TEMPLATES FOR POSITIVE AND NEGATIVE CONTROL TOFFOLI NETWORKS

MD ZAMILUR RAHMAN
Bachelor of Science, Jahangirnagar University, 2005
Master of Science, Jahangirnagar University, 2007

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

TEMPLATES FOR POSITIVE AND NEGATIVE CONTROL TOFFOLI NETWORKS

MD ZAMILUR RAHMAN

Date of Defense: December 16, 2014

Dr. Jacqueline E. Rice
Supervisor   Associate Professor   Ph.D.

Dr. Stephen Wismath
Committee Member   Professor   Ph.D.

Dr. Hua Li
Committee Member   Associate Professor   Ph.D.

Dr. Howard Cheng
Chair, Thesis Examination Committee Associate Professor   Ph.D.

Dedication

To my beloved parents

Abstract

Circuit realizations obtained from existing logic synthesis approaches may not be optimal and thus one commonly applies post-synthesis optimization techniques to get better realization of the circuits. This thesis proposes two new templates (templates 4 and 7) for positive and negative control Toffoli gates as well as proposing algorithms for post synthesis optimization of reversible positive and negative control Toffoli networks by utilizing the set of templates. When applying the templates to circuits generated by the improved shared cube synthesis approach [23] a reduction in quantum cost was achieved for 86 of the 110 circuits. On average a 21.34% reduction in quantum cost was achieved, and in some cases up to 53.58% reduction was obtained.

Acknowledgments

At first, I would like to take the opportunity to express my gratitude and appreciation to my supervisor, Dr. Jacqueline E. Rice for her continuous support, encouragement, and invaluable guidance throughout my M.Sc. program. I would also like to thank my committee members Dr. Stephen Wismath and Dr. Hua Li for their time, constructive comments, and suggestions.

Special thanks to my colleagues in the research group and in the lab for their help and inspiration during the program.

I am also thankful to Natural Sciences and Engineering Research Council of Canada (NSERC) for funding this research.

Finally I would like to thank my parents, brother, wife, and relatives for their constant support and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Power dissipation and heat generation are serious problems in today's traditional circuit technologies. According to R. Landauer's observation in 1961, the amount of energy dissipated for each lost bit of information is $KTln2$ where K is the Boltzmann's constant $(1.3807 \times 10^{-23} JK^{-1})$ and T is the Temperature [8]. This is a significant amount of energy for millions of operations. In [1], Bennett showed that in order to not dissipate energy the system must be logically reversible. Reversible circuits do not erase any information when operations are performed. In reversible circuits, all operations are performed in a bijective manner. Thus fan-out and feedback operations are not allowed in reversible circuits. Such features of reversible circuits prevent the use of existing algorithms and tools for circuit synthesis and optimization. Reversible logic synthesis is the process of generating a circuit from a given reversible specification. Reversible circuits have applications in fields such as optical computing [3] and nanotechnology [18]. Research on reversible logic synthesis has attracted much attention after the discovery of powerful quantum algorithms in the mid 1990s [25]. Quantum circuits are inherently reversible. Quantum gates are represented by unitary matrices which may include complex elements. The most commonly used quantum gate library includes NOT, CNOT, Controlled-V, and Controlled-$V^{\dagger}$ gates. Interested readers can refer to [25] for a detailed discussion of quantum computing. As a result, reversible logic is being considered as an alternative to conventional logic. Instead of conventional logic gates reversible gates like Toffoli gates, Fredkin gates, and Peres gates are used in re-

versible circuits. Different gate libraries developed using reversible gates are listed in [37].

The main focus of this thesis is to optimize the reversible circuit by applying templates. We propose two new templates for positive and negative control Toffoli gates (templates 4 and 7). Template 4 can be applied to two $\geq$ 3-bit Toffoli gates with controls on the same lines while template 7 can be applied to two different size $\geq$ 3-bit Toffoli gates. We propose algorithms to optimize reversible circuits utilizing the set of templates.

1.2 Organization of the Thesis

The remainder of the thesis is organized as follows.

The necessary background material of this thesis is presented in chapter 2. We present various reversible gates, gate libraries, and cost metrics for evaluating reversible circuits.

Chapter 3 begins with the overview of various logic synthesis approaches. This chapter also describes several post-synthesis optimization approaches with examples.

In chapter 4, we describe the proposed set of templates including the basic algorithm with an example. Experimental results are also tabulated with discussions.

Next in chapter 5 we discuss the basic template matching technique along with use of the moving rule and illustrate the approach step-by-step with an example. Another improved algorithm with experimental evaluations is described in a later section. Part of this chapter has been published in [28].

Chapter 6 concludes the thesis and provides direction for possible future work.

Chapter 2

Background

2.1 Boolean Logic Functions

A Boolean function $f(B)$ is a function of the form $f : \mathbb{B}^n \to \mathbb{B}^m$, where $B = \{0,1\}$ and $n, m \in \mathbb{N}$. The input vectors for the function can take 2^n possible values, i.e. all possible values from 0 to $2^n - 1$. The truth table is the simplest way to represent a Boolean function. For example, Table 2.1 shows the truth table of the full adder function. The function has 3-inputs (c_{in}, x, y) and 2-outputs (c_{out} and sum).

