measured qubits must remain constant. This requirement insures that each measurement result has the same weight in the overall error (error is directly proportional to the number of measured qubits).

Observe that using the ME error is much less precise than using the EE error. For a three qubit function, the matrix based method compares all 64 coefficients of the matrix while the ME evaluates only 16 output states. The implication is that the error obtained in ME has less information about the system.

When the desired state is deterministic (reversible permutative function), the correct output state can be obtained exactly up to the phase of the unmeasured state. In the case, when the desired single-qubit state observation probability is not deterministic, both states are detected by the single-qubit measurement operator. For instance eq 1.7 shows how to interpret the encoding of a single qubit superposed quantum state.

$$
\begin{aligned}
(0.5,0)(0.5,0)(0,0)(1,0)(0,0)(1,0) =& \frac{1}{\sqrt{2}}(|011\rangle \pm |111\rangle) \\
=& \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)|11\rangle
\end{aligned} \tag{1.7}
$$

The first qubit's is described by $(\frac{1}{2},0)(\frac{1}{2},0)$ which means that it is to be observed with a probability of $\frac{1}{2}$ in the $|0\rangle$ state and with the probability $\frac{1}{2}$ in the $|1\rangle$ state. The second and the third qubits are described by $(0,0)(1,0)$ which means that both qubits are to be in the $|1\rangle$ state with probability 1 after the measurement process. Thus the state can be factored with respect to the first qubit as shown in eq 1.7.

For multi qubit measurement the probability of a desired output state is the product of probabilities of observation for all individual qubits; various multi-qubit states observed under two measurements of maximally opposed observables can be indistinguishable.

The GA always builds only two measurement operators. From the external observer point of view, this measurement process looses information proportionally to the number of simultaneously measured qubits. For instance the desired states in eq. 1.8 and 1.9 are not distinguishable under this incomplete measurement.

$$
\begin{aligned}
(\frac{1}{3},0)(\frac{2}{3},0)(\frac{1}{10},0)(\frac{9}{10},0)(0,0)(1,0) =& \frac{1}{30}|001\rangle + \frac{3}{5}|111\rangle \\
&+ \frac{3}{10}|011\rangle + \frac{1}{15}|101\rangle
\end{aligned} \tag{1.8}
$$

$$
\begin{aligned}
(\frac{1}{10},0)(\frac{9}{10},0)(\frac{1}{3},0)(\frac{2}{3},0)(0,0)(1,0) =& \frac{1}{30}|001\rangle + \frac{3}{5}|111\rangle \\
&+ \frac{1}{15}|011\rangle + \frac{3}{10}|101\rangle
\end{aligned} \tag{1.9}
$$

Observe that the desired state is $|111\rangle$ (with observation probability $\frac{2}{3} * \frac{9}{10} = \frac{3}{5}$) and the undesired state is $|001\rangle$ (with observation probability $\frac{1}{10} * \frac{1}{3} = \frac{1}{30}$) have similar probabilities of observation in both cases (eq. 1.8 and 1.9) are same despite the single qubit probabilities are different. Also observe that when specifying the a single desired and singel undesired output observable state (as in eq. 1.8 and 1.9) the unspecified terms (in this case $|100\rangle$ and $|010\rangle$) represent the lost information because during the evaluation process these states are not being probed for. Thus the encoded state in eq. 1.9 is not a complete quantum state. The described encoding is sufficient, however, to encode problems presented in this book.

The measurement of multi-qubit entangled states is also possible by specifying equal probabilities of observation for each possible output state (eq. 1.10). The complete

state given by the specification in eq. 1.10 is shown in the right side of the same equation. However, the desired and the undesired states are generated in such manner by the GA algorithm that allows to select two out of all equiprobable states. Th8us in this case the desired and undesired states are the states $|001\rangle$ and $|111\rangle$.

$$(1.10) \qquad (\frac{1}{2},0)(\frac{1}{2},0)(\frac{1}{2},0)(\frac{1}{2},0)(0,0)(1,0) = \frac{1}{4}|001\rangle + \frac{1}{4}|111\rangle + \frac{1}{4}|011\rangle + \frac{1}{4}|101\rangle$$

The purpose of using two different evaluation methodologies was to allow more control over the evolutionary search. In general the measurement evaluation is used for the exploration; searching for novel functions or for incompletely specified function synthesis. The EE approach is more useful to describe completely specified permutative functions for the cost optimization and for the circuit size optimization.

Observe that because the EE method is a difference of squares between the outputs of the target and the synthesized quantum circuit, the obtained unitary matrix of a circuit can differ from the target circuit by complex phase. This is illustrated in section 2.4.3. Also, because when using the ME method the GA generates two measurements for each possible output, the algorithm can search for the target circuit or for the negation of the target circuit. The synthesis of negated circuits is interesting because the desired circuit can be easily generated from such circuit simply by negating all qubits. Synthesis of a negated circuit is illustrated in section 2.3.2.

1.4 Fitness functions of the GA

The error defining the correctness of a (potential) solution is used in the fitness function to shape the landscape of the solutions in order to find the global optimum. The fitness function quantifies how good the individuals (candidate solutions) are. As already mentioned, the fitness function is the mechanism allowing to determine the correctness of a given individual. In order for the fitness function f to correctly approximate the problem space:

- $\forall c_i \in G, \exists f(c_i)$, i.e. f must be differentiable
- $f(c_i) \geq 0$, i.e. f must exists for every individual problem representation.

The fitness function evaluates, during each generation, all the individuals of a population and it determines which individuals are more likely to "survive" and which will be possibly discarded. The fitness value of each individual should represent how close the individual is to the optimal solution represented by the fitness value 1.

The evaluation of usefulness of each individual is based on the value of its fitness function. The method requires sometimes various degrees of adjustment of the fitness value. One of such parameterizations used in adjustment is the penalty function. The penalty function represents a negative component (lowering the fitness value of individuals) in cases where an individual c_i gets outside of the validity of its phenotype - the individual hits the boundary constraints. Thus the general form of an evaluation function $e(c)$ is shown in eq. 1.11.

$$e(c) = f(c) + p(c) \tag{1.11}$$

where $f(c)$ is the fitness function and $p(c)$ is the penalty function. A simple penalty function is equal to zero for valid individuals ($p(S) = 0, \forall S \in F$) and strongly non zero for invalid individuals. There are several variants of penalty functions e.g. Death Penalty, Static Penalties, Dynamic Penalties, Annealing Penalties, Adaptive Penalties, Segregated GA and Co-evolutionary Penalties [Yen05].

1.4.1 Simple Fitness Functions

Four different fitness functions f_1, f_2, f_3 and f_4 are implemented and can be chosen by declaring them in the input file.

$$f_1 = 1 - \frac{Error}{E_m} = 1 - e \tag{1.12}$$

The first fitness function, eq. 1.12 is the simplest and it represents the fitness that is inversely dependent on the overall error. The maximal error E_m is calculated as

$$E_m = 2^{2^n} \tag{1.13}$$

with n being the number of wires/qubits. This error can be normalized using the equation 1.5.

The second fitness function is described in equation 1.14.

$$f_2 = \frac{1}{Error + 1} \tag{1.14}$$

The fitness function f_2 preserves a small probability for the less fit individuals due to its exponential character. Using f_2 allows individuals with very small fitness to be selected during replication with a higher probability than using fitness f_1. These two fitness functions are graphically represented by the bold lines in Figure 1.6. The overall error is calculated for an example of a 3-qubit circuit (the maximal error is $2^{2^3} = 64$).

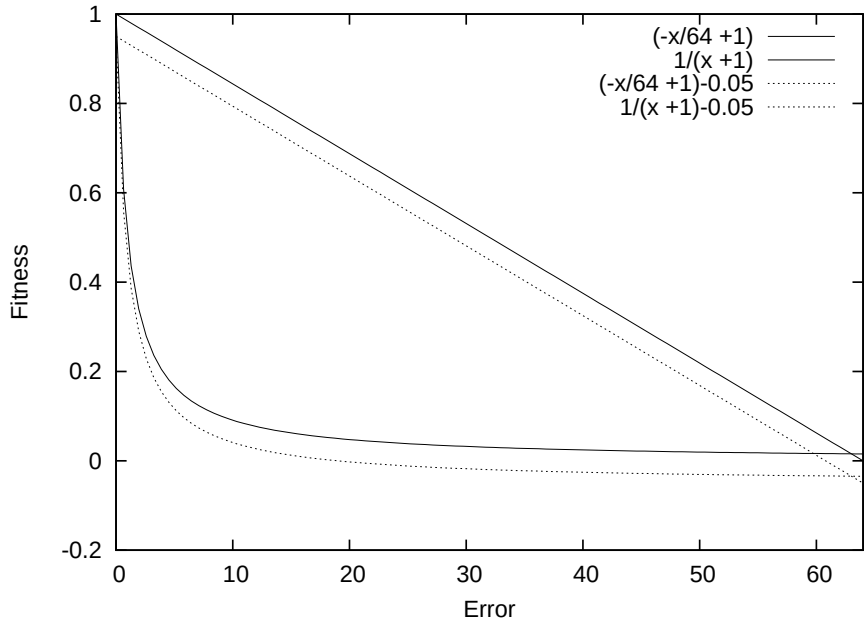


Figure 1.6: Schematic representation of the four available fitness functions. The solid lines represent respectively the fitness functions f_1 and f_2 and the dashed lines represent the fitness functions f_3 and f_4.

1.4.2 Cost Based Fitness Functions

The cost function is based on a parameter known as the *minimum cost* that is provided by the user and that permits to estimate a normalization constant. This means that the cost function acts as a bonus inversely proportional to the size of the circuit to the fitness function for a given estimated and unreachable minimum. In this book the *cost function* is defined by

$$G(c) = exp^{-\frac{(MinCost-Cost)^2}{\frac{Cost}{2}^2}} \tag{1.15}$$

where $Mincost$ is the parameter given by the user (line 09 eq. 1.1) and $Cost$, given by $\sum_{j=1}^{k} c_j$, is the sum of costs of all gates in the evolved circuit. Equation 1.15 was experimentally determined to be sensitive enough to influence circuits that are both far and close to the optimal cost.

The two weighted fitness functions (eqs. 1.16 and 1.17) calculate the fitness value using the fitness function and the cost function together. In this case, the fitness is dependent on the error of the gate with respect to the truth table of the specification as well as on the cost of the circuit. Each component of these weighted functions

can be adjusted by the values of parameters Alpha (α) and Beta (β) (line 6+7 in pseudo-code 1.1). The two weighted fitness functions are given in equations 1.16 and 1.17, respectively.

$$f_3 = \alpha\,(1 - e) + \beta G(c) \tag{1.16}$$

$$f_4 = \alpha\left(\frac{1}{Error + 1}\right) + \beta G(c) \tag{1.17}$$

In Figure 1.6 the dotted lines which represent the weighted fitness functions are shown assuming that the cost function is constant for all the error values. The second term from eqs. 1.16 and 1.17 - the cost - was set to -0.05 to represent the worst case for settings $\alpha = 0.95$ and $\beta = 0.05$. This corresponds to a circuit that is so long that for the user-defined minimum its cost is very large, thus its fitness is penalized by $f(s) = 0.95 * E + 0.05 * 0 = E - 0.05$.

In other words, in the two weighted fitness functions from eqs. 1.16 and 1.17, the term $\beta * Cost$ lowers the fitness functions f_3 and f_4 by a constant number in comparison with the fitness functions f_1 and f_2. This formulation of a weighted fitness function allows the selection process to explore solutions not only related to the correctness of the circuit itself but also to explore the problem space in a different cost-based neighborhood of solutions. This is because each individual would be affected in a similar manner and the overall length of the individual strings allows the GA to overcome local minima such as $f_{\alpha_i,\beta_i}(s_k) \geqslant f_{\alpha_j,\beta_j}(s_l), e_k < e_l$. On one hand the α and β coefficients allow individuals with lower circuit cost and higher error to reproduce and on the other hand they allow to progressively adjust the overall cost of the pool of individuals.

The reasons for these various fitness functions are the following:

- to allow different selection pressures during the individual selection process this will be discussed in Section 1.5,
- by calibrating the cost to always underestimate the minimal possible size of the desired circuit allows the user to further manipulate the selection process.

For instance the fitness function f_3 is not equal to one, unless both the cost of the circuit and the error are minimal. Thus a GA using such a weighted function has more freedom for searching a solution, because the fitness function is now optimizing the circuit for two parameters. Similarly in the case of the fitness function f_4 this type of fitness function lowers the fitness function values of longer circuits, therefore preferring the shorter ones. Thus individuals with different circuit properties will

have equal fitness value. Let two individuals s_0 and s_1 have the fitness calculated according to eq. 1.18 and eq. 1.19.

$$f(s_0) = \alpha E_0 + \beta C_0 \tag{1.18}$$

$$f(s_1) = \alpha E_1 + \beta C_1 \tag{1.19}$$

Then it is obvious that if $f(s_0) = f(s_1)$ for $E_0 < E_1$ and $C_0 > C_1$ then for $\Delta C = C_0 - C_1, \Delta E = E_1 - E_0$ we can write $E_0 = E_1 - \Delta C$. This means that two circuits, one with larger error but shorter in size than the other one will have more chances to get selected for replication. The method thus preserves the diversity of the population to a larger extent. As will be seen later in Chapter 2, this property is an important requirement for successfully synthesizing larger quantum circuits.

The general expression for the cost of a circuit is given by eq. 1.20,

$$C_{C_i} = \frac{C_{Min}}{Cost_i} \tag{1.20}$$

where $Cost_i \geq C_{Min} > 0$ and $Cost_i$ is the cost of the given solution (circuit) calculated from the individual compoenent gates cost used in the circuit and C_{C_i} is the cost of the circuit calculated with respect to a underestimated cost of the ideal circuit given by C_{MIN}. This function requires that the minimum (C_{Min}) value given by the user is not realizable for the given circuit with cost $Cost_i$.

In the case where the estimated minimum C_{MIN} is not well known (a smaller circuit implementing desired function might exist), an exponential Cost function can be used, as in eq. 1.21.

$$C_{c_i} = e^{-|(C_{Min} - Cost_i)|} \tag{1.21}$$

This exponential cost function is less sensitive to variations in size of larger circuits and at the same time the distinct cost advantage to the smaller circuits. Thus this cost function is good for hard problems where a premature convergence occurs often.

1.5 The Selection Process

The process of selection chooses the best individuals for reproduction. Two (or more) parent individuals from the current population are chosen for the reproduction. The

selection process simulates the principle of the natural selection by preferring the more fit individuals to the less fit ones. Thus individuals with higher value of fitness are selected more often (with a higher probability) than those with lower fitness values. With each individual being a potential solution or carrying a piece of its genotype required to find the solution, an appropriate selection method of individuals must be applied to find the problem solution. This means that for a successful search one requires that the selection pressure is low at the beginning of the search and the selection pressure becomes high towards the end of the search. The idea behind the *selection pressure* is the following: from an initial random population of individuals, the solution will be found if the selection process preserves enough of variety in the genetic pool of the population to allow overcome the local fitness maxima. This means that if the selection picks only individuals with the highest fitness (*the high selection pressure case*), the global solution might not be found because the individuals do not have enough genetic material to generate the solution. On the other hand, if the selection is too relaxed (*the low selection pressure case*) the search process will take too long and might not converge at all. The selection of individuals will become independent of their fitness values which reduce the evolutionary process to a random search.

In general, the initial population generation is very important to the success of the evolutionary computation. This is because during a GA computation the information that is contained in the population of the individuals decreases with every generation as the main computational operation is the recombination or crossover of individuals. The replication and crossover do not bring any new information into the genetic pool. The mutation operator does bring external (random) changes in the population, but in general it is used only to perturb the system, rather than insert large amounts of random data. A mutation process inserting too much random elements will again reduce the evolutionary process to a random search. In this case the selection process will pick individuals proportionally to their fitness value, but when each generation is highly modified by the mutation operator does not allow any predictable convergence to the global optimum solution is not possible.