Table 2.1: Truth table of a full adder function

c_{in}	x	y	c_{out}	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

2.2 Traditional Logic

In traditional logic the number of inputs to a circuit may be not equal to the number of outputs. In many cases the number of inputs is greater than the number of outputs or vice-versa. Consider the truth table of a full adder function is given in Table 2.1, where the function has three inputs (i.e. c_{in}, x, y) and two outputs called sum (sum) and carry (c_{out}).

AND, OR, NOT, EX-OR, EX-NOR, NAND, and NOR are used as standard gates in tradi-tional logic. Among all the gates, NAND and NOR gates are universal gates. This means that it is possible to implement any logical function using NAND (alternatively NOR) gates alone.

2.3   Reversible Logic

In reversible logic the number of inputs and outputs are equal. In other words, a re-versible logic function is bijective (i.e. one-to-one and onto). If a gate has n-inputs, then it has n-outputs. A 3-input (a, b, c) and 3-output (a', b', c') reversible function is given in Table 2.2. There is a one-to-one correspondence between its input and output vectors. To keep the number of inputs and outputs equal, introducing constant inputs and garbage outputs is a solution. An irreversible function can be embedded into a reversible function by adding constant inputs and garbage outputs. However, the constant inputs must have certain values to realize the functionality in the outputs and the garbage outputs have no impact on the computation. The minimum number of garbage outputs required to convert an irreversible function to a reversible function is $log_2(q)$, where q is the number of times that an output pattern is repeated in the truth table [12]. As shown in the truth table of the irreversible full adder function in Table 2.1, the output 01 and 10 has the most number of occurrences and is repeated three times. Thus, at least $\lceil log_2(3) \rceil = 2$ garbage outputs and one constant input are required to make the full adder function reversible. The truth table of the reversible full adder function is shown in Table 2.2 where g_i is the constant input and g_{o1} and g_{o2} are the garbage output. There is more than one reversible function associated with each irreversible function depending on the number of constant inputs and garbage outputs. Thus the circuits resulting from synthesizing these embedded reversible functions have a different number of gates and hence have different quantum costs. A traditional and reversible full adder is shown in Figure 2.1.

Table 2.2: Truth table of a reversible full adder function

g_i	c_{in}	x	y	c_{out}	sum	g_{o1}	g_{o2}
0	0	0	0	0	0	0	0
0	0	0	1	0	1	1	1
0	0	1	0	0	1	1	0
0	0	1	1	1	0	0	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	0	1	0
0	1	1	1	1	1	0	1
1	0	0	0	1	0	0	0
1	0	0	1	1	1	1	1
1	0	1	0	1	1	1	0
1	0	1	1	0	0	0	1
1	1	0	0	1	1	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	1	0	1

(a) Traditional full adder [9] (b) Reversible full adder [6]
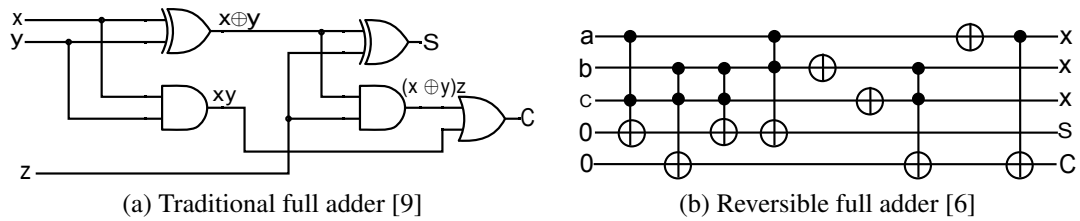
Figure 2.1: A traditional and reversible full adder

2.4 Reversible Gates

A reversible gate has the same number of inputs and outputs and realizes a reversible function. All traditional logic gates (except NOT) are irreversible because they have a different number of inputs and outputs. On the other hand, Toffoli [35], Fredkin [7], and Peres [27] gates are the most popular reversible gates. Definitions of these gates are given below. In this research we focus solely on the Toffoli gate.

5

2.4.1 Toffoli Gates

An n-bit Toffoli gate or Multiple Control Toffoli (MCT) gate is a reversible gate that has n inputs and n outputs where $(i_1, i_2, \ldots, i_n)$ is the input vector, $(o_1, o_2, \ldots, o_n)$ is the output vector, and $o_j = i_j$ where $j = 1, 2, \ldots, n-1$ and $o_n = i_1 i_2 \ldots i_{n-1} \oplus i_n$. The first $n-1$ bits are known as controls and the last n^{th} bit is known as the target. The MCT gate passes all the inputs to the outputs and inverts the target bit when all control bits are 1. When $n = 1$, this gate is known as a NOT gate with no controls. When $n = 2$, the gate is known as a controlled-NOT (CNOT) gate or Feynman gate. For the sake of simplicity we assume that the nth bit is the target; however the target bit could be any of the n bits with which the gate interacts.

A negative-control Toffoli gate is a gate that may have one or more negative controls. The gate maps the n inputs $(i_1, i_2, \ldots, i_n)$ to the n outputs $(o_1, o_2, \ldots, o_n)$ where $o_j = i_j$, $j = 1, 2, \ldots, n-1$ and $o_n = \overline{i_1} i_2 \ldots i_{n-1} \oplus i_n$, and $\overline{i_1}$ is a negative control. This gate passes all the inputs to the outputs and inverts the target bit when all the positive controls have value 1 and negative controls have value 0.

When a Toffoli gate has more than one target, this gate is called a multiple target Toffoli gate. This gate passes all the inputs to the outputs and inverts the target bit when all the controls are 1 and is known as a multiple target Toffoli gate or an extended Toffoli gate (ETG).

Here, $\oplus$ represents the target line, $\bullet$ indicates a positive control, and $\circ$ is use to indicate a negative control line. A Toffoli gate can also be written as $T(C; t)$ where C is the set of controls and t is the target line. The size of a Toffoli gate refers to the number of controls plus target. Figure 2.2 illustrates different versions of the Toffoli gate as described above.

2.4.2 Fredkin Gates

An n-bit Fredkin gate or Multiple Control Fredkin (MCF) gate is a reversible gate that has n inputs and n outputs where $(i_1, i_2, \ldots, i_n)$ is the input vector, $(o_1, o_2, \ldots, o_n)$ is the out-

(a) A NOT gate   (b) A CNOT gate   (c) A 3-bit Toffoli gate

(d) A negative-control Toffoli gate   (e) A multiple-target Toffoli gate
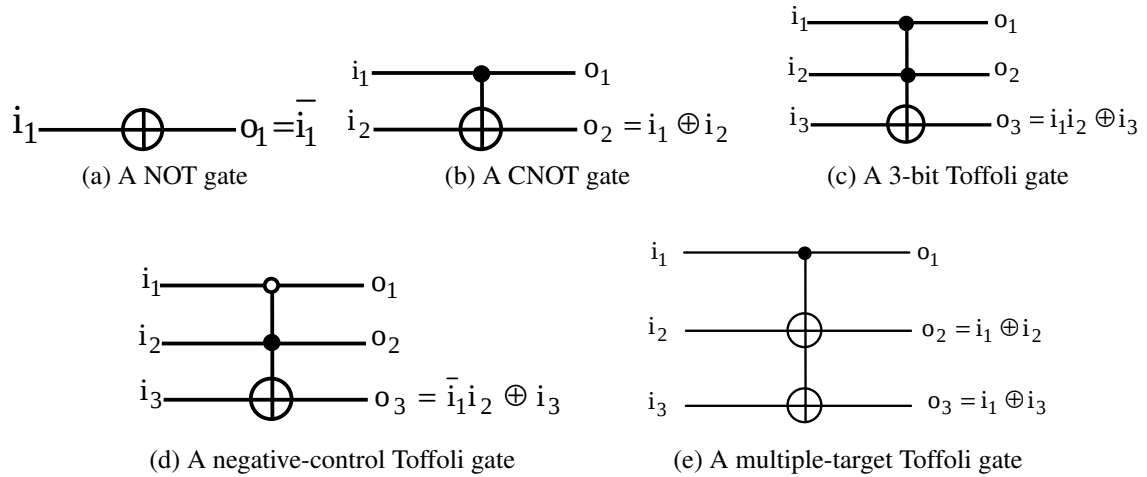
Figure 2.2: Toffoli gates

put vector, and $o_j = i_j$ where $j = 1, 2, \ldots, n-2$ and $o_{n-1} = (\overline{i_1}\, \overline{i_2} \ldots \overline{i_{n-2}}) i_{n-1} + i_1 i_2 \ldots i_{n-2} i_n$ and $o_n = (\overline{i_1}\, \overline{i_2} \ldots \overline{i_{n-2}}) i_n + i_1 i_2 \ldots i_{n-2} i_{n-1}$. Here, $\oplus$ represents the target and $\bullet$ indicates the positive control line. The first $n-2$ bits are known as controls and the last $n-1$ and n bits are known as the targets. This gate passes all the inputs to the outputs and they are swapped iff all the first $n-2$ bits are 1. A 2-bit Fredkin gate is known as a swap gate. A swap gate and a 3-bit Fredkin gate are shown in Figure 2.3.