This GA uses the fitness proportional selection (eq. 1.22). This approach is mainly known from the Roulette Wheel and the Stochastic Universal Sampling methods. The Roulette Wheel is the simplest of the selection methods, it allows to randomly select two individuals with probabilities $p(c_i)$ of selection proportional to the fitness values of these individuals. This is shown in equation 1.22.

$$p(c_i) = \frac{f(c_i)}{\sum_{j=1}^{n} f(c_j)} \text{ with } \sum_{j=1}^{n} f(c_i) > 0 \tag{1.22}$$

$p(c_i)$ is the probability with which an individual c_i is chosen in the selection process

Figure 1.7: The fitness normalization represented on a line (a roulette wheel). Each individual has a segment allocated proportional to its fitness

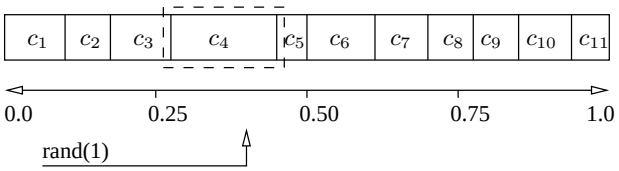


Figure 1.8: The Roulette Wheel Selection - a random number generator picks the individuals one by one located on the equi-distant points over the individuals fitness-sum representation. Here, $p \approx 0.45$ selects individual c_4.

with n being the population size. Every individual is represented by a section of a circle proportional to its $p(c_i)$ (eq. 1.23). This proportional scaling is shown in Figure 1.7. Thus the whole population can be represented as a roulette wheel with sections proportional to $p(c_i)$.

$$\sum_{i=1}^{n} p(c_i) = 1 \tag{1.23}$$

The "ball" inside the roulette wheel represents a random number in the interval [0, 1]. The individual, which section is "hit" by the roulette ball is selected (Figure 1.8). Here it is c_4.

The Stochastic Universal Sampling (SUS) is similar to the Roulette Wheel selection method. Every individual obtains again a section on the roulette wheel proportional in size to the fitness value. A fixed number of individuals $l \geq 2$ is chosen, this is the number of individuals that a single step of the SUS selection procedure will select for recombination (in Figure 1.9 it is c_4). Now create another ruler with l equi-distant points on it and of the same length as the roulette wheel. Finally a random number is generated in the interval $[0, \frac{1}{l}]$ that indicates the location of the first point on the ruler on the roulette wheel. This can be equally obtained, by wrapping the ends of the roulette wheel (to create a circle) and then generating a random number in the interval $[0, 1]$ and allocating the remaining $l - 1$ points on the roulette wheel accordingly.

In our implementation two individuals are selected for replication when using the SUS selection. Therefore a random number in the interval $[0, 1]$ is produced and multiplied by the fitness sum of all individuals. Since only two individuals should

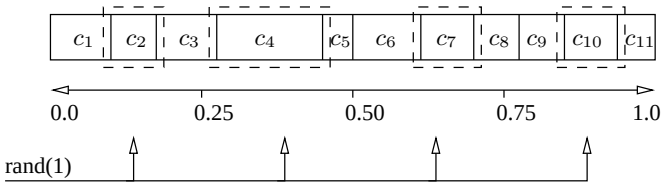


Figure 1.9: For N given individuals to select, pick a random number, and from its location pick N-1 equidistant points on the fitness landscape. In this case the selected individuals are c_2, c_4, c_7 and c_{10}.

be combined, the second individual is determined by adding $\frac{1}{2}$ *mod* 1 to the first generated random number and the result is multiplied by the fitness sum of all individuals.

Another selection mechanism is the Tournament Selection, that allows to overcome some disadvantages of the fitness proportional selection methods. In this case to select an individual c, one chooses randomly k individuals (uniformly distributed) from the population and takes the best individual (one with the highest fitness). The selection pressure is in this case controlled by the parameter k. If k is increased, the selection pressure will be raised as well. The probability of an individual of being selected using this method is calculated from equation 1.24.

$$p_s(c_i) = \left(1 - \left(1 - \frac{1}{n}\right)^k\right) \times \left(1 - \frac{i}{n}\right)^{k-1} \tag{1.24}$$

The Tournament Selection is the last selection method that is implemented within the presented GA for QLS. The number of individuals which are randomly chosen can be controlled via a parameter in the input file. In Table 1.1 line 16 the number of the tournament participants was set to 5.

1.6 Crossover and Mutation

The crossover is the primary operator of a GA. With the crossover, new individuals are produced out of selected parents by exchanging information (piece of their genome) between parents and creating new recombined offsprings. The crossover is applied to all individuals with a probability $p > 0$. There are various types of crossover operators, [Rai96].

The simplest crossover method is the **single-point crossover**. After the selection, a random number is generated such that $p_0 = rand(length(c_1)),\ p_0 < length(c_1), p_0 < length(c_2)$, with $length(c_i)$ is the length of the genome of the i^{th} individual. This

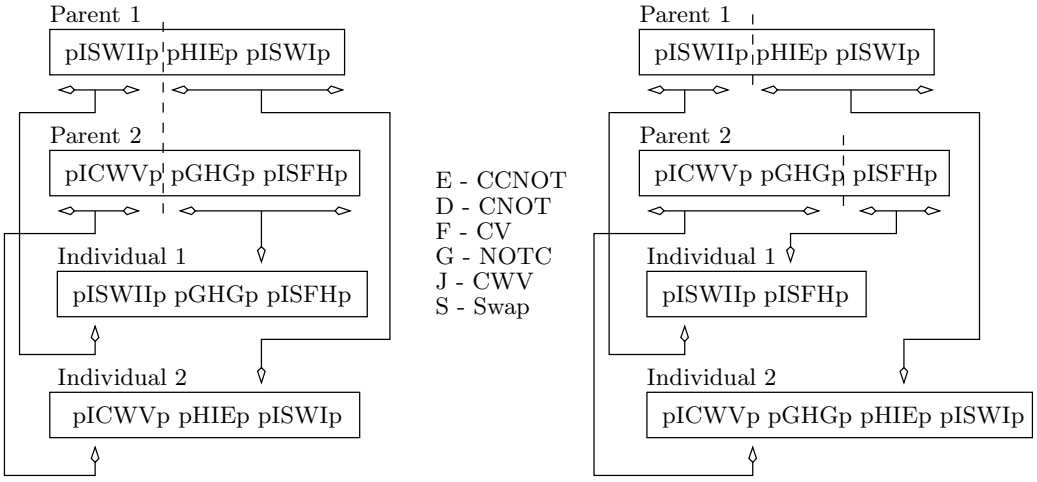


Figure 1.10: single-point crossover

random number marks the position where the crossover should take place for both individuals. This is shown in Figure 1.10 for the encoding introduced in Section 1.2.1. The single-point crossover can also be implemented. A random location of crossover is selected for each of the individuals participating in the exchange. This is shown in Figure 1.10.

The **two-point crossover** is the second crossover method. At the beginning, two random numbers are created for each individual selected for the recombination process. These numbers select the two positions within the first individual where the recombination is applied. Both numbers may not exceed the length of the first individual. The same is done for the second individual. This is illustrated in Figure 1.11.

The **Uniform Crossover** is a generalization of the n-point strategy such that for every genome it is decided by a random choice from which parent-individual it is taken. A random number on the unit interval is generated for each gene. If $r < 0.5$ the gene is taken from the first parent individual, otherwise it is taken from the second parent individual.

1.6.1 Mutation

After the Evaluation, Selection and Crossover a mutation is applied with a given probability (Table 1.1 in line 3). If a generated random number is below the specified mutation probability, then a mutation is applied. The mutation is a secondary operator and serves to insert new or lost gene material into the population, in general with a very small probability $p < 0.1$. If mutation is performed too often,

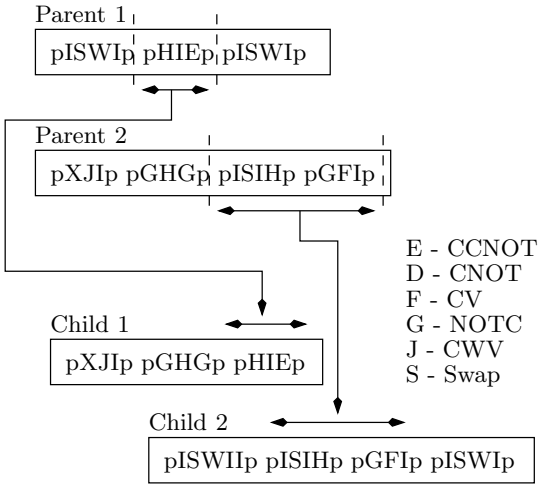


Figure 1.11: Two point crossover: in both parents, two random points are selected and the segments between such points from each parent are exchanged.

the search process degenerates to a complete random search. If the mutation is applied too rarely it does not create jumps large enough in the genomes and thus does not help to overcome some local maxima (Section 1.5).

Various mutation operators are available [Rai96], here we will describe some of them. The simplest mutation operator is a single bit flip (a random change) represented using our encoding in Figure 1.12a. This operation is naturally extended to a bitwise mutation, where the mutation operator is applied on each bit (gate) in the genome. By using the Swap Mutation (Fig. 1.12b), also called the Exchange-Based Mutation, the contents of the random selected genes is exchanged (Figure 1.12a).

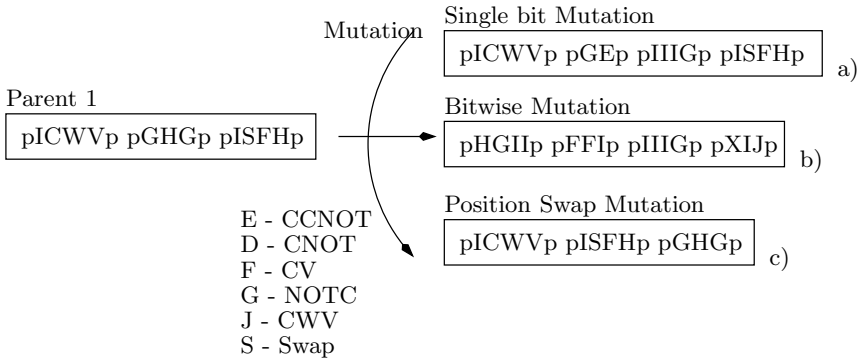


Figure 1.12: *Flip* mutation a) bitwise *Flip* mutation b) and *Swap* mutation c)

In a more detailed mutation variant, the mutation operator can be separated into three distinct actions on the genome. First, a random number selects a position

within the individual and the chosen segment will be replaced (Fig. 1.13).

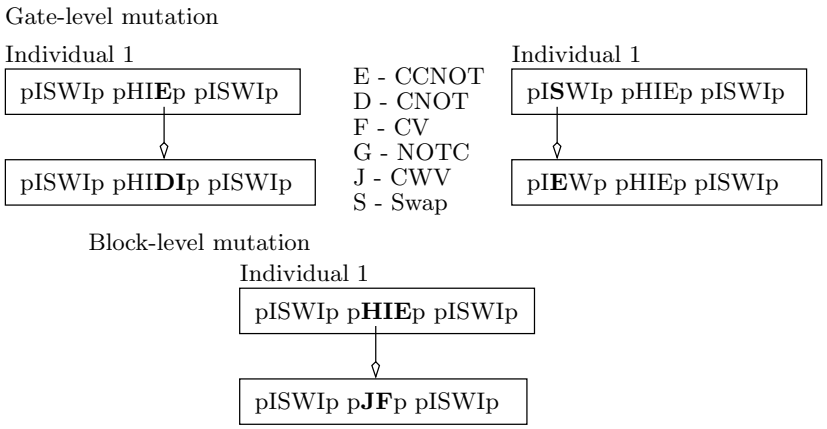


Figure 1.13: Replacement of a segment - the first mutation method

The second mutation performs an erasure of one segment somewhere within the individual on a given random position (Fig. 1.14).

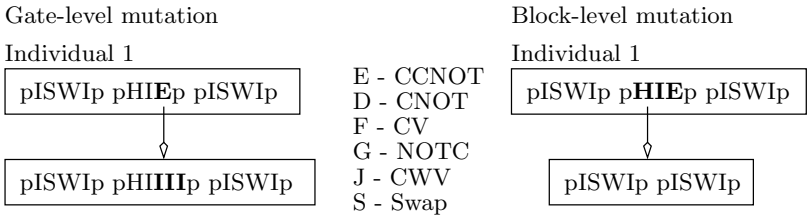


Figure 1.14: Erasure of a segment - the second mutation method

The last mutation type adds one segment to the end of an individual (Fig. 1.15).

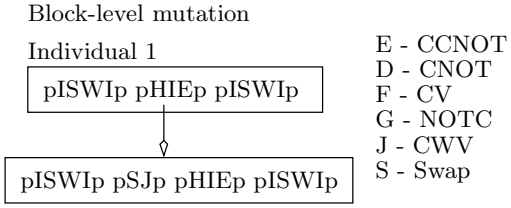


Figure 1.15: Adding of a segment - third mutation method

The ratio of how often each of these mutation types are applied in the GA is a distribution of choices over a uniform random distribution. This is because the mutation operator is applied to the full-width circuit blocks. For example, for a

circuit that is built as $[H] \otimes [CNOT] * [W] \otimes [X] \otimes [Z]$ assume that the $[X]$ gate was selected for mutation. If the replacement gate is of the same size (in this case one qubit), the gate is just replaced, and the circuit remains otherwise unchanged. In the case when the dimension of the replacement gate is greater than the size of the original gate, the whole segment containing $[W] \otimes [X] \otimes [Z]$ is deleted and regenerated so as to contain the replacement gate. Similarly, in the case when the gate selected as the replacement has less qubits than the original gate, the segment is completed by empty strings so that the width of the segment remains constant.

1.6.2 Additional GA tuning strategies

1.6.2.1 Replacement Strategies

The replacement strategy represents the method by which old individuals (parents) are replaced with the new ones (offsprings). There are various approaches to the replacement; the most common are: *generational GA* , *Steady-State GA* and *Elitistic GA* [Gol89].

1.6.2.2 Generational GA

In the generational GA, the population of offspring individuals completely replaces the population of their parents. Thus all information *transmitted* from generation to generation is completely contained in the offsprings and all information contained in the parent generation is discarded. This approach can generate such an offspring population that the best individual of the offspring generation is worse with respect to the best parent individual.

1.6.2.3 Elitism

To solve the problem of loosing information just by the selection problem the Elitism strategy is one of the best known to choose. The Elitism allows to preserve the best individuals from the parent population to be saved in the children population. Thus no matter the results of the selection process or the average fitness, the n-best individuals from the parent population are copied into the offspring population. When using Elitism, the GA copies only one best individual from the parent population to the new generation in order not to loose the best individual during the replication process.

1.6.2.4 Steady-State GA

Another approach to the problem of premature convergence, is the so-called *Steady-State GA* . Similarly to Elitism, the Steady State GA also preserves some individuals from the parent population and copies them to the offspring population; however proportions in the overall mechanism are inverted. This time most of the population is kept unchanged and only a small number of individuals is changed. The Steady State GA evaluates both populations (parent and offspring) together, and only the best individuals from both the parent and offspring populations are taken into the next generation. Unlike Elitism, however, the Steady State GA is based on the principle of overlapping population and the fact that for finding the solution only small and more controlled steps can be used. Genetic algorithms which are based on this strategy tend to converge faster. Higher mutation and crossover probabilities can be used with the Steady-State-GA, because good population members are protected through the selected replacement strategy (replacement of individuals with lower fitness values).

1.6.2.5 Boundary-Constraints

Another type of general restrictions imposed on the GA are the Boundary constraints. These constraints can be simple bounds (specification of minimum and maximum) or a linear coupling of parameter values (max or min of a function defined over the set of all elements of the individual encoding). They can be related to the feasibility or to the quality of the solution. A genetic algorithm achieves the best solution, if the encoding, the initialization and the operators of a problem are chosen, such that all possible individuals are valid solutions. That means that all boundary constraints are always satisfied and thus no mechanism needs to be considered to repair the potentially invalid individuals. However, it is possible that such an integrity of the genome cannot be assured and thus illegal genomes may be created. Therefore, these invalid individuals should get a worse evaluation, so that they won't be reproduced in the next generation.

1.6.2.6 Repair Mechanism

Beside avoiding the violation of the Boundary constraints by designing such a GA that does not generate invalid offsprings, a repair mechanism can be used. In our approach, all generated circuits are valid circuits. However the recombination can generate individuals that have very long genomes and mutation can generate completely (temporarily) invalid individuals. Individuals that are too long are not desired because they are out of the initial specifications and thus a repair mechanism

is able to shorten the genome. Individuals that undergo mutation as in Figure 1.13 can be modified so that the circuits represented by them would be invalid. As can be seen on the top-right of the Figure 1.13, the genome is mutated within one parallel segment: $pISWIp \rightarrow pIEWp$. If mutation only is applied the result would be $pISWIp \rightarrow pIEWIp$, which is an invalid circuit. Thus a repair mechanism has been implemented so that the final chromosomes in every population are valid circuits. This repair is shown schematically in Figure 1.16.