(a) A SWAP gate   (b) A 3-bit Fredkin gate
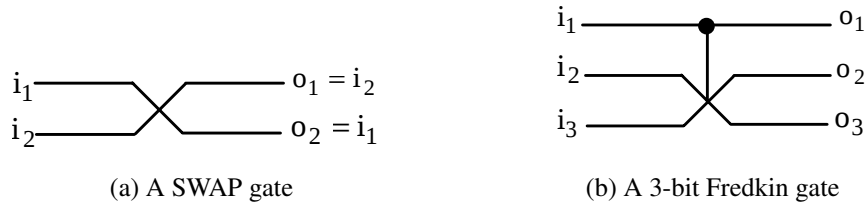
Figure 2.3: A swap gate and a 3-bit Fredkin gate

2.4.3 Peres Gate

A 3-bit Peres gate is a reversible gate that has 3-inputs and 3-outputs where (i_1, i_2, i_3) is the input vector and (o_1, o_2, o_3) is the output vector, where $o_1 = i_1$, $o_2 = i_1 \oplus i_2$ and $o_3 = i_1 i_2 \oplus i_3$. Here, $\oplus$ represents the target and $\bullet$ indicates the positive control line. A 3-bit Peres gate is shown in Figure 2.4.
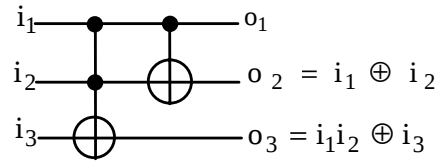
Figure 2.4: A 3-bit Peres gate

2.5 Reversible Gate Library

A gate library consists of different types of gates that can be used in building a circuit. In traditional logic circuits a commonly used gate library consists of AND, OR, and NOT gates. A reversible gate library is composed of a set of reversible gates that can be combined to realize any reversible function. Different gate libraries developed using reversible gates are listed below [37]:

- Multiple Control Toffoli gates (MCT)

- Multiple Control Fredkin gates (MCF)

- NOT, CNOT, and Toffoli gates (NCT)

- Multiple Control Toffoli gates plus Peres gates (MCT+P)

- Multiple Control Toffoli gates plus Multiple Control Fredkin gates (MCT+MCF)

2.6 Reversible Circuit

A reversible circuit is a cascade of reversible gates, generally from a gate library as discussed above, without fan-out and feedback. If a reversible circuit is built using only NOT, CNOT, and Toffoli gates (NCT) or Multiple Control Toffoli gates (MCT), then it is known as a Toffoli circuit. Figure 2.5 shows an example of a Toffoli circuit.
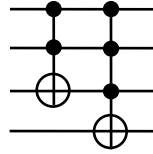
Figure 2.5: A reversible circuit

2.7 Self-reversible and Conservative

A gate is said to be self-reversible if the gate reverses its logic function [26]. The Toffoli gate is self-reversible. Consider the circuit shown in Figure 2.6. The outputs of the second Toffoli gate are $X = P$, $Y = Q$, and $Z = PQ \oplus R$, and the input of the second Toffoli gate is the output of the first Toffoli gate, that is, $P = A$, $Q = B$, and $R = AB \oplus C$. By substituting P, Q, R with A, B, C, we obtain $X = A$, $Y = B$, and $Z = AB \oplus AB \oplus C = C$.
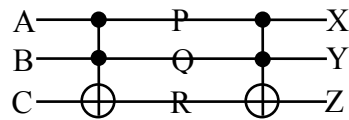
Figure 2.6: Self-reversibility of Toffoli gate

A gate is said to be conservative if it preserves the number of logical ones in the input [26]. The Toffoli gate is not a conservative gate because the mapping changes from (111) to (110) as shown in Table 2.7(a). This property allows us to build sequential circuits with zero internal power dissipation.

As shown in Table 2.7(b), the Fredkin gate maintains both the self-reversible and conservative property.

2.8 Cost Metrics

A given reversible function may be realized in different ways, resulting in different circuits. The following three common cost metrics are used in evaluating the reversible circuits.

9

i_1	i_2	i_3	o_1	o_2	o_3
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

(a) 3-bit Toffoli gate

i_1	i_2	i_3	o_1	o_2	o_3
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	1

(b) 3-bit Fredkin gate

Figure 2.7: Truth table of (a) the 3-bit Toffoli gate and (b) the 3-bit Fredkin gate

2.8.1 Gate Count

Gate count is the simplest way to evaluate different reversible circuits. This refers to a simple count of the number of gates in a circuit. It does not, however, consider the complexity of the circuit. Consider two circuits where the first circuit consists of three 2-input Toffoli gates and the second circuit consists of two 6-input Toffoli gates. In this case a gate count might indicate that the second circuit is preferable, as it has fewer gates. However, the second circuit contains significantly more complex gates.

2.8.2 Garbage Output

Garbage output is another measure to evaluate circuits. In reversible circuits, some extra outputs are required to maintain the reversibility. Adding a garbage output adds extra line(s) to the circuit. In some cases it is not possible to remove all the garbage outputs. According to [21], reduction of reversible circuit cost is possible by adding lines.

2.8.3 Quantum Cost

The quantum cost is an important measure for comparison of reversible circuits. The quantum cost of a gate is defined as the number of basic quantum operations needed to realize the gate [13]. Any reversible gate can be decomposed into basic quantum (1×1 and 2×2) gates. The number of basic quantum gates required to implement a circuit is

referred to as the quantum cost of the circuit. The quantum cost of the NOT, CNOT, and 3-bit Toffoli gate is 1, 1, and 5, respectively. In general, as the number of controls for a gate increases so does the quantum cost.

The quantum cost of an n-bit negative control Toffoli gate with at least one control is exactly the same as the cost of an n-bit Toffoli gate. When all the controls are negative, an extra cost of 2 is required if zero or $(n-3)$ garbage lines are used. An additional cost of 4 is required when only one garbage line is used [17]. The quantum costs for the Toffoli gates are given in Table 2.3 [10]. The first column represents the size of the gate, the second column represents the number of garbage line, the third column is the name of the gate, and the last column is the quantum cost.

Table 2.3: Quantum cost of Toffoli gates

Size (n)	Garbage	Name	Quantum Cost	
			with all positive controls	with all negative controls
1	0	NOT,t1	1	1
2	0	CNOT,t2	1	3
3	0	Toffoli,t3	5	6
4	0	Toffoli,t4	13	15
5	0	t5	29	31
5	2	t5	26	28
6	0	t6	61	63
6	1	t6	52	56
6	3	t6	38	40
7	0	t7	125	127
7	1	t7	80	84
7	4	t7	50	52
8	0	t8	253	255
8	1	t8	100	104
8	5	t8	62	64
9	0	t9	509	511
9	1	t9	128	132
9	6	t9	74	76
10	0	t10	1021	1023
10	1	t10	152	156
10	7	t10	86	88
$n > 10$	0	tn	$2^n - 3$	$2^n - 1$
$n > 10$	1	tn	$24n - 88$	$24n - 84$
$n > 10$	$n - 3$	tn	$12n - 34$	$12n - 32$

Since a Fredkin gate is efficiently simulated by a size n Toffoli gate and 2 CNOT gates, the cost of a size n Fredkin gate is calculated by adding 2 to the size n Toffoli gate.

The cost of $T(a,b,c)\ T(a,b)$ (Peres gate) or $T(a,b)\ T(a,b,c)$ (inverse of Peres gate) is set to be 4 instead of $5+1=6$, as the quantum implementation of each of these patterns was found with cost 4 [19].

2.9   Summary

This chapter describes the necessary background required for the subsequent chapters. The chapter focuses on various reversible gates, reversible circuits, reversible gate libraries, and cost metrics for the evaluation of reversible circuits.

Chapter 3

Reversible Logic Synthesis and Post-Synthesis Approaches

3.1 Logic Synthesis

Logic synthesis is the process of realizing a logic function into a circuit design in terms of gates. Reversible logic synthesis tools transform a given reversible function to a reversible circuit. In reversible logic synthesis, the number of gates, the number of constant inputs, and the number of garbage outputs in the cascade should be as small as possible. The resulting circuit from different synthesis approaches may not be optimal. Hence, post synthesis optimizations are required to achieve low quantum cost reversible circuits. A general flow in reversible logic synthesis approaches is depicted in Figure 3.1.
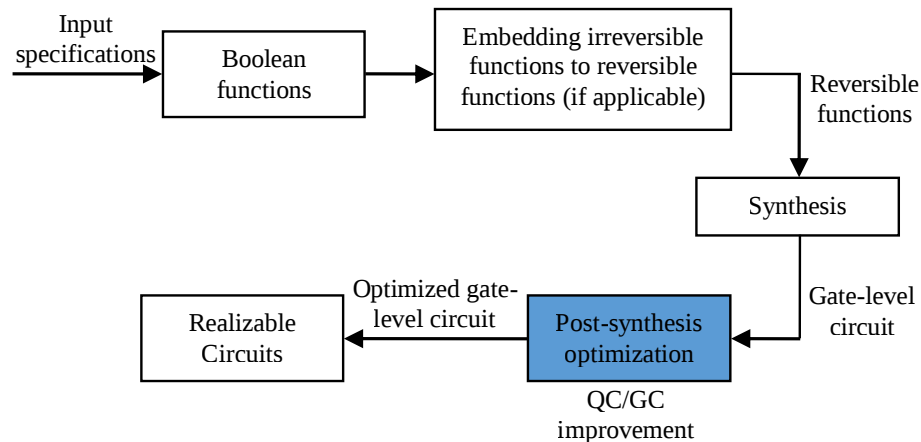
Figure 3.1: General flow in reversible logic synthesis approaches

A Boolean function is the input to the process. The function can be either reversible or irreversible. An irreversible function can be embedded into a reversible function by adding

constant inputs and garbage outputs [12]. Different synthesis approaches are available to realize the given function. For instance, the transformation-based synthesis approach [19] deals only with reversible functions. On the other hand, ESOP-based synthesis [6] can take reversible or irreversible functions as input. The output of any synthesis step is the gate-level circuit. The resulting circuits may not be optimal. Post-synthesis optimization techniques need to be applied to minimize the gate count and quantum cost of a circuit. In the following section we summarize several existing logic synthesis approaches. Section 3.3 gives a brief overview of the existing post-synthesis optimization approaches for reversible logic.