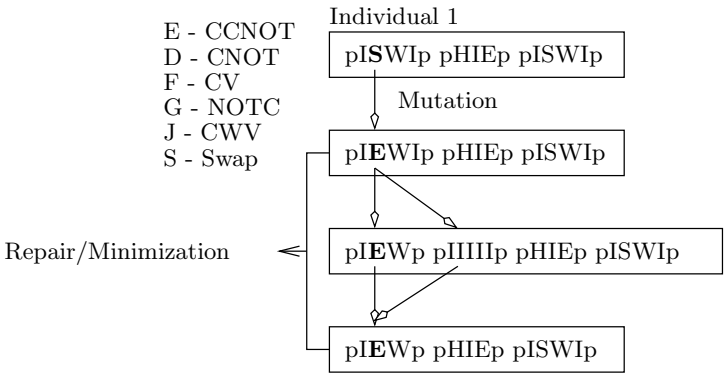


Figure 1.16: Example of the repair process. The repair mechanism is able to break down individual segments of gates and reconstruct them so as the circuit remains structurally valid.

1.6.2.7 Termination Condition

The term *termination condition* represents a set of rules that will terminate the evolutionary computation. In the simplest case, this condition terminates the computation if one of the following happens:

- A pre-specified maximum number of iterations (generations) was reached (absolute termination),
- an optimal or a sub-optimal solution was found.

In our GA the termination condition is reached if one of the individuals of the population has no error with respect to the specified gate. The information about this search including the used quantum gates with all information (truth table etc.) is printed, as well as the result with the truth table, the cost and the genotype of the individual. Also, since there is no guarantee to find a solution, the GA terminates after 10000 generations.

1.7 Conclusion

This Chapter presented the evolutionary algorithm used in this Book for the design and synthesis of permutative circuits as well as for some of th eintroduced models of sequential devices. The algorithm used is a result of both theoretical as well as empirical (experimental) knowledge that was gained during the various experiments performed. The knowledge gained during the performed taks of automated QLS was used to upgrade the GA and optimize most of its componenets for fast circuit evaluation and implementation of the GA on parallel devices such as GPU. The describes results in this book are thus selected from the most significant for the topic at hand as wll as such results are presented that shows that the GA is capable to succesfgully design the desired quantum circuits.

Chapter 2

Examples of Evolutionary Search for Quantum Logic Synthesis of Quantum Combinatorial Circuits

2.1 Synthesis of Quantum Circuits with Evolutionary Search

In this chapter we present the experimental settings, results, extensions and observations of the basic evolutionary approach to Quantum Logic Synthesis. In particular, this chapter describes how by using the GA from Chapter 1 we were able to re-synthesize some already known universal gates and minimize them in some cases as well as find novel realizations of some of the gates.

Although our methods are aimed to discover a logic circuit representing a desired function, in order to be more technology specific we assume in some search variants the use of quantum gates representing the Nuclear Magnetic Resonance (NMR) quantum computers [YSPH05, ZLSD02, ?]. Also, as shown in chapters ?? and 1, we assume that the cost of every gate is calculated using some quantum cost function which may change from experiment to experiment. Finally the goal of this Chapter 2 is also to demonstrate the methodology used to find the least costly (in number of gates) realizations of well-known Toffoli, Fredkin, Miller, Peres and Peres family gates as well as to design new quantum gates given their specifications.

The Evolutionary Quantum Logic Synthesis (EQLS) has been explored from various points of view in the last decade. On one hand the Genetic Programming has been widely used to synthesize quantum circuits and logic functions [Rub01, Lei04, MCS04, MCS05, Spe04, SBS08, LPK10]. On the other hand GA based methods have

been applied to various quantum circuits as well [Yab00, LPMP02, LPG+03, Luk09, LP08, LPK10]. In general the focus of these approaches is either on a particular function (Reversible, quantum-permutative) or a general approach is used to see how well the evolutionary approach deals with this difficult problem. Several problems in EQLS have already been studied and analyzed, some of them are complexity of the quantum search space, high dimensionality of the quantum space, large number of quantum gates, etc. From these previous studies, it can be concluded that Evolutionary methods are well suited for research aimed to discover novel principles and novel quantum gate realizations of moderate size [Luk09]. Following this reasoning, in this book the focus is on the discovery and a deeper understanding of dynamics of the EQLS under various experimental conditions. In particular we are searching for novel ways of synthesizing quantum algorithms and circuits using classical evolutionary methods.

The synthesis of quantum logic circuits using evolutionary approaches has two fundamental aspects:

- First, it is necessary to find the circuit that: either (A) exactly corresponds to the specification, or (B) differs only slightly from the specification. Case (A) is verified by a tautology of the specification function and the solution function. In case of a truly quantum circuit this is done by a comparison of unitary matrices or by the comparison of the observable results after the measurement operation. In case of permutation functions this can be also done by comparing the truth tables. Observe that non-permutative matrices cannot be represented by truth tables which leaves the representation of unitary matrices as the only canonical function representation. This representation is responsible for less efficient tautology verification during fitness function calculations, which considerably slows down the software execution time. Case (B) calculations for permutation circuits are verified by an incomplete tautology (tautology with accuracy to all combinations of input values and with arbitrary logic values for don't care combinations). In some applications such as robot control or Machine Learning it is sufficient that the specification and the solution are close, like, for instance, differing only in a small percent of input value combinations (Chapter **??**).

- Second, fundamental aspect of quantum logic synthesis is that the cost of the circuit has to be as close as possible to the known (or expected) minimum cost, in order to allow the least expensive possible quantum hardware implementation (like the minimum number of electromagnetic pulses in NMR technology).

Both of the above introduced problems are adressed in this chapter using a Genetic Algorithm. GA is well suited to search a relatively large problem space

where no global cost or objective function can be defined and where there is a little knowledge about the structure of the problem space. The use of GA does not guarantee a succes in the synthesis process but it is a great starting point and tool for exploration and principles extraction from otherwise a too large problem space. As such the GA is extensively used for circuit synthesis in this chapter and various heuristics improving its performance are demonstrated. Also, hardware description (GPU computing) used to accelerate the circuit computation is presented and described.

To cover experimentally the two aspects mentioned above, the presented experimentations cover benchmarks that can be separated into the following sub-categories:

- Exact Synthesis problems of completely specified functions
  - Evolutionary synthesis of permutative universal gates such as Fredkin, Toffoli, Majority and Miller gate using exact measurement based evaluation of costs (single and multi-qubit measurement)(ME evaluation, Section 2.3).
  - Evolutionary synthesis of permutative universal gates such as Fredkin, Toffoli and the Entanglement quantum gates using exact matrix evaluation based evaluation of costs (EE evaluation, Section 2.4).
- Approximate Synthesis problems of incompletely specified functions
  - Evolutionary Synthesis of permutative universal gates such as Fredkin, Toffoli and some additional benchmark functions (Section 2.3, 2.4 and Chapter **??**).
  - Algorithmic, user-driven pseudo-random exhaustive search for permutative circuit structure-based quantum logic synthesis using the GAEX and EX algorithm (Chapter **??**).
- Approximate Synthesis problems of incompletely specified functions for Machine Learning (Chapter **??**)
  - The synthesis of quantum circuits with quantum output states for novel robotic behaviors (Chapter **??**).
  - The synthesis of FSM represented as quantum circuits (Chapter **??**).

This Chapter is organized as follows. Section 2.2 discusses the experimental settings and conditions of the GA. Section 2.3 discusses the obtained results of the evolutionary search using the ME evaluation methodology and section 2.4 analyzes the results of the evolutionary search using the EE evaluation. Finally section 2.5 concludes this chapter and discusses all the benchmarks and obtained results.

2.2 Experimental setup of GA

2.2.1 Input Sets of Quantum Primitives

Chapter **??** presented a general approach to the calculation of the cost using single and two-qubit quantum gates or pulses as unit components of the total circuit cost. For higher level gates (such as CCNOT, Fredkin, etc.), we assume that each gate has a cost equal to the sum of its component gates in a given model. For instance, the single-qubit gates have a unit cost 1 (including Identity). The two-qubit operations have the cost either given along with the definition of the gate (by the user) or they must be realized by synthesis algorithm from smaller primitives. In such a case the cost of the used universal gates is equivalent to the realization that the algorithm has built; all gates are either derived from the initial gate set or are in the initial set. Other costs, such as those presented earlier can be used as well.

Note that in the classical reversible logic synthesis such as [MMD06, MDM05, WGMD09, **?**] the quantum cost is given as follows: single qubit gates have no cost and two qubit gates have a cost of 1. Thus a Toffoli gate realized using the Cv and CNOT quantum gates has a cost of 5 and Fredkin has a cost of 7. The general cost function we adopt here is the cost of 1 for single and two qubit gates. This cost is more accurate than assuming that single qubit gates are for free and allows to easily make the comparison between our cost and the one used in reversible logic synthesis literature.

As introduced in Chapter **??**, the used gates are separated into sets. The experiments described in this chapter use three sets. Each input-gate set contains a small set of single-qubit unitary transformations that, in general are selected from the set $S_1 = \{$Wire, X ($R_x(\pi)$), Y ($iR_y(\pi)$), Z ($iR_z(\pi)$)$\}$.

The three input gate sets categories are:

1. Limited angle rotations: $S_{lr} = S_1 \cup \{R_x(\theta), R_y(\theta), R_z(\theta), I_{zz}(\theta)\}$ (with angles $\theta = \pm\pi, \pm\frac{\pi}{2}, \pm\frac{\pi}{3}$),

2. Full: $S_f = S_1 \cup \{$ H, CNOT, CCNOT (Toffoli), SWAP, C_SWAP (Fredkin), Majority$\}$.

3. Partial: $S_p = S_1 \cup \{$SWAP, V, $V^\dagger$.C_V, C_V$^\dagger\}$,

The S_{lr} set represents one of the most general units of quantum computing - single-qubit rotations and the two-qubit interaction gate. All four operators in S_{lr} are parameterized by θ and thus the set size depends on the desired and allowed precision. This input gate set was used to verify results from [LKBP06] by searching

the underlying problem space. However, because each logic operation in a system built from these smallest segments (NOT is a single pulse, Hadamard is two pulses, CNOT five pulses, CV five pulses, etc) it is very difficult for the GA to keep so many small component gates together and thus the synthesis using this set is one of the most difficult but potentially provides the smallest possible quantum gate cost.

The set S_f represents the full set of gates (using universal gates as input) and was used to determine the functionality of the GA. In particular, the full set was used to verify the GA capacity to re-synthesize a gate. Naturally when the target gate is an element of the full set, this gate is removed from the set of input gates. This set is also used when searching for more difficult permutative functions.

The third set S_p is one of possible partial sets used to search for smaller universal gates and for novel realizations of universal gates. In general, the set S_p contains a small subset of quantum gates. For instance most common partial input-gate sets are $\{I, H, CNOT\}$, $\{I, V, V^\dagger, CV, CV^\dagger, CNOT\}$, $\{I, X, Y, Z, Phase, CNOT\}$, $\{X, H, Z, CZ, CH\}$ and so on.

The results are separated into two categories based on the ME and the EE error evaluation methods. In both cases, the results of synthesis of Toffoli, Fredkin and Miller gates were are analyzed in details for both ME and EE evaluations.

2.3 Discussion of the Results of the Evolutionary Quantum Logic Synthesis using ME evaluation

2.3.1 Toffoli Gate

Table 2.1: Fixed parameters during search for Toffoli gate

Population	100	Generations	200
Mutation	0.05	Crossover	0.8
t_{min}	7	t_{max}	12
σ	6		
input gate set	$\{I, X, V, V^\dagger, CNOT, CV, CV^\dagger\}$		

The Toffoli gate was successfully reinvented by our GA as well as novel implementations have been found. In this section, the GA performance is analyzed with the focus on the multi-qubit measurement and the single qubit measurement. The experiments are presented and analyzed with respect to fixed parameters shown in

Table 2.1.

Table 2.1 shows parameters t_{min} and t_{max} (they represent the approximate size limits of the circuits analyzed) that limit the problem space to circuits between seven and twelve segments and with a user-given minimal cost of 6. This cost is suboptimal, despite the fact that as already shown, the Toffoli gate can be synthesized using the S_p set of gates with 5 two-qubit gates. Thus the default cost is 5, but because there are four single qubit Identities in a Toffoli gate the minimal cost of the Toffoli gate is $C_{toffoli} = 5 + 4 = 9$.

The reason that all gates (including Identity) have to have a cost, is the fact that if there were gates with cost 0, the synthesizer would prefer 0-cost gates over other gates that could be useful in the synthesis process. Consequently gates with no cost could lead to erroneous circuits creating a local minimum leading the evolutionary process to less successful searches (the GA is stuck in some local optimal fitness). This means that circuits with a high error and low cost will have the same fitness as circuits with lower error and higher cost.

Figures 2.1 and 2.2 represent the averages of the fitness value over 50 runs, the current best fitness value, the cost and the average error per generation as well as the fitness, the error and the cost for the currently best solution.

2.3.1.1 Single-qubit ME model

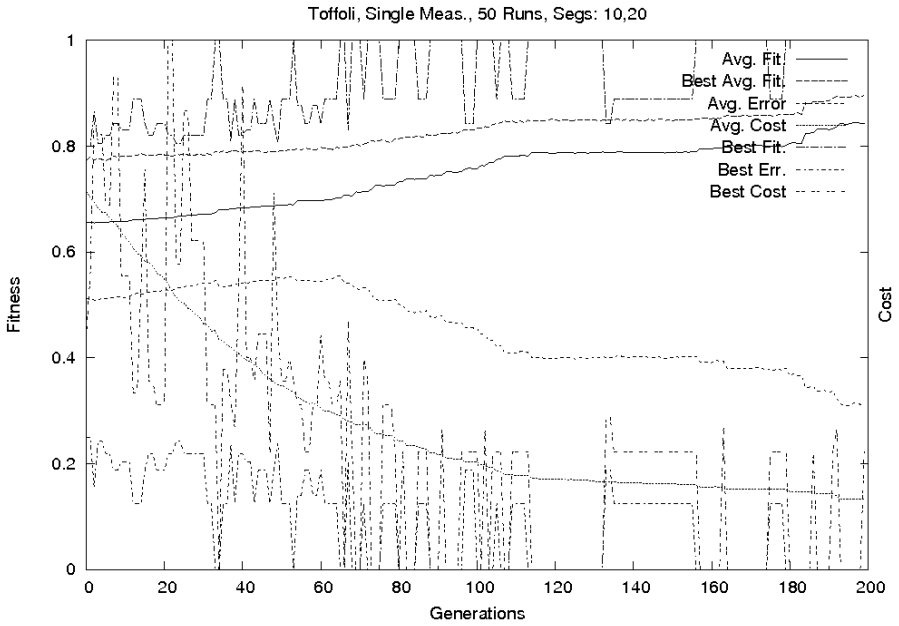
Figure 2.1 shows the results of the evolutionary synthesis of Toffoli gate using single-qubit measurement (measuring only single qubit) and Figure 2.2 illustrates the multi-qubit measurement (measuring all output qubits). Figures 2.1a and 2.1b represent the results of the search for Toffoli gate using single qubit ME evaluation and using between 10 and 20, and between 20 and 40 segments, respectively.

Observe that the single qubit measurement is statistically successful when the GA is exploring the problem subspace of a circuit size where the solutions can be found quickly. This can be seen on Figure 2.1a: the cost decrease over the evolution of the population of the solutions and the overall increase of the fitness value indicate the algorithm convergence.

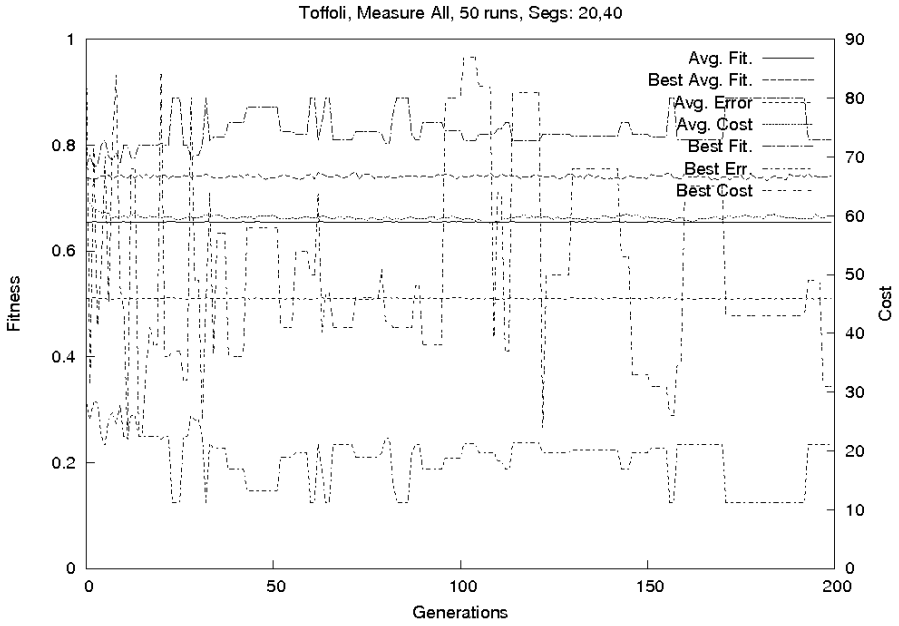
However, such convergence is only observed when the parameterization of the GA corresponds to an easily found global minimum. Such parameterization is generally unknown but can be determined:

- experimentally - by combining parameters and observing results from performed experiments

or



(a) Single-qubit measurement, $t_{min} = 10$, $t_{max} = 20$



(b) Single-qubit measurement $t_{min} = 20$, $t_{max} = 40$

Figure 2.1: Results of Toffoli gate using the single-qubit measurement for two different settings of t_{min} and t_{max}. Observe the overall convergence of the evolutionary search in (a) and compare to relative non-convergence in (b) due to over-estimated parameter values t_{min} and t_{max}

- theoretically - by specifying parameters closest to a known minimal realization of a gate.