3.2 Logic Synthesis Approaches of Reversible Logic

Several reversible logic synthesis approaches have been proposed, including transformation based synthesis [19], Exclusive-OR Sum-of-Products (ESOP) based synthesis [6], and binary decision diagram (BDD) based synthesis [36].

3.2.1 Transformation-based Synthesis

The transformation-based approach [19] is based on the examination of the truth table of a given reversible function. A basic naive and greedy algorithm has been given to synthesize the circuits in one direction. The approach maps the input to the corresponding output by transformation. The individual steps of each transformation correspond to a cascade of gates. When both sides are matched, the cascade of gates implements the given function. The steps of the basic algorithm include:

- use NOT gates to transform the first row of the output to $00\ldots0$ and update the corresponding positions of all the output rows in the truth table.

- use a simplest gate to transform each output pattern to the corresponding input pattern without affecting the previous rows.

This approach does not introduce any unnecessary garbage outputs and the circuit will have at most $(m-1)2^m + 1$ gates, where m is the number of variables.

An improvement of the basic algorithm is the bidirectional algorithm also proposed in [19]. This algorithm can be applied in both directions simultaneously. The approach then compares the gates at both the input and output side of the circuit and chooses the gates which offer the best advantage. The resulting circuit has a reduced number of gates compared to the circuit provided by the basic algorithm.

3.2.2 Binary Decision Diagram-based Synthesis

Every Boolean function f can be represented by a Binary Decision Diagram (BDD). A Binary Decision Diagram (BDD) over Boolean variables X is a directed acyclic graph $G = (V; E)$ with the following properties [32]:

- Each node $v \in V$ is either a terminal or a nonterminal.

- Each terminal node $v \in V$ is labeled by a value $t \in T = \{0, 1\}$ and has no outgoing edge.

- Each nonterminal node $v \in V$ is labeled by a Boolean variable $x_i \in X$.

- The Shannon decomposition [33], $f = \overline{x_i} f_{x_i=0} + x_i f_{x_i=1}$, $(1 \leq i \leq n)$ holds for each nonterminal node which leads to two outgoing edges $e \in E$ whose successors are denoted by $low(v)$ (for $f_{x_i=0}$) and $high(v)$ (for $f_{x_i=1}$), respectively.

The size of a BDD is defined by the number of non-terminal nodes. The size of a BDD can be significantly reduced, if shared nodes are used. Asymptotically, the resulting reversible circuits are bounded by the size of the BDD.

The approach proposed in [36] can cope with Boolean functions containing more than one hundred variables. The approach synthesizes a given reversible function from a BDD representation of the function. A BDD of the given reversible function is built using a tool such as CUDD [34]. Each node of the BDD is substituted by a cascade of Toffoli gates.

BDDs may use shared nodes. The use of shared nodes in BDD causes fan-outs in the resulting circuit, which are not allowed in reversible logic. To avoid fan-outs, an additional line is introduced. If f is a function with n primary inputs which is represented by a BDD containing k nodes, then the resulting circuit consists of at most $k + n$ circuit lines and $3k$ gates.

3.2.3 ESOP-based Synthesis

The Exclusive-OR Sum-Of-Products (ESOP) representation of a Boolean function is the same as the SOP (Sum-Of-Products) representation except that the OR operator is replaced by the EX-OR operator. For example, the SOP and ESOP expression of the function $f(x,y,z) = (x + y\bar{z})(\bar{x} + \bar{y} + z)$ are $\bar{x}y\bar{z} + x\bar{y} + xz$ and $x \oplus y\bar{z}$, respectively. A function in the ESOP form is commonly written as a list of cubes, known as a cube-list. In any cube, a literal is a Boolean variable in the negative or positive polarity. A product term composed of the Boolean AND of literals is called a cube. 1 denotes the positive polarity, 0 denotes the negative polarity, and $-$ denotes the don't care value. The cube list of a full adder (rd32_19) is illustrated in Figure 3.2.

$$
\begin{array}{ccc|cc}
x_1 & x_2 & x_3 & f_1 & f_2 \\
1 & 1 & - & 1 & 0 \\
1 & - & 1 & 1 & 1 \\
- & 1 & 1 & 1 & 0 \\
0 & - & 0 & 0 & 1 \\
- & 0 & - & 0 & 1 \\
\end{array}
$$

Figure 3.2: Full Adder ESOP cube list

The approach proposed in [6] derives a cascade of Toffoli gates from an ESOP cubelist. Each positive (negative) control leads to a positive (negative) literal in the respective cube. Each line that does not contain a control connection is represented by a don't-care in the cube. Like the BDD-based approach, the ESOP-based approach has the ability to handle functions with large number of input variables. The resulting circuit will require $2n + m$ lines, where n is the number of inputs and m is the number of outputs. The two input

lines correspond to each input in the positive and negative polarity. The output of the basic algorithm generates a Toffoli gate for each cube of each output in the list of ESOP cubes.