Observe that in Figure 2.1b the size of the circuit given by t_{min} and t_{max} is too large and the evolutionary process is less successful.

2.3.1.2 Multi-qubit ME model

Figures 2.2a and 2.2b represent the results of searching for Toffoli gate using multi-qubit ME evaluation for fitness function calculation and using between 10 and 20 circuit segments and between 20 and 40 circuit segments; respectively. In all these experiments the fitness function from eq. 1.14 is used.

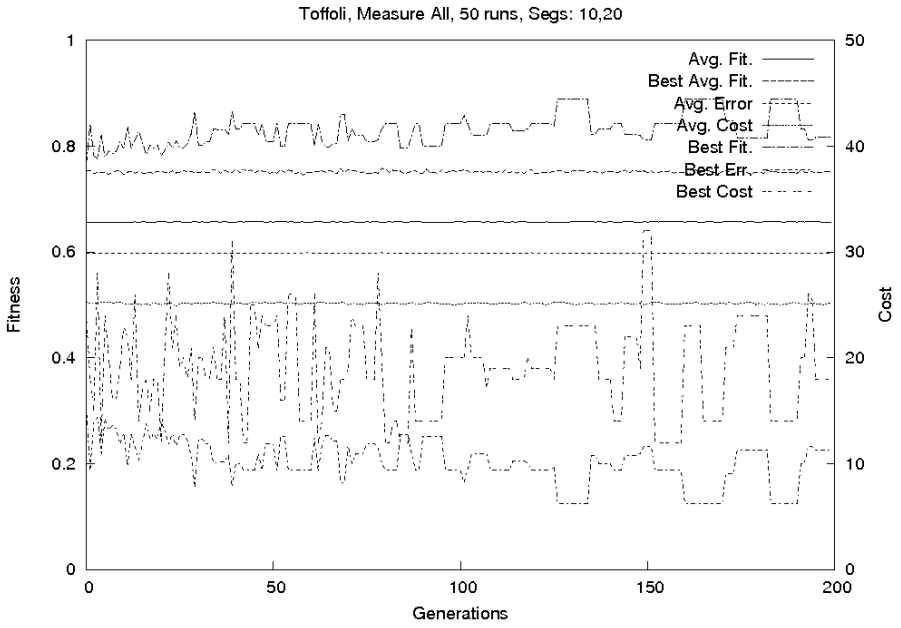
When using the multi-qubit measurement, the output is a 3×3 reversible function and as can be seen the evolutionary process is less successful (Figure 2.2a and 2.2b). This is due to the fact that in the alloted number of generation cycles, either only an approximate solution was found or most of the GA runs did not converge in the allocated time. In order to find an exact solution in general it is required to either provide more computational time or restrict the problem space of the GA by reducing the number of input gates or the number of the allowed segment gates for instance.

Figures 2.2a and 2.2b can be commented by the following observations.

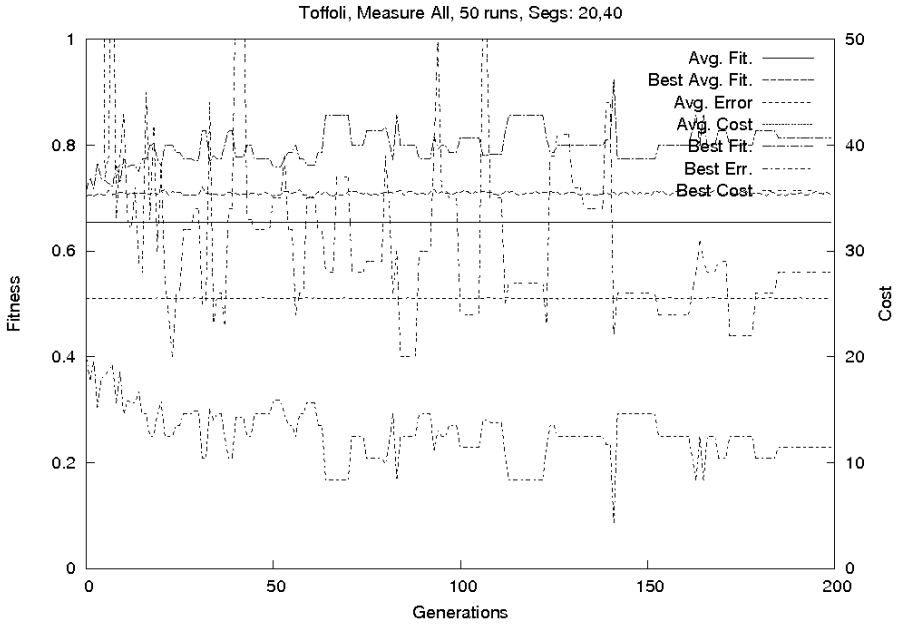
- The first set of curves represents the averages of the population (fitness value, the current best fitness value, the cost and the average error). This set of curves also shows the trend of the evolutionary process. The shape of these curves is in most graphs slowly increasing as more and more solutions are found. Observe that the average values can be almost constant when the solutions are rare due to the time constraint.

- The second group (the fitness, the error and the cost for the currently best solution) represents the average fitness of the best individual and the bottom line represents the associated error for the best individual. The fitness is the topmost curve mapped on the primary axis. The error curve is symmetric to the fitness curve with respect to the line $fitness = 0.5$ and the cost curve has the same shape as when using the disproportionate fitness (mapped on the secondary y axis on the right)

Observe that the GA convergence is not seen here, despite that the algorithm found the correct solutions (Example of this can be seen in Figure 2.3 where a successful search for a Toffoli gate is shown despite the overall non-convergence of the



(a) Three-qubit measurement $t_{min} = 10$, $t_{max} = 20$



(b) Three-qubit measurement $t_{min} = 20$, $t_{max} = 40$

Figure 2.2: Results for Toffoli gate using the multiple-qubit measurement for two different settings of t_{min} and t_{max}

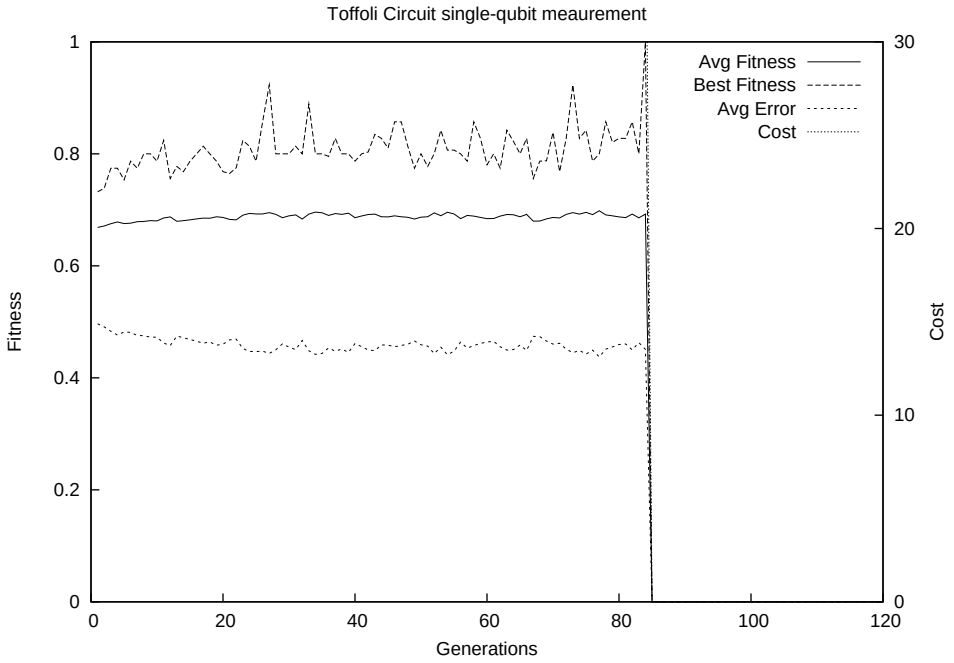


Figure 2.3: Example of successful synthesis of the Toffoli gate using a partial input gate-set S_p and a single-qubit ME method.

evolutionary search). The lower convergence rate results from the fact that while synthesizing with multiple single qubit measurements, the solution matrix must match three disjoint observations (measurements). Naturally this is more difficult than when using a single measurement. Also, similarly to the single-qubit measurement case, increasing the parameters t_{min} and t_{max} generates worse results if these parameters overestimate the sizes of possible solution circuits. Finally, the main differences between Figures 2.2a and 2.2b are the circuit cost (observe that in Figure 2.2b the cost of circuits is much higher as a consequence of synthesizing a larger circuits) and the circuit fitness (in Figure 2.2b the fitness is lower, again as a result of an overestimation of the circuit size).

2.3.1.3 Comparison of Single-qubit and Multi-qubit ME methods

Figure 2.4 shows several Toffoli gates that were discovered using our GA with the single-qubit ME error model. Observe, that despite all these circuits represent correctly the single output function, the other qubits in the whole state can be still in a superposition; for instance eq. 2.1 shows the unitary matrix of the circuit in Figure 2.4c representing the single-output Toffoli function, that is built from controlled-V

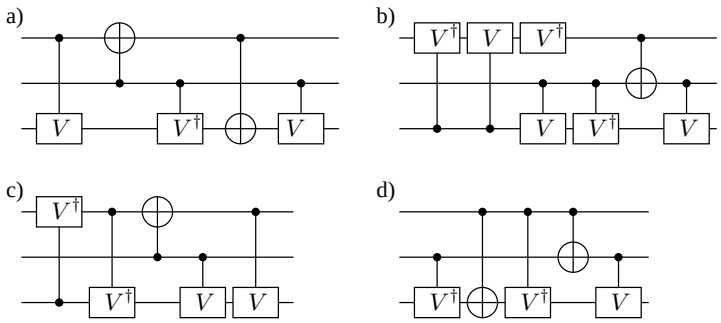


Figure 2.4: Circuits discovered by the GA as a single output Toffoli Gate using the single-qubit ME evaluation. (Unitary matrix representing the Toffoli gate is given in eq. 2.1)

and controlled-V† gates only.

$$
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.5-0.5i & 0 & 0 & 0 & 0.5+0.5i & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0.5+0.5i & 0 & 0 & 0.5-0.5i & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0.5+0.5i & 0 & 0 & 0 & 0.5-0.5i & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5-0.5i & 0 & 0 & 0.5+0.5i & 0
\end{pmatrix} \tag{2.1}
$$

Observe that for any three qubit logic input, the operator from eq. 2.1 generates a 3×3 quantum probabilistic logic function; the whole output state $U|\psi\rangle$ can be in a superposition (eq. 2.2),

$$
\begin{aligned}
U\left|001\right\rangle =& \frac{1+i}{2}(-i\left|001\right\rangle + \left|101\right\rangle \\
Measurement(\frac{1+i}{2}(-i\left|001\right\rangle + \left|101\right\rangle) =& \frac{1}{2}\left|001\right\rangle\left\langle 001\right| + \frac{1}{2}\left|101\right\rangle\left\langle 101\right|
\end{aligned} \tag{2.2}
$$

while the measurement of only the output qubit generates a deterministic value corresponding correctly to the third qubit of the Toffoli gate. Both of the possible outputs are shown in Table 2.2.

Thus using single-qubit measurement for quantum circuit synthesis, a $n \times 1$ reversible logic function can be easily synthesized but at the cost of losing the permutative reversibility on other qubits. In this case, the Toffoli gate was synthesized using

Table 2.2: Truth table of a 3×3 quantum probabilistic and a 3×1 deterministic functions generated by the circuit from Figure 2.4c.

$q_2q_1q_0$	All Measured $q_2q_1q_0$	Single Measure q_0
000	$\|000\rangle\langle 000\|$	$\|0\rangle\langle 0\|$
001	$\frac{1}{2}\|001\rangle\langle 001\| + \frac{1}{2}\|101\rangle\langle 101\|$	$\|1\rangle\langle 1\|$
010	$\|110\rangle\langle 110\|$	$\|0\rangle\langle 0\|$
011	$\frac{1}{2}\|011\rangle\langle 011\| + \frac{1}{2}\|111\rangle\langle 111\|$	$\|1\rangle\langle 1\|$
100	$\|100\rangle\langle 100\|$	$\|0\rangle\langle 0\|$
101	$\frac{1}{2}\|001\rangle\langle 001\| + \frac{1}{2}\|101\rangle\langle 101\|$	$\|1\rangle\langle 1\|$
110	$\frac{1}{2}\|011\rangle\langle 011\| + \frac{1}{2}\|111\rangle\langle 111\|$	$\|1\rangle\langle 1\|$
111	$\|010\rangle\langle 010\|$	$\|0\rangle\langle 0\|$

the single-qubit ME error model, which generates the single-output reversible logic function on the output target but the control qubits are either not restored to the original input values or can even remain in some quantum superposed or entangled state (Table 2.2).

2.3.2 Synthesis of Fredkin Gate

Table 2.3: Parameter values used during search for Fredkin gate

Population	100	Generations	200
Mutation	0.05	Crossover	0.8
t_{min}	10	t_{max}	20
σ	10		
input gate set	$\{I, X, V, V^\dagger, CV, CV^\dagger, CNOT\}$		

Similarly to Toffoli, the Fredkin gate was successfully re-synthesized and novel circuit implementations have been found for it with our GA.

2.3.2.1 Single-qubit ME model

The experiment was started with the single measurement synthesis method that resulted in some interesting gates. One of synthesized Fredkin gates is shown (with minimization) in Figure 2.5; the blocks named C_0 and C_1 are the aggregated logic blocks resulting from the concatenations of gates following the rules introduced in Chapter **??**. Again the minimization allows to note that this gate is not minimal.

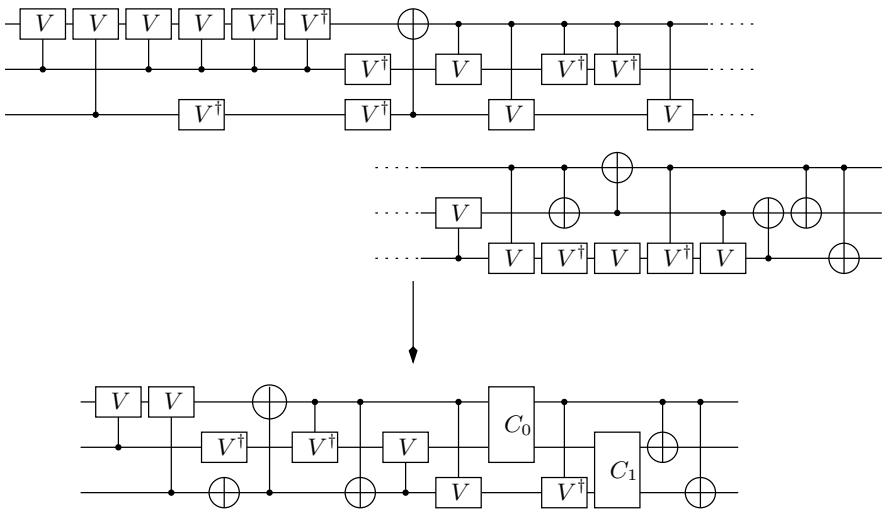


Figure 2.5: Example of realization (with minimization) Fredkin gate using single qubit ME model.

The gate is however still interesting because it provides the bottom two qubits with correct function while the top qubit remains in a superposed state.

Observe that the gate (Figure 2.5) is not Fredkin but negated Fredkin. This means that while Fredkin gate defined as single output function is given by $f_{Fredkin} = [0, 1, 0, 1, 0, 0, 1, 1]$ the discovered circuits implements $f_{\overline{Fredkin}} = [1, 0, 1, 0, 1, 1, 0, 0]$. Such circuits can be generated during the synthesis process, but as it is not guaranteed that potential solutions will lead to actual solutions, the evolutionary process might not find a solution at all. The synthesis of such *negated* gate is interesting because it is easy to obtain the correct gate from it by simply negating the output value.

Because the Fredkin gate is essentially a single-controlled-qubit two-qubit-target unitary gate, the ME error model can be specified for two measured qubits. Some of the results using the full input-gate set S_f are shown in Figure 2.6. As can be seen the synthesized gates can be minimized to the well known Fredkin circuit realization but as only two qubits are measured, the third qubit is modified by an additional gate. This additional gate can be removed without changing the output of the target two qubits. Moreover, naturally because Fredkin use only a single qubit to control no superposition can be obtained on the control qubit and thus two qubit measurement is sufficient to obtain the classical fully reversible gate.

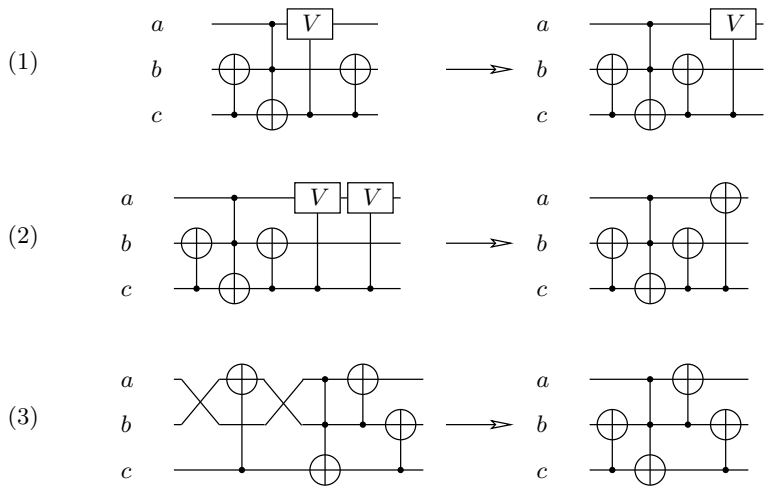


Figure 2.6: Example of realization (with minimization) of Fredkin gate using the two-qubit ME error model and the S_f input-gate set. Observe that all presented gates contain the well known realization of Fredkin ($[NOTC][TOFFOLI][NOTC]$) on qubits b and c, but with additional gates changing the value on the third, unmeasured qubit a.

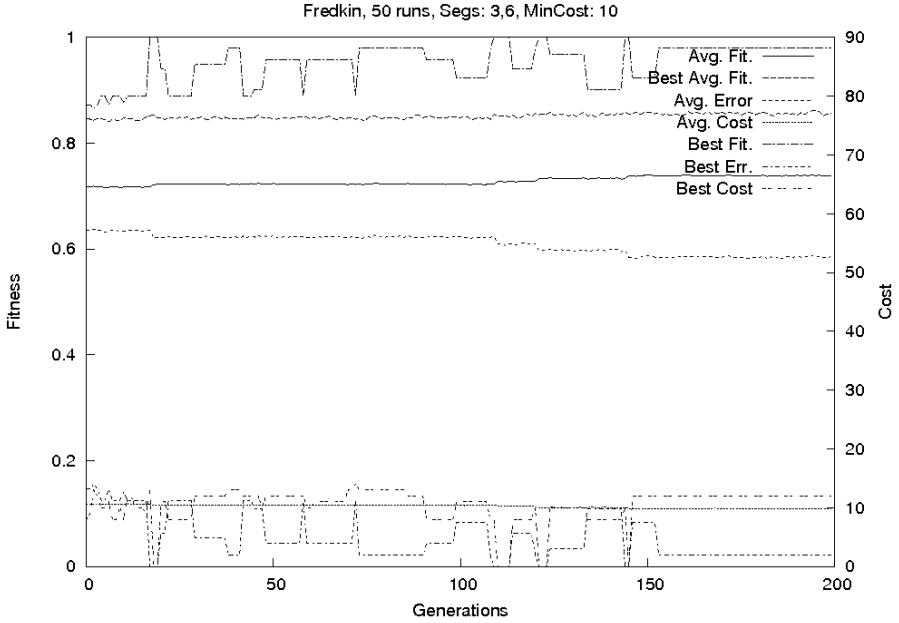
2.3.2.2 Analysis of using Partial vs. Complete input-gate set.

The GA performance is analyzed with the focus on the effect of the size of the input gate-set; the convergence of the algorithm is compared with different sizes of th einput gate set. Table 2.3 shows the parameters that have been used for the following experiments as constants. In this case, the t_{min} and t_{max} parameters were increased in order to give the GA enough space for the search. The minimal cost σ remains the same. As in the Toffoli case this cost underestimates the cost of a minimal Fredkin gate built using the GA.

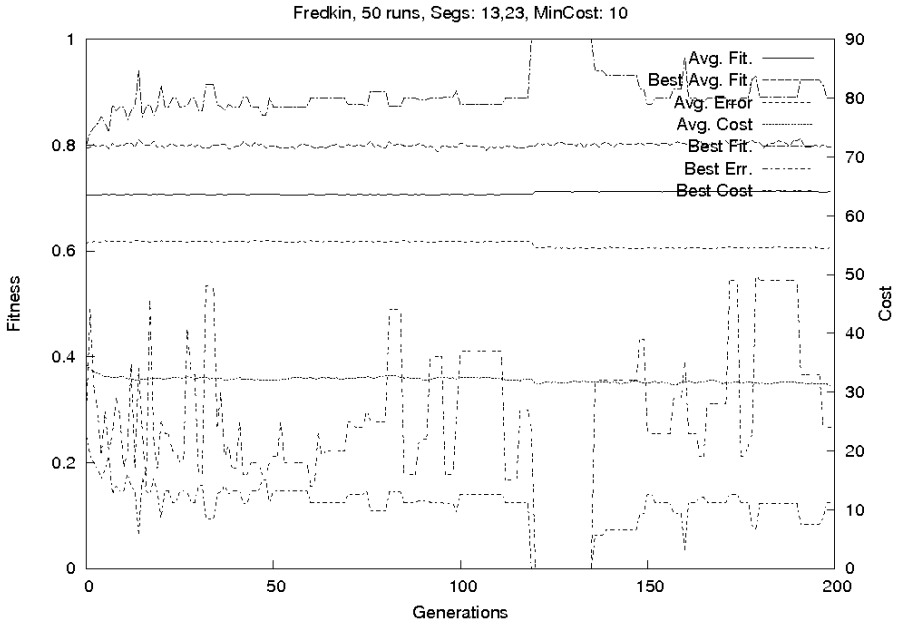
The synthesis process was more difficult (smaller success rate and longer runs required) in the case when incomplete sets of gates have been used. Figures 2.7 and 2.8 represent two sets of runs, both using fitness function from eq. 1.14 and the ME error evaluation evaluation.

Figures 2.7a and 2.7b present the search experiments using the S_f input gate-set and Figures 2.8a and 2.8b show the results from the same experimental settings but using the S_p input gate-set.

Observe, that Figures 2.7a and 2.7b illustrate the problem of over-estimation of the size of the circuits with respect to the given input set. Figures 2.8a and 2.8b illustrate the problem of under-estimation of the size parameters: observe that in

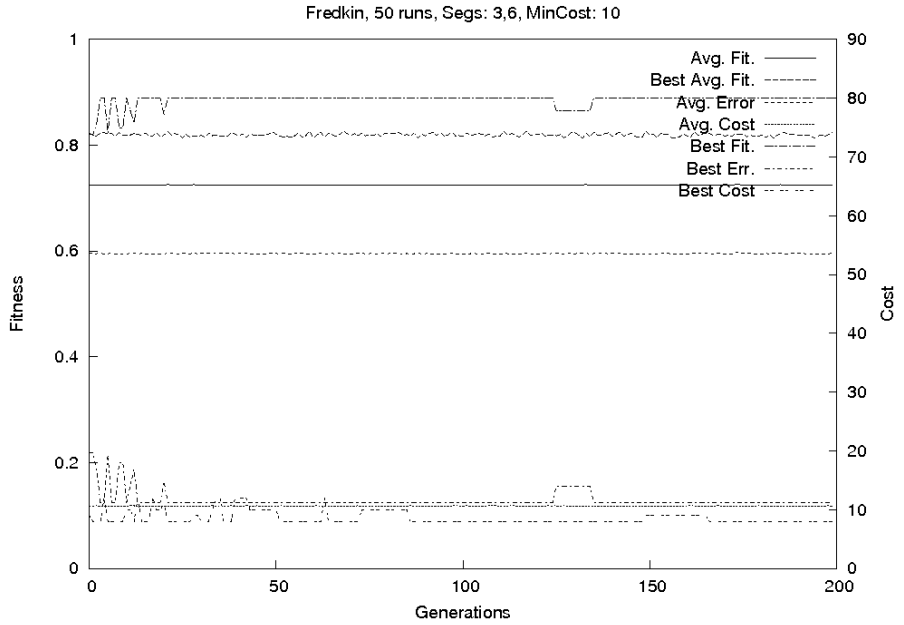


(a) Full set S_f, $t_{min} = 3$, $t_{max} = 6$

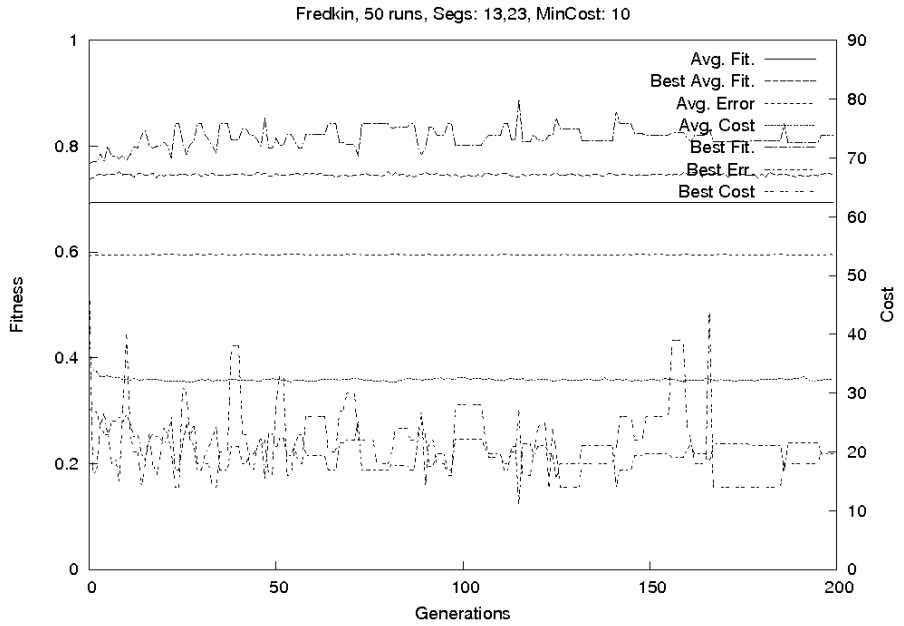


(b) Full set S_f, $t_{min} = 13$, $t_{max} = 23$

Figure 2.7: Comparison of results for Fredkin gate using the full input gate-set S_f for two different settings of t_{min} and t_{max}. Similarly to the case of Toffoli gate, the overestimation of the size parameters leads to poorer results.



(a) Partial set S_p, $t_{min} = 3$, $t_{max} = 6$



(b) Partial set S_p, $t_{min} = 13$, $t_{max} = 23$

Figure 2.8: Comparison of results for Fredkin gate using the partial input gate-set S_p and for two different settings of t_{max} and t_{min}. (a) - the values of parameters t_{max} and t_{min} are too small for the GA to find a solution and thus the statistical average is stuck (shown as a straight line on the top) in a locally best solution. (b) - an increase in size allows the GA to explore larger circuits and also to find solutions, despite the fact that overall convergence is not observed here.

Figure 2.8a the GA is stuck in a local minimum while in Figure 2.7b the algorithm searches larger circuits allowing it to find solutions (Using the S_p input gate-set for $(t_{min}, t_{max}) = 3, 6$ no solution is found; the best individual fitness average is stuck at a local minimum as the lower upper bound t_{max} does not allow the GA to explore larger circuits.).

Example of successful search for Fredkin gate is shown in Figure 2.9. It illustrates (as in the case of Toffoli gate) that the evolutionary process finds a solution despite an overall convergence (averaged over 20 runs of the GA) is not observed. Observe, that during the evolutionary run, the GA fitness (in each generation) is stuck at 0.8 and when the solution is found the fitness goes to 1 and the error goes to 0 in a single jump.

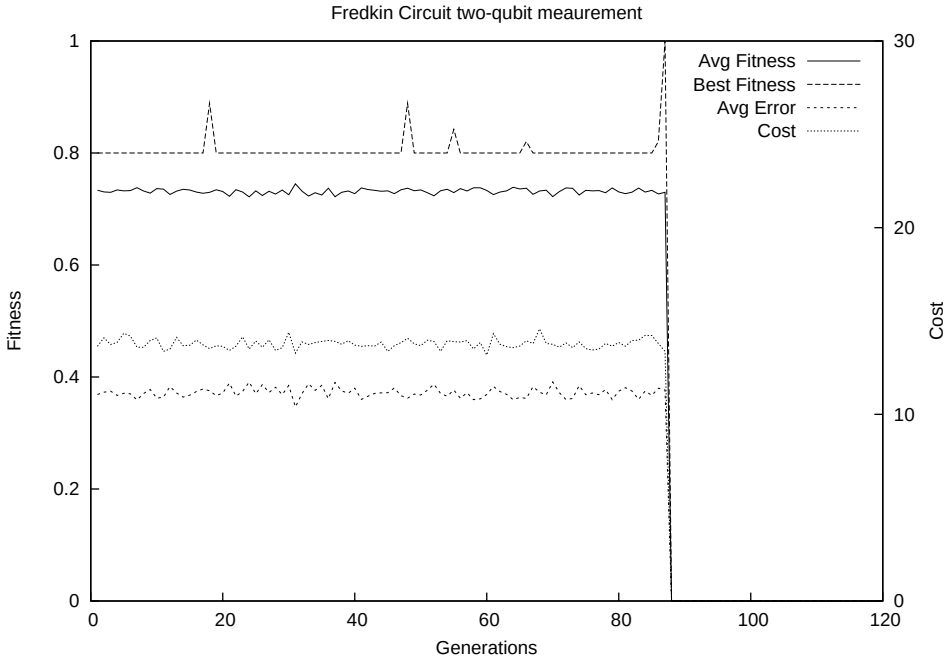


Figure 2.9: Example of successful synthesis of the Fredkin gate using a full input gate-set S_f and a two-qubit ME method.

These observations confirm that the algorithm behaves according to our expectations; by increasing the size of the input set the time to find the solution increases as well. This is natural as the GA can chose from more input component gates and thus the search space is larger. Moreover these results illustrate that using a universal gate in the input-gate set naturally reduces the required size of the circuits to be built compared to the case when such a gate is missing from the input-gate set. This is because as the GA is an artificial-evolution driven random process, circuits

may carry a lot of unminimized gates. Thus to create such gates more efficiently, a logic minimizer or some additional heuristics can be added to the GA in order to provide the desired boost.

2.3.3 Synthesis of a Majority Gate

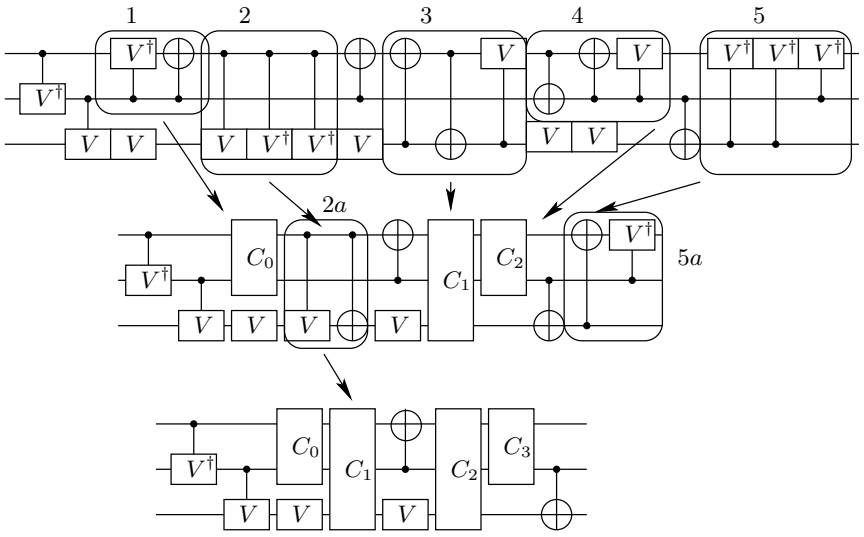


Figure 2.10: Example of the synthesized Majority circuit with the consequent minimization and final result (using the ME error model evaluation).

The synthesis process of Majority gate which was first realized in [Mil02] (as one output of a permutative 3×3 reversible function) is presented in this section. Here our interest is only in single-qubit output Majority Gate. The Figure 2.10 shows a realization of the Majority gate found by the GA as well as the final circuit after minimization. The blocks denoted by numbers 1,2,3,4,5 can be each combined into a simpler gate. This is shown by transforming for instance the block 1 to the compound gate C_0. The gates denoted by C_0, C_1, C_2 and C_3 are compound gates created by concatenating (collapsing) two or more gates that occupy the same wires and that do not have any other gate between each other. For instance, the block 4 that is composed of $[CINOT][NOTIC][VIC]$ is transformed into a single quantum gate because all gates in it occupy the same qubits and no other gate exists between them.

Observe that each step of the minimization first minimizes the gates ($V \odot V = NOT$), and only then minimizes the final gate concatenation is done. Thus at the end of the process, there are four blocks C_0, C_1, C_2, C_3. Each block is created from [NOT$V^\dagger$][NOTC] (on top two wires), [CIV][CI$V^\dagger$][CI$V^\dagger$] $\rightarrow$ [CIV][CINOT],

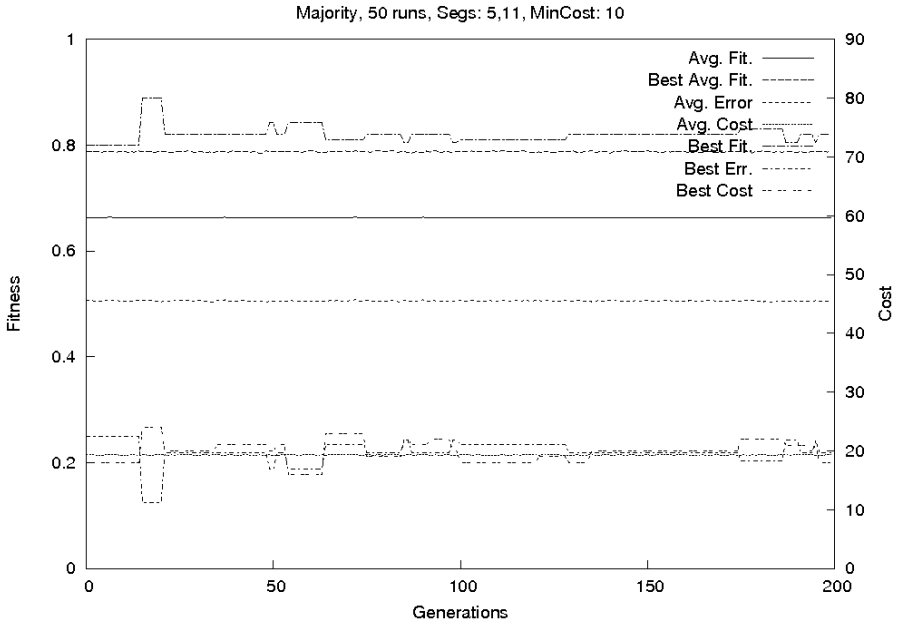


Figure 2.11: Synthesis of Majority gate using single-qubit ME with partial input gate set S_p and $t_{min} = 5$, $t_{max} = 11$.

[NOTIC][CINOT][VIC] and [CNOT][NOTC][VC] (on top two wires) respectively. Note that the block 5 ([NOTIC][$V^\dagger \otimes I$]) was removed because the gates it contains are irrelevant to the single output function realized on the bottom wire.

2.3.3.1 Influence of t_{max} and t_{min} on the Evolutionary QLS

The Majority gate was successfully synthesized using the GA with solutions close to the cost-optimal circuits. Similarly to the Fredkin gate synthesis, the Majority gate synthesis requires longer time (and is less successful) than in the case of the Toffoli gate because the GA had more troubles to keep larger groups of gates together allowing to generate universal gates such as Toffoli or Fredkin which are required to build the Majority gate.

Figures 2.11, 2.12 and 2.13 illustrate the Evolutionary Synthesis process of the Majority gate for various settings for the minimal and maximal sizes of the synthesized circuit; t_{min}, t_{max} respectively. Observe, that as expected (and as already observed in the cases of Toffoli and Fredkin), the prior knowledge about the desired circuit size directly affects the results: for larger circuits the precise settings of the same cost and t_{min}, t_{max} parameters are much more important than for smaller circuits. Also observe, that the problem space represented by the bounds given by the minimal t_{min} and maximal t_{max} numbers of segments must contain circuits with the

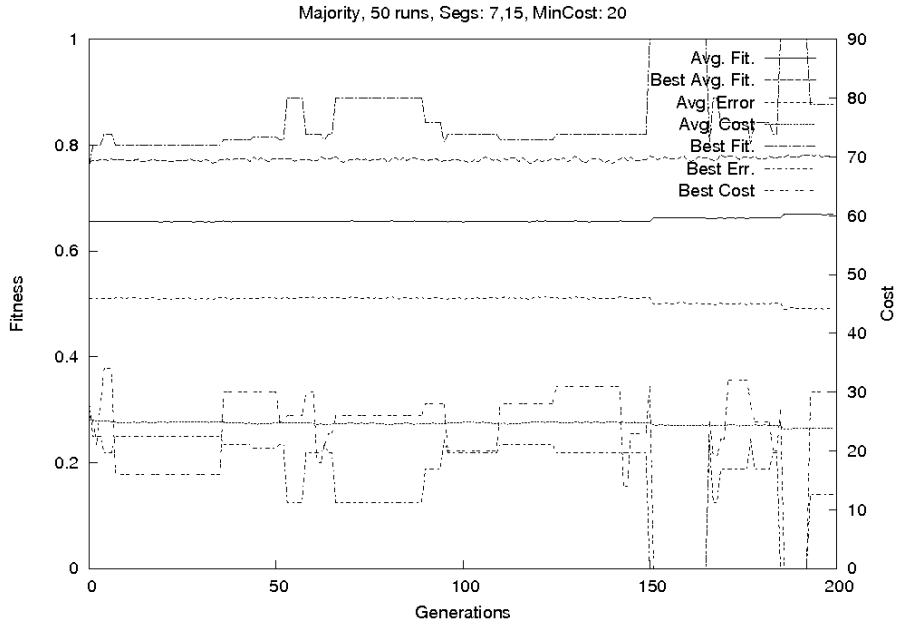


Figure 2.12: Synthesis of Majority gate using single-qubit ME with partial input gate set S_p and $t_{min} = 7$, $t_{max} = 15$.

desired minimal cost.

From an experimental point of view, the size of the problem space given by the t_{min}, t_{max} parameters was observed to be inversely proportional to the number of solutions found. This can be seen for instance in Figure 2.7 which shows that for a given cost, the GA will generates circuit solutions when the relations between the desired minimal cost and the bounds t_{min}, t_{max} are both such that for a minimal cost c_m there is at least one global solution (circuit) with a cost $c \geq c_m$ and its size is $t_{min} \geq t \geq t_{max}$.

In the presented experiment, the evolutionary search using $(t_{min}, t_{max}) = (5, 11)$ (Figure 2.11) and $(t_{min}, t_{max}) = (13, 29)$ (Figure 2.13) respectively under- and over-estimates the appropriate circuit size, while the search using $(t_{min}, t_{max}) = (7, 15)$ (Figure 2.11) was successful.

The goal of manipulating the t_{min}, t_{max} parameters was to restrict the problem space size in order to make the search computationally tractable. Another parameter that was used is the size of the circuit; i.e. the maximum number of parallel segments in an Individual. For instance, the GA starts with the minimal number of segments set up to 2 and the maximal number of segments set up to 5. Note that in order to successfully design a gate the desired minimal cost must be below the optimum minimal cost.

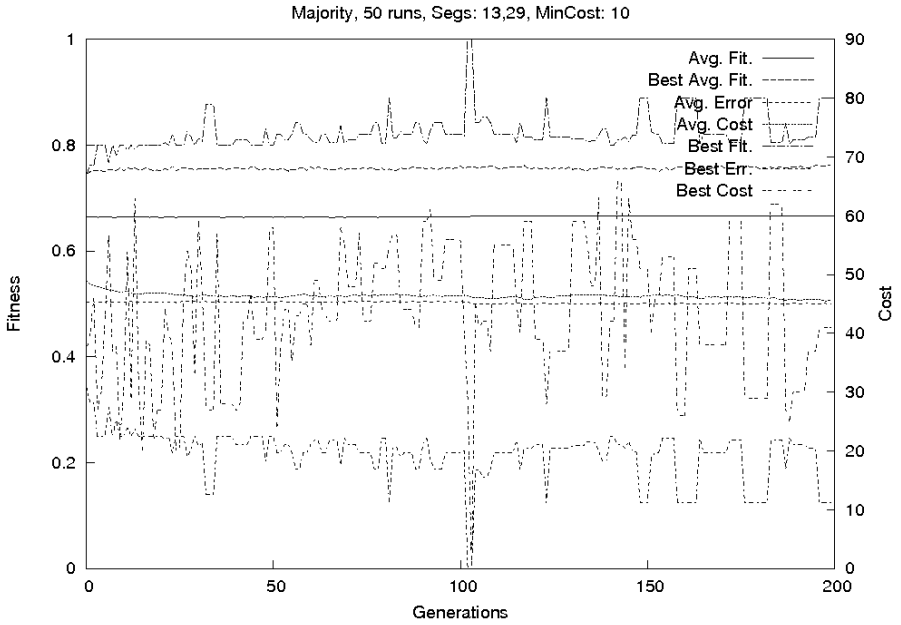


Figure 2.13: Synthesis of Majority gate using single-qubit ME with partial input gate set S_p and $t_{min} = 13$, $t_{max} = 23$.

The range of the size of the acceptable circuit size (defined by $l = t_{max} - t_{min}$) was also considered as a factor during the evolutionary process. This is because when l is small ($l < \epsilon$), then many circuits have to be repaired after crossower, thus information is lost too early in the evolutionary process (any repair mechanism naturally entails information loss). On the other hand, a large value of parameter l has an opposite effect and of searching circuits with larger cost, despite that the solution may be much less expensive.

The importance of the t_{min}, t_{max} parameters (as well as of l) on the synthesis process is clearly observable during the evolutionary search. When looking for a circuit of a known size n (and assuming this size is minimal) it is required to allow the GA to search spaces with chromosome length above and below the user specified circuit size. From previous work in this area [LPMP02,LPG$^+$04,LP08] it was shown that a messy algorithm (with random size of the individual circuit (gene)) was not successful. In the here studied evolutionary synthesis method, the size of the circuit is much more controlled, however it is required that the initial estimate of the maximal size of the circuit appropriately over-estimates the minimal size (or the expected minimal size) of the circuit. This is important because this assumption allows to design given Boolean functions with various costs allowing different results close to minimum. Thus, the search space has to be restricted around the expected minimum as close as possible.

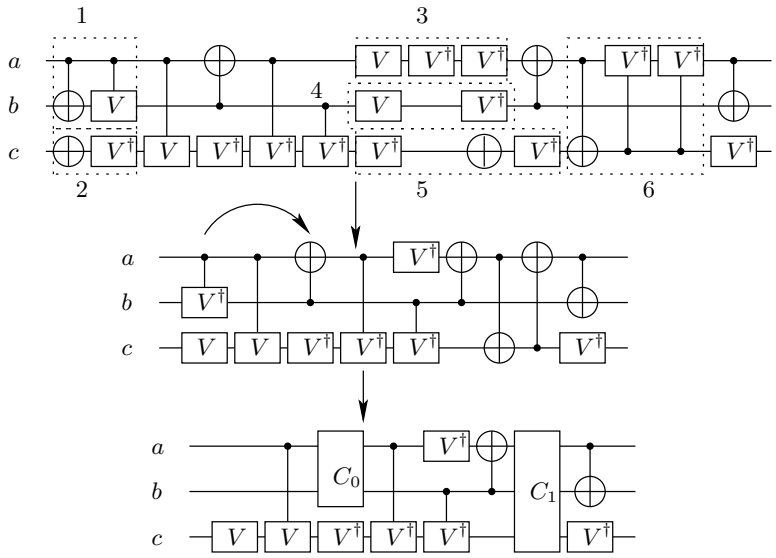


Figure 2.14: Example Majority circuit found by the GA. The Figure also shows steps of the minimization process. Observe that the aggregate gate C_0 is created by moving the $CV^\dagger$ gate near the CNOT gate (left-to-right arrow in the middle circuit).

Similarly to the synthesis of Toffoli or Fredkin gates, some circuits obtained during the synthesis of Majority gate have the output qubit deterministic despite the fact that some other qubits can be still in a probabilistic state. One of such circuits is shown in Figure 2.14. Observe that again the circuit is minimized by concatenating neighbor gates defined on the same qubits. For instance on the bottom qubit, the sequence of single qubit operations $[V^\dagger][NOT][V^\dagger]$ reduces to Identity and thus is removed.

Figure 2.15 shows a successful search for the Majority gate. Observe that similarly to the synthesis of Fredkin and Toffoli the evolutionary process finds the solution in one step. This means that the fitness function during the search reaches some local optimum and then in one step jumps to value one.

2.4 Discussion of the Results of the Evolutionary Quantum Logic Synthesis using EE evaluation

2.4.1 Toffoli Gate

Figure 2.16 represents typical circuits that have been obtained during the synthesis process using the S_f and the S_p input gate-sets.

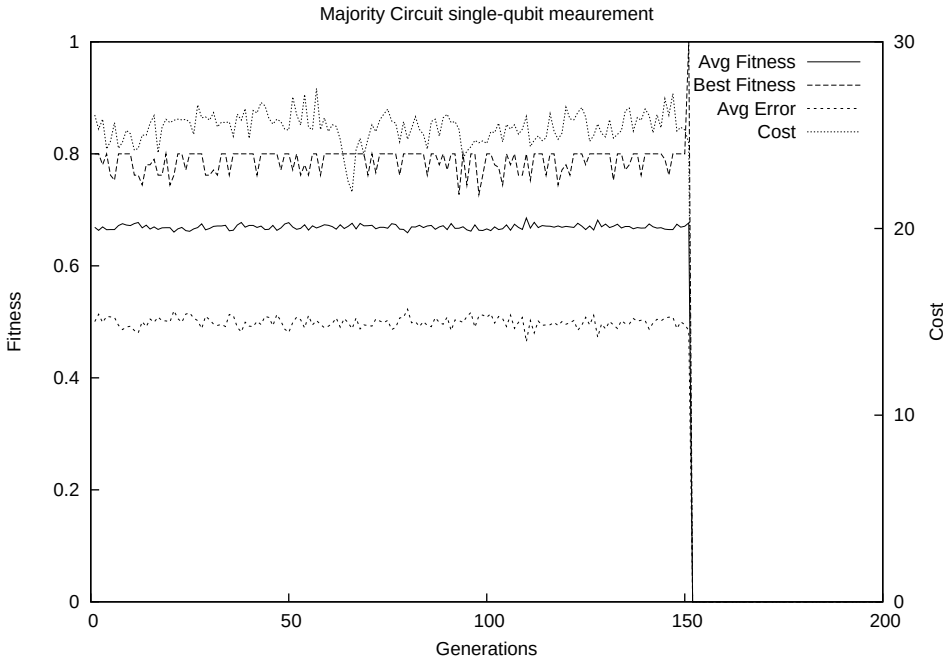


Figure 2.15: Example of successful synthesis of the Majority gate using a partial input gate-set S_p and a single-qubit ME method.

The first circuit (2.16a) is the result of a run where fitness from equation 1.14 was used together with a complete starting set of gates. As can be seen, this result is very expensive because it includes three Fredkin gates and four Feynman gates. This circuit can be simplified by removing two of the three subsequent identical Feynman gates based on the fact that Feynman gate is its own inverse.

The next one, Figure 2.16b, is a result of the same setting as with the previous one but the starting set was biased. The available gates in this biased set were only those with the number of inputs smaller or equal to 2 qubits. Similarly to the previous circuit, the group of gates in dash-squares can be concatenated in order to reduce the cost of the circuit. Moreover two pairs of consecutive identical Feynman gates (in dotted groups) can be removed. Thus, the second from left dotted group is removed entirely and the first from right dotted group is replaced by a single Feynman gate. The rightmost Feynman gate can be moved before the last Controlled-V gate and added to the dotted rectangle. Although this circuit is longer than the first one, its cost is the same. It is composed only from 2-qubit and 1-qubit gates. The first circuit has cost 67 (5*2 + 19*3) after minimization and the second circuit after minimization has the cost of 45 (9*5).

The two circuits on the bottom of Figure 2.16 were found using the improved fitness

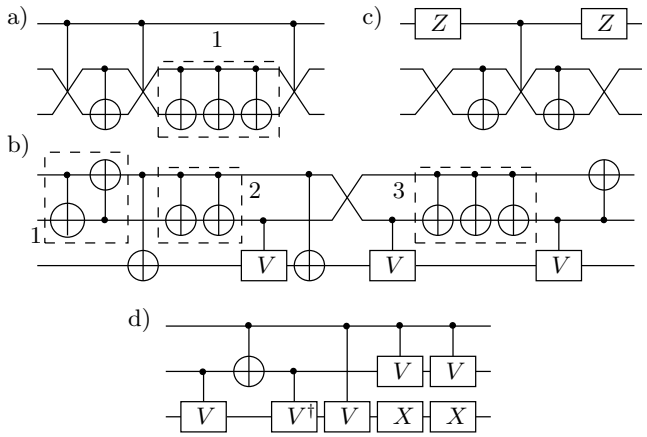


Figure 2.16: Various realizations of Toffoli gate obtained from GA before their optimizing transformations were applied. (a) - three CNOT gates (the dashed box 1) are collapsed into a single one, (b) - the two Z gates on the topmost wire can be removed as they do not change the observable of the circuit (after measurement), (c) - the dashed box 1 is collapsed to a single two-qubit gate, frame 2 is removed as $[CNOT] \times [CNOT] = [I]$ and frame 3 is collapsed to a single CNOT gate, (d) - this circuit is minimized to the well known minimal Toffoli gate by removing the two NOT gates and transforming two CV gates into a single CNOT.

function from equation 1.14 and 1.16. The circuits from Figures 2.16a, b and c were found using the complete starting set S_f. The circuit from Figure 2.16d was found using the biased set S_p. The major improvement using the equation 1.16 as the fitness function is that in a well delimited problem space this fitness function can speed up the search and possibly find the optimal gate. However as is discussed in Section 2.5 this improvement is limited to the search for less expensive gates as in general it results in longer runs of the GA. Observe that by flipping over two Feynman gates from Figure 2.16c and removing the swap gates a solution with one Fredkin and two Feynman gates is generated, which is close to the known realization of Toffoli gate from Fredkin gate. The only difference are two Pauli-Z rotation gates, which can be removed, as the analysis shows.

The results from Figure 2.16 have been obtained using the EE error model method. However, some of the exact benchmarks have also been synthesized using the ME error model evaluation method. In the case of the Toffoli gate, the gate can be specified by observed values only on desired qubits; a single output function.

Interestingly, using the EE error model method (as well as using the ME error model) method, our software reinvented also the famous circuit of Smolin [SD96], since the sequence of two Pauli-X (NOT gates) can be removed, and the sequence

of two Controlled-V is equivalent to Feynman gate (Figure 2.16d). After these transformations, our gate is composed from the same basic quantum primitives as the popular (and used in most designs and papers to realize quantum algorithms) Smolin's solution [SD96], but in a different order. The comparison of the unitary matrices shows however that the circuit invented by our GA and the circuit invented by Smolin are equivalent. These examples show that not only can our software "reinvent" the realization of the known gates but it can also create similar minimum cost realizations of other gates, as will be presented in chapter **??**. We were not able to find a better solution to Toffoli and Fredkin gates from quantum primitives, because perhaps such realizations do not exist with the primitive used by us. (the hybrid gates with smaller costs have been found by [LBA+08].

2.4.2 Fredkin Gate

Figure 2.17 presents some of many realizations of Fredkin gate found using the EE error-model evaluation, the fitness function from eq. 1.14 and the full set of gates S_f. Figure 2.17a and Figure 2.17b show the result when the full input gate-set S_f was used. Figure 2.17c and Figure 2.17d present results when we used the input gate-set defined by $S_f - [SWAP]$.

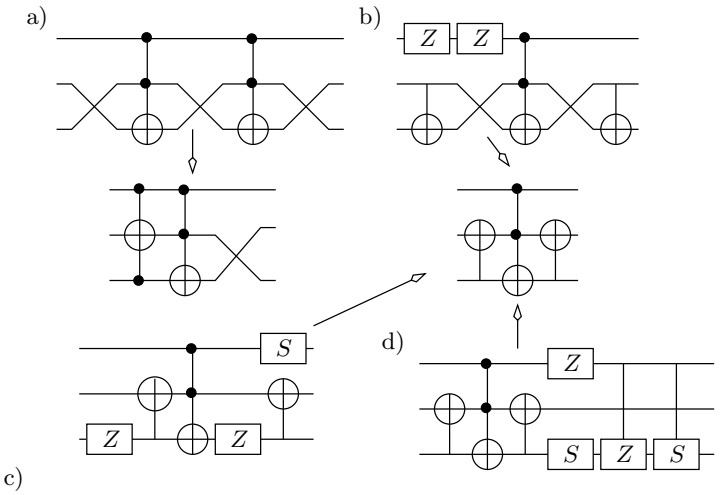


Figure 2.17: Results of synthesis of Fredkin gate. *a*) this circuit is a result of application of fitness from eq. 1.14 and a complete set, *b*) this circuit is a result of fitness from 1.16 and a complete set. Figures c) and d) represent results obtained using the same settings but without using the SWAP gate in the input gate-set.

Again, observe that most of the obtained circuits can be minimized to the well known

optimal Fredkin solution. The consecutive Pauli-Z gates can be canceled (Figure 2.17b) since this gate is its own inverse (known as a standard local transformation). Next, in both Figure 2.17a and Figure 2.17b the swap gates can be removed by manipulating the circuit: changing the *ab* controlled Toffoli to *ac* controlled Toffoli (Figure 2.17a), changing to two Feynman gates, by flipping them upside down (CNOT gates to NOTC (bottom-up CNOT)), removing the SWAP gates. This leads to the known (minimal) realization of Fredkin gate that uses one Toffoli and two Feynman gates (Figure 2.17b). Also observe that the gates from Figures 2.17c and Figure 2.17d are easily transformed into the minimal solution, by simply removing the single qubit gates $[Z]$ and $[S]$. This removal does not have any effect on the resulting function because the $[Z]$ and $[S]$ gates manipulate only the phase of the quantum state.

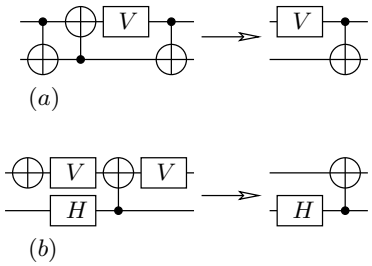


Figure 2.18: Results of Synthesis and Minimization of the Entanglement circuit for 2 qubits.

One of the observations of the obtained results is the mutual presence of Fredkin and Toffoli gates in the circuits representing Toffoli and Fredkin respectively. In fact, using one of the two gates does experimentally guarantee finding a result, while not providing such a universal gate highly reduces the chances of finding the correct gate. As can be seen in the results of synthesis using the complete starting set, the Fredkin gate is present in all Toffoli implementations and vice versa. Without a universal gate in the input gate-set the GA has troubles during the evolutionary process to form and preserve such segments on multiple qubits. It can be concluded that providing more complex universal gates allows the GA to overcome a local functional minimum. This principle should be used to generate quantum algorithms using large blocks such as circuits for QFFT [].

2.4.3 Entanglement Circuit

Another set of benchmarks were the entanglement circuits for two, three and four qubits. Figure 2.18 shows the results of synthesis of the two-qubit entanglement

circuit. Observe that both presented examples can be minimized to the well known entanglement circuits. Also, as described in chapter 1, two different implementations of the entanglement circuit can be observed for two and three qubits: using the well known Hadamard gate and using the V gate (Figure 2.18 and 2.19).

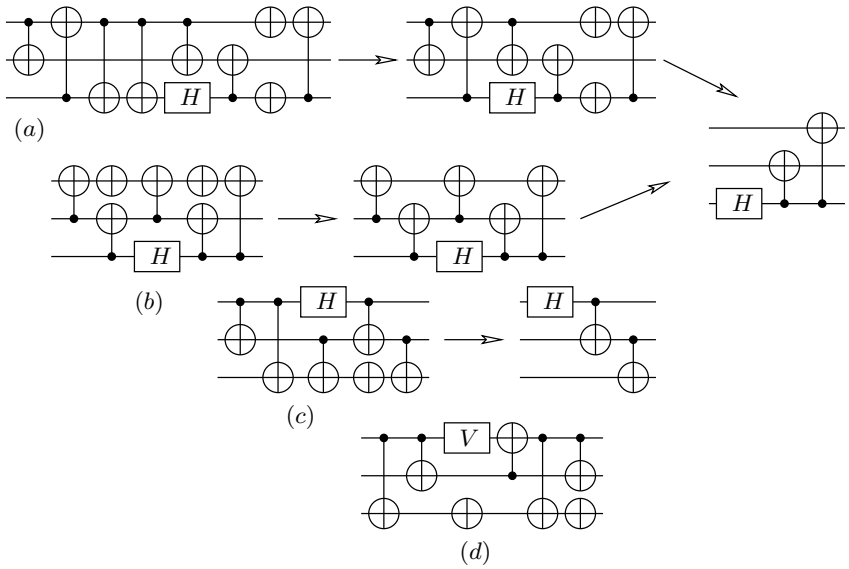


Figure 2.19: Results of Synthesis and Minimization of the Entanglement circuit for 3 qubits.(a) and (b) represent circuits that simplify to the circuit on the right, (c) minimizes to circuit representing the same logic function but minimizing to a different circuit (on the right), (d) another realization of the 3 qubit entanglement circuit

Figure 2.19d shows an entanglement circuit for three qubits realized with only V and CNOT gates (the NOT gates can be removed without changing the overall circuit output). It is interesting to observe that this circuit (as well as the circuit in Figure 2.18a) generates various entangled states. These states will be indistinguishable under measurement from the Hadamard based circuit generated entangled states.

Observe, that most of the gates synthesized here can be minimized to the well known entanglement circuit not because the presented results are directly sub-circuits of such gates, but rather because most of the component gates in the presented circuits can be removed. Such action will modify the possible entangled output states but will preserve the entanglement.

Figure 2.20 illustrates the four-qubit entanglement circuits together with the minimization process. Similarly to the case of two and three qubits, half of the circuit can be thrown away in the case when the goal is to obtain an arbitrary entanglement circuit. Observe that under such assumption (when the only goal is to obtain any

type of entanglement), all the circuits from Figure 2.20 have the same cost when minimized.

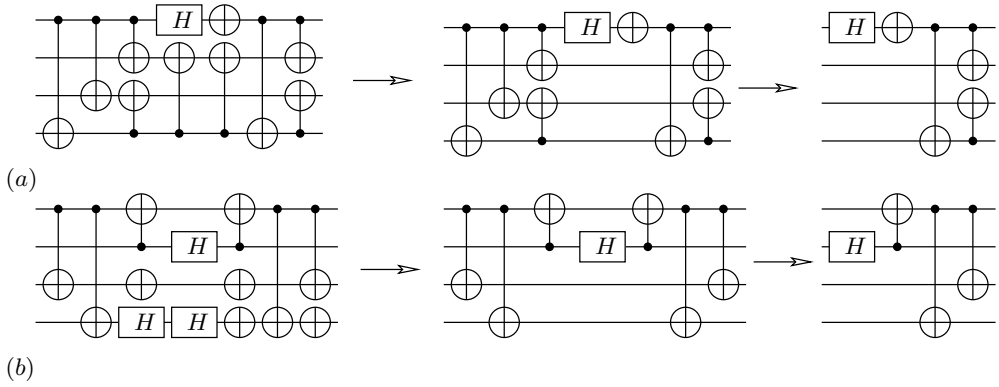
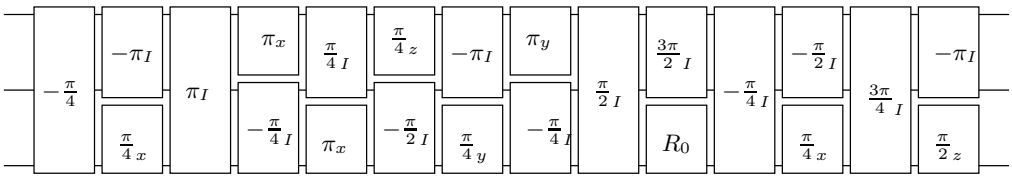


Figure 2.20: Results of Synthesis and Minimization of the Entanglement circuit for 4 qubits. The GA-synthesized circuits are on the left and on the right is the same circuit after inimization that was performed manually. Observe that circuitf from (a) and (b) despite having a different intial structure can be minimized in to the same final circuit.

2.4.4 The level of Synthesis and the Pulse-based synthesis



$$R_0 = \tfrac{\pi}{2}_y \cdot -2\pi_x$$

Figure 2.21: Example of sub-optimal approximate Toffoli using the S_{lr} input gate set.

In the case of the pulse-based synthesis (input set S_{lr}) the search was more difficult both in the case of the Toffoli gate but also for all other benchmarks. In fact as will be seen, the synthesis using the unitary pulses of arbitrary (user limited) angles was mostly unsuccessful.

Figures 2.21 and 2.22 represent some of the results using the GA. The rotation was parameterized by the set $\{\pi, -\pi, \frac{\pi}{2}, -\frac{\pi}{2}, \frac{\pi}{4}, -\frac{\pi}{4}\}$. One of the particular observations

on the pulse-based synthesis is that the algorithm found during these experiments mainly only approximate gates in the alloted time.

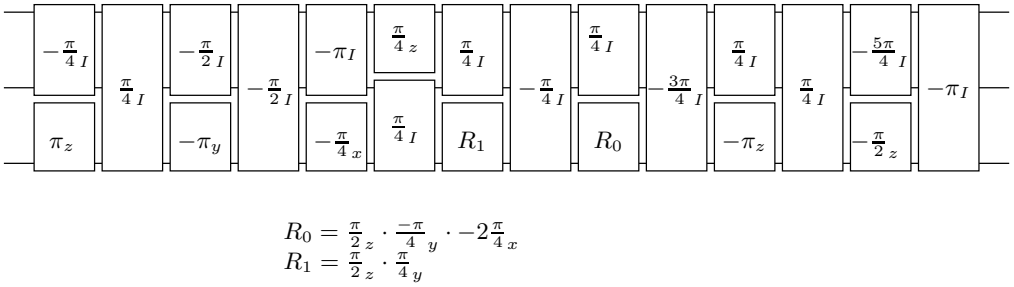


Figure 2.22: Example of sub-optimal approximate Toffoli using the S_{lr} input gate set.

The lack of success when synthesizing using the angle-parameterized unitary rotations can be attributed to the following factors:

- the GA has troubles to keep many small unitary rotations together in order to preserve higher level gates such as Toffoli or Fredkin
- the input-gate set grows too fast in size and thus to obtain a desired circuit the GA requires both more time and more computation ressources

2.4.5 The CZ Synthesis

One of the most interesting result was generated by the GA when it was used to search quantum space using the input gate set $\{I, H, Z, CZ, CH\}$. The GA was searching for the same logic primitives as discussed previously, i.e. Toffoli, Fredkin. The obtianed gates are unqie in the sense that they are not comonly seen in the quantum logic synthesis approaches and they provide the lowest cost of both the Toffoli and Fredkin reversible gates.

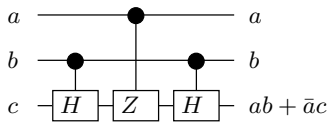


Figure 2.23: Toffoli gate with cost of 3.

The most interesting feature of the discovered Toffoli and of the Fredkin gates is the fact that both have the same cost after minimization. The Toffoli gate is shown

in Figure 2.23 and the Fredkin gate - obtained by adding two auxilliary CNOT gates is shown in Figure 2.24.

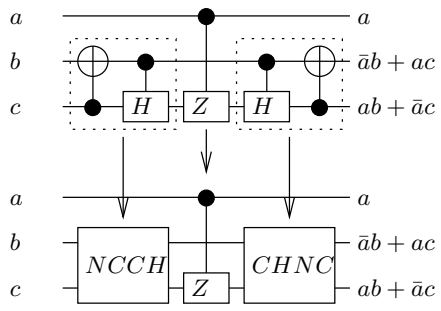


Figure 2.24: Fredkin gate with cost of 3.

Observe that this gate was originally synthesized without the CNOT so the original design of the Fredking gate has more gates. It is shown in Figure 2.25. This result is very motivating for the EQLS because it is for the first time that a circuit with a better cost than previously designed by humans wa obtianed.

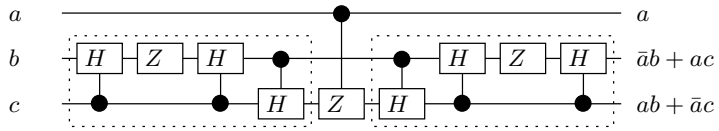


Figure 2.25: Fredkin gate fully expanded.

2.5 Discussion of the results of the Evolutionary Synthesis

2.5.1 Results Comparison

Table 2.4 presents a summary of the results in QC search for the benchmarks used in this chapter. The first column is the name of the function searched, the second column shows if the software solution was found using either the non-optimized fitness function (eq. 1.12), or the optimized fitness function (eq. 1.16). The third column shows, for each circuit, the solution times in terms of numbers of generations necessary to arrive at a solution. Each run was stopped after it reached the maximal generation count in the cases for which no solution was found previously in the run.

Table 2.4: Comparison of various fitness functions for all illustrated benchmark functions

Problem Name	*Solution* [Fitness (eq. 1.12)/(eq. 1.16)]	*# Generations* [Fitness (eq. 1.12)/(eq. 1.16)]
Toffoli	YES/YES	$< 200/ < 500$
Fredkin	YES/YES	$< 200/ < 750$
Majority	YES/NO	$< 100/ < 300$
Miller	YES/NO	$< 400/ < 800$
Entangle2	YES/YES	$< 100/ < 100$
Entangle3	YES/YES	$< 100/ < 150$
Entangle4	YES/YES	$< 200/ < 400$

As can be seen in five cases out of the seven test benchmark problems (Toffoli, Fredkin, Miller and Entangle4), a significant time increase was observed when using the fitness function from eq. 1.16 (Table 2.4 rows 1,2,4 and 7). Two arguments support the use of the improved fitness function (eq. 1.16). First, the solution was found in all cases and it was found even with a biased set. This argument implies that the cost function allows the GA to preserve individuals in the population that would be otherwise discarded. This allows to preserve the diversity in the population. Second, the obtained results using the fitness function (eq. 1.16) in general generated shorter circuits as well as circuits either optimal or close to the known optimal circuits.

2.5.2 Problems Encountered during the Evolutionary QLS

As is described in this chapter, the GA was able to find quickly circuits for two, three and also for four qubits, but was much slower for larger gates and circuits. The reason for this lower performance is due to following problems:

- the size of input gate-set
- the allowed size of the circuit to be searched
- the level of the element gates
- the relation between the error evaluation and the cost of the circuit

The size of the input-gate set was already discussed and analyzed in Section 2.3.2 as well as the size of the problem space was analyzed in Section 2.3.3. Next, the

following two points are discussed; the impact of the level of the synthesis on the performance of the GA in Section 2.4.4 and the relation between the error evaluation and the cost function in Section 2.5.2.1.

2.5.2.1 The Cost-Error ratio

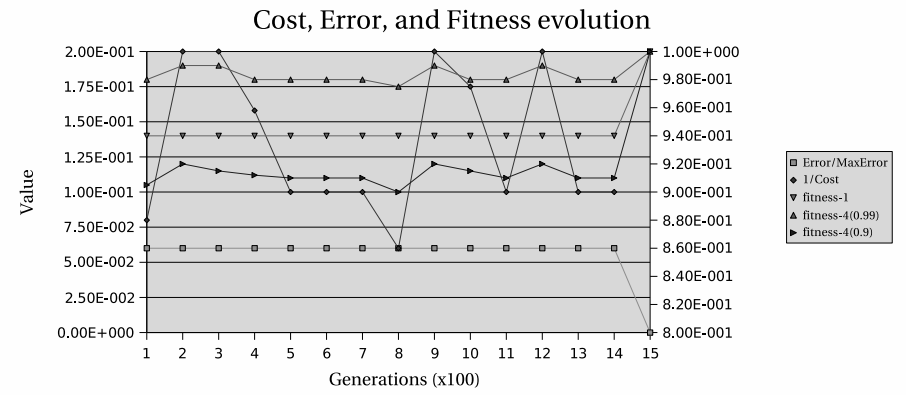


Figure 2.26: A comparative chart between two different fitness functions during the same GA run.

Figure 2.26 shows the recording of one run searching for the Fredkin gate using the fitness function from eq. 1.14. For illustration the fitness from equation 1.16 is also drawn. The figure shows the best result of each hundred generations. The first curve Error shows the evolution of the scaled error as used in the fitness function. Second curve is the cost (1/Cost) and it is to be maximized to increase the global fitness function. The larger the circuit the more the cost is reduced. The last three curves show the values for fitness function from equations 1.14 and 1.16. Fitness function from eq. 1.16 is illustrated with two curvesthe ; first with parameters $\alpha = 0.99$ and $\beta = 0.01$ and the second with $\alpha = 0.9$ and $\beta = 0.1$. For more clarity all fitness functions are mapped on the secondary y-axis on the right.

The Fitness function from eq. 1.16 with $\alpha = 0.9$ captures mainly the correctness of the circuit as well as the cost ($\beta = 0.1$). Such configuration however requires a large final jump to the correct solution (≈ 0.1). Such a fitness function can be used to minimize the size constraints but might not be the best candidate to find circuits with minimal error.

The best fitness function to capture global properties during a search for a correct circuit is the fitness function from eq. 1.16 with $\alpha = 0.99$. This is because on one hand $\alpha = 0.99$ captures the correctness of the circuit and on the other hand a $\beta = 0.01$ takes into account the size of the circuit in such a way that allows both to

search for the correct circuit and minimize the size (cost) of it in an appropriate ratio. Consequently, comparing it to the fitness function (eq. 1.14) one can observe similar result but with small oscillations in the fitness function (eq. 1.16 with $\alpha = 0.99$).

Figure 2.26 illustrates also the generally observed slow convergence of the GA. Observe that this can be seen in Figures 2.3, 2.9 and 2.15 where the results appear as a jump in the fitness value rather than as a result of a convergence of the fitness value to the value of one. The reason for this is, is the structure of the error that is always symmetric. In other words because the evolved circuit is a unitary matrix built from unitary matrices (quantum gates) any discrepancy between the target and the synthesized circuit will be always on 2^r coefficients.

The structure of the error in an unitary matrix also means that when synthesizing using only permutative component gates, the magnitude of the error will always be at least $\frac{2}{2^n}$ which results in the fact that the GA overall fitness value is very often stuck on a local maximum before in one step reaching the fitness value of one and error value of 0. In the case when arbitrary quantum gates are used for synthesis the magnitude of the error can be arbitrarily small. However because in the present case we use only a set of selected quantum gates the generated error is again bounded by the used gates. Thus confirming that the GA reaches the correct solution (fitness = 1) after a jump from a local optimal fitness value.

2.6 Conclusion of the Evolutionary QLS

This chapter presented a new approach to quantum logic synthesis using a a fitness/error function and a circuit cost. Similarly to the previous work in this area [], our algorithm was able to successfully use the selected cost policy and synthesis constraints to generate new circuits implementing permutative reversible circuits. However we were also able to obtain novel circuits preiously unknown to exists. Such example can be seen in Figure **??** showing a Fredkin gate having the same cost as a Toffoli gate. This is quite a novel result and it illustrates the power the evolutionary computation has. Indeed as it was shown in the search using the NMR pulses, to find a result is quite difficult because the amount of input gates is too large. On the other hand with a small set of gates algorithmic approach will outperform GA in both speed and optimal solutions. Thus the discovery of novel gates and novel minimal solutions of universal reversible gates is a task where the GA is the great match.

Another interesting result of the search presented here is that despite the fact that using the NMR pulses the synthesis was not very successful for three-qubit circuits the joint approach that used pseudo-binary and unitary pulses allowed to find out

new circuits, otherwise not possible to discover on two qubits. This was not achieved in previous researches of their authors.

The Figure **??** and **??** shows the implementation of the CNOT and CV gates using NMR pulses, that have been designed by our GA algorithm. Because the I_{zz} does not commute, the circuit is split in half. Thus, two sides of the quantum circuit can be distinguished; but this observation is only possible when looking into the sub-permutative-logic level. Similar observation was made with the synthesis model using only the $CV/CV^{\dagger}$ and $CNOT$ gates. The usefulness of this anti-symmetric property is the fact that circuits with a given structure can have different function when the order of the component gates is inverted.

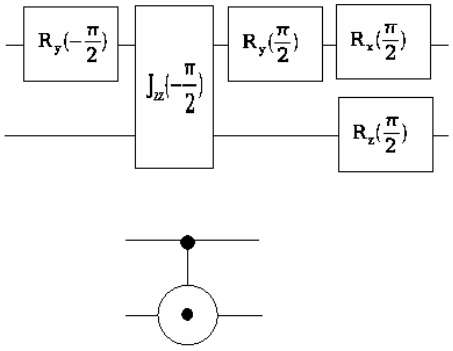


Figure 2.27: Inversely constructed CNOT generates CE

To conclude this Chapter the following observations can be made:

- The cost-error ratio is a difficult problem that can be partially solved using evolutionary QLS
- The QLS as evolutionary process is sensitive to multiple factors such as the size of the input-gate set, the width of the synthesized circuit, the level of synthesis, the ratio between user specified minimal cost and the desired circuit size. Together, these problems makes the evolutionary QLS a hard problem
- The evolutionary QLS allowed us to rediscover already known realizations of quantum circuits (Fredkin, Toffoli) as well as invent novel interesting gates such as Entanglement, Majority, etc.
- The ME and EE methods of evaluations are quite useful insights into the QLS from an algorithmic point of view and allowed us to discover novel, partial realization of universal quantum gates.

Bibliography

[BBC+95] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and Weinfurter H. Elementary gates for quantum computation. *Physical Review A*, 52:3457–3467, 1995.

[EvKK95] A. E. Eiben, C. H. M. van Kemenade, and J. N. Kok. Orgy in the computer: multi-parent reproduction in genetic algorithms. In *94*, page 10. Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, 31 1995.

[GKD89] D.E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis and first results. *Complex Systems*, 3:493–530, 1989.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, MA, 1989.

[Hol75] J.H. Holland. *Adaptation in Natural and Artificial systems*. The University of Michigan Press, Ann Arbor, 1975.

[HSY+06] W. N. N. Hung, X. Song, G. Yang, J. Yang, and M Perkowski. Optimal synthesis of multiple output boolean functions using a set of quantum gates by symbolic reachability analysis. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and systems*, 25(9):1652–1663, 2006.

[LBA+08] B. P. Lanyon, M. Barbieri, M. P. Almeida, T. Jennewein, T. C. Ralph, K. J. Resch, G. J. Pryde, J. L. O'Brien, A. Gilchrist, and A. G. White. Quantum computing using shortcuts through higher dimensions, April 2008.

[Lei04] A. Leier. *Evolution of Quantum Algorithms Using Genetic Programming*. PhD thesis, University of Dortmund, 2004.

[LKBP06] S. Lee, S.J. Kim, J. Biamonte, and M. Perkowski. The cost of quantum gate primitives. *Journal of Multiple-Valued Logic and Soft Computing*, 12(5-6):561–574, 2006.

[LP02] M. Lukac and M. Perkowski. Evolving quantum circuit using genetic algorithm. In *Proceedings of the 2002 NASA/DoD Conference on Evolvable hardware*, pages 177–185, 2002.

[LP05a] M. Lukac and M. Perkowski. Combining evolutionary and exhaustive search to find the least expensive quantum circuits. In *Proceedings of ULSI symposium*, 2005.

[LP05b] M. Lukac and M. Perkowski. Using exhaustive search for the discovery of a new family of optimum universal permutative binary quantum gates. In *Proceedings of International Workshop on Logic & Synthesis, Poster Session*, 2005.

[LP08] M. Lukac and M. Perkowski. Evolutionary approach to quantum symbolic logic synthesis. In *IEEE Congress on Computational Intelligence (WCCI)*, pages 3374–3380, 2008.

[LPG+03] M. Lukac, M. Perkowski, H. Goi, M. Pivtoraiko, C. H. Yu, K. Chung, H. Jee, B.-G. Kim, and Y.-D. Kim. Evolutionary approach to quantum reversible circuit synthesis. *Artif. Intell. Review.*, 20(3-4):361–417, 2003.

[LPG+04] M. Lukac, M. Perkowski, H. Goi, M. Pivtoraiko, C. H. Yu, K. Chung, H. Jee, B-G. Kim, and Y-D. Kim. Evolutionary approach to quantum and reversible circuits synthesis. In *Artificial Intelligence in Logic Design*, pages 201 – 257. Kluwer Academic Publisher, 2004.

[LPK10] M. Lukac, M. Perkowski, and M. Kameyama. Evolutionary quantum logic synthesis of boolean reversible logic circuits embedded in ternary quantum space using structural restrictions. In *Proceedings of the WCCI 2010*, 2010.

[LPMP02] M. Lukac, M. Pivtoraiko, A. Mishchenko, and M. Perkowski. Automated synthesis of generalized reversible cascades using genetic algorithms. In *Proceedings of Fifth Intern. Workshop on Boolean Problems*, pages 33–45, 2002.

[Luk09] M. Lukac. *Quantum Logic Synthesis and Inductive Machine Learning, Ph.D. dissertation.* PhD thesis, Portland State University, 2009.

[MCS04] P. Massey, J.A. Clark, and S. Stepney. Evolving quantum circuits and programs through genetic programming. In *Proceedings of the Genetic and Evolutionary Computation conference (GECCO)*, pages 569–580, 2004.

[MCS05] P. Massey, J.A. Clark, and S. Stepney. Evolving of a human-competitive quantum fourier transform algorithm using genetic programming. In *Proceedings of the Genetic and Evolutionary Computation conference (GECCO)*, pages 1657–1664, 2005.

[MDM05] D. Maslov, G. W. Dueck, and D. M. Miller. Synthesis of Fredkin-Toffoli reversible networks. *IEEE Transactions on VLSI*, 13(6):765–769, 2005.

[Mil02] D. M. Miller. Spectral and two-place decomposition techniques in reversible logic. In *Proc. Midwest Symposium on Circuits and Systems, on CD-ROM*, August 2002.

[MMD06] D. M. Miller, D. Maslov, and G. W. Dueck. Synthesis of quantum multiple-valued circuits. *Journal of Multiple-Valued Logic and Soft Computing*, 12(5-6):431–450, 2006.

[MT78] S. Martello and P. Toth. Algorithm for the solution of the 0-1 single knapsack problem. *Computing*, 21:81–86, 1978.

[Rai96] G. Raidl. *Evolutionary Algorithms*. University of Technology, Wien, 1996.

[Rub01] B.I.P. Rubinstein. Evolving quantum circuits using genetic programming. In *Congress on Evolutionary Computation (CEC2001)*, pages 114–121, 2001.

[SBS08] R. Stadelhofer, W. Banzhaf, and D. Suter. Evolving blackbox quantum algorithms using genetic programming. *Artif. Intell. Eng. Des. Anal. Manuf.*, 22:285–297, 2008.

[SD96] J. Smolin and D. P. DiVincenzo. Five two-qubit gates are sufficient to implement the quantum fredkin gate. *Physical Review A*, 53(4):2855–2856, 1996.

[Spe04] L. Spector. *Automatic Quantum Computer Programming: A Genetic Programming Approach*. Kluwer Academic Publishers, 2004.

[WGMD09] R. Wille, D. Große, D.M. Miller, and R. Dreschler. Equivalence checking of reversible circuits. In *Proceedings of the ISMVL*, 2009.

[Yab00] T. Yabuki. Genetic algorithms for quantum circuit design – evolving a simpler teleportation circuit –. In *In Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, 2000.

[Yen05] Ö. Yeniay. Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and computational Applications*, 10(1):45–56, 2005.

[YSPH05] G. Yang, X. Song, M. Perkowski, and W. N. N. Hung. The power of large pulse-optimized quantum libraries: Every 3-qubit reversible function can be realized with at most four levels. In *Proceedings of Proc. of IWLS*, 2005.

[ZLSD02] J. Zhang, Z. Lu, L. Shan, and Z. Deng. Realization of generalized quantum searching using nuclear magnetic resonance. *Physical Review A*, 65:034301.1–034301.4, March 2002.