Since the basic algorithm always requires $2n + m$ lines, an optimization technique was also proposed in [6]. However, in many cases the negated forms of a literal are not used, and this is easy to get by using a NOT gate when necessary. This way the basic approach optimizes by removing the unnecessary negated lines, which considerably reduces the number of lines to $n + m$ but adds some NOT gates.

As described in section 2.4.1, a Toffoli gate has both positive and negative controls. By using this gate in the ESOP based circuits, the gate count and quantum cost can be reduced. A Toffoli gate is generated for each cube of each output. If a variable has positive polarity in the cube, the positive control of the Toffoli gate is connected to the input line. If the variable has negative polarity in the cube, the negative control of the Toffoli gate is connected to that line. Thus additional NOT gates are not required except for the one cube with negative polarity only.

3.2.4 Shared Cube Synthesis

The approaches discussed in the previous section generate Toffoli gates for each cube in the ESOP expression. The generated Toffoli gates have the same controls but different targets when the cubes are shared by two or more outputs. The shared cube synthesis approach [30] improves the quality of the circuit by generating a Toffoli gate for each such cube and transfers the cube to the other output lines via CNOT gates. Using a greedy approach the algorithm examines all pairs of outputs that have the largest number of common cubes. One Toffoli gate is generated for each of these cubes and then a CNOT gate is added to connect the shared Toffoli gates between two outputs. This process is repeated for all pairs of outputs with shared cubes. Finally, Toffoli gates for the remaining cubes are added to the circuit. As in the optimized ESOP-based approach [6] the resulting circuit also contains $n + m$ lines. This approach proposed in [30] does not use negative-control Toffoli

gates; however, the usage of this type of gates was later suggested in [29] and [31].

The approach described in [30, 29, 31] takes the best advantage of shared functionality if the ESOP terms are shared by only two outputs. However, if the shared terms exist in more than two outputs then transformation of each term may require more than one Toffoli gate, which is inefficient. The approach proposed in [24] optimizes the synthesis by generating exactly one Toffoli gate for a cube. CNOT gates then are used to pass these to other outputs. The cubes which are not shared by multiple outputs are called ungrouped cubes and are treated separately. The resulting circuits generated by this approach used negative-control Toffoli gates. The approach is much better compared to the shared cube synthesis approach [30] in terms of quantum cost. We use the circuits resulting from the improved shared cube synthesis approach [24] for post-synthesis optimization.

3.3   Post-synthesis Optimization

The circuits obtained by different synthesis methods may not optimal. This can be achieved by rearranging the gates to improve gate count, quantum cost, and garbage output in a circuit. Gate count simply refers to the number of gates in a circuit without considering the complexity of the gate. The quantum cost of a complex gate is much higher than a simple gate. Consider the two circuits shown in Figure 3.3. The circuit shown in Figure 3.3(b) has higher gate count but lower quantum cost compared to the circuit in Figure 3.3(a). To calculate the quantum cost of a gate, we use the values from Table 2.3. As described in section 2.8.3, quantum cost is one of the most important measures to evaluate circuits. Garbage output is also another measure to evaluate circuits. Template matching is one post-synthesis technique used to improve the gate count and quantum cost of a circuit.

If a circuit is non-optimal then it may be possible to decrease the size and quantum cost of a circuit by replacing sequences of gates with another equivalent sequence; this is known as a template-driven reduction method, or template matching [19]. Template matching is an approach to reduce the number of gates and quantum cost by removing unnecessary

Figure 3.3: Gate count comparison of two circuits

gates from the network and has no effect on the functionality of the circuit. A complete set of templates for reversible circuits is given in [19]. For instance, some of the templates are shown in Figure 3.4. The matching procedure finds all suitable sets of gates for each template. When a template match is found, the substitution is performed by the templates. The process then repeats and may cause a template rejected earlier to become applicable.
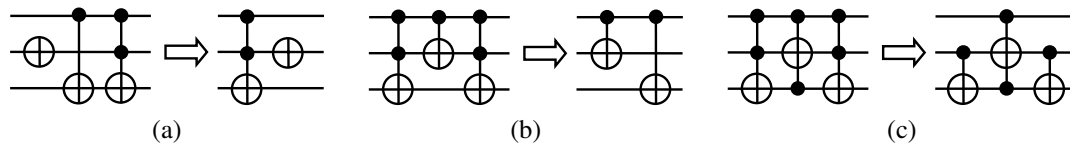
Figure 3.4: Some reversible templates [19]

One of the properties of reversible circuits is that if two circuits c_1 and c_2 realize the functions f and f^{-1}, then $c_1 c_2$ realizes the identity function. Therefore, if a sub-circuit realizes the identity function, then the sub-circuit can be removed from the circuit [15]. A template matching method generally defines all the templates up to a certain size for a given gate library. All Fredkin-Toffoli templates with less than six gates are given in [14]. All Toffoli gate templates of size up to 7 and some templates of size 9 can be found in [16]. Figure 3.5 shows some reversible templates. A circuit is examined to find a subsequence of gates (more than half) in either forward or backward direction in a template. Then the matched sequence of gates in the circuit can be substituted with the inverse cascade of the remaining sequence of gates in the corresponding template [15]. For instance, the template $G_0 G_1 \ldots G_{m-1}$ can be applied in two directions as follows [15]:

- Forward application: A sequence of gates in the circuit that matches the cas-
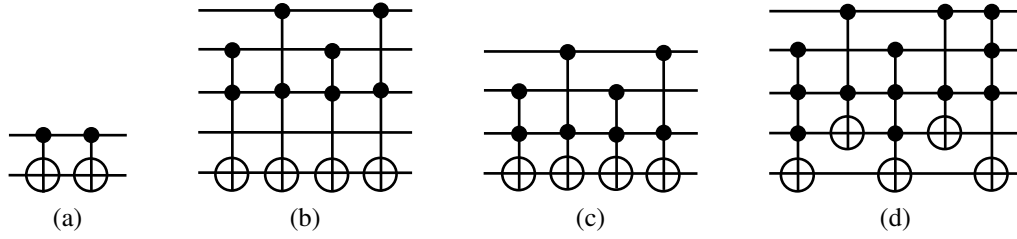
Figure 3.5: Some reversible templates [15]

cade $G_i G_{(i+1)\ \mathrm{mod}\ m} \cdots G_{(i+k-1)\ \mathrm{mod}\ m}$ in the template is replaced with the sequence $G^{-1}_{(i-1)\ \mathrm{mod}\ m} G^{-1}_{(i-2)\ \mathrm{mod}\ m} \cdots G^{-1}_0 G^{-1}_{m-1} \cdots G^{-1}_{(i+k)\ \mathrm{mod}\ m}$, where $\frac{m}{2} \leq k \leq m$.

- Backward application: A sequence of gates in the circuit that matches the cascade $G_i G_{(i-1)\ \mathrm{mod}\ m} \cdots G_{(i-k+1)\ \mathrm{mod}\ m}$ in the template is replaced with the sequence $G^{-1}_{(i+1)\ \mathrm{mod}\ m} G^{-1}_{(i+2)\ \mathrm{mod}\ m} \cdots G^{-1}_{m-1} G^{-1}_0 \cdots G^{-1}_{(i-k)\ \mathrm{mod}\ m}$, where $\frac{m}{2} \leq k \leq m$.

If more than half of the gates in the template are matched, then the portion of the network is replaced with the remaining gate in the template. However, it may be advantageous to replace exactly the half of the matched gates, if the replaceable gates have lower quantum cost than the replaced gates. For instance, if a template consists of four gates, then the idea is to find a portion of the circuit that matches a majority of the template, e.g. the first three gates, and then these three gates can be replaced with the remaining gate in the template. Consider the example shown in Figure 3.6, where the cascade has 6 Toffoli gates. The last gate in the cascade can be moved into the third position (from the left) as shown in Figure 3.6(b) based on the moving rule described in section 5.1.1. Then the first 3 gates match with the templates as shown in Figure 3.5(b) and are substituted by the remaining gates from the template. The resulting circuit after template matching is shown in Figure 3.6(c).

In [2], the authors defined positive/negative control Toffoli gates as PNC gates and proposed new merging, moving, and splitting rules and an algorithm for PNC gates. A deletion rule is defined as if two adjacent PNC gates have the same control and target bits, then the two gates can be deleted. This is also known as self-reversibility as described in section 2.7. If two PNC gates have the same target bit and control bits, but one of the control

(a) (b) (c)
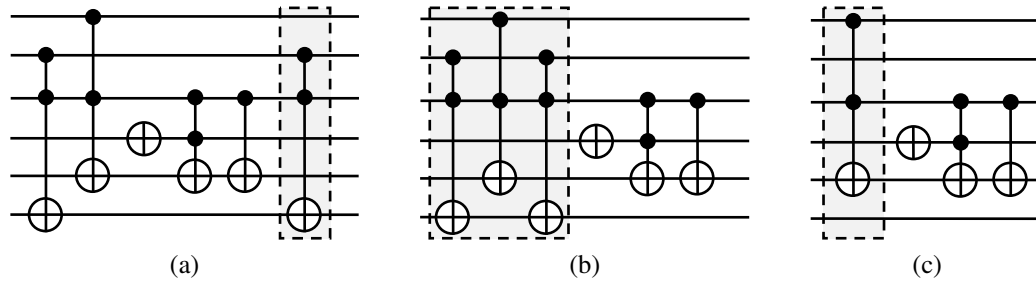
Figure 3.6: An example of identity template matching

bits has reverse value, then one gate is termed as a brother gate relative to the other gate. The two brother gates can be merged into one gate. This new merged gate is defined as a parent gate and the two brother gates are defined as child gates relative to the parent gate. A child gate can be obtained by merging the parent gate and the other child gate. These rules are defined as merging rules [2]. Alternatively, a parent gate can be split into its two child gates or a child gate can be split into its parent gate and its brother gate. This is defined as splitting rules [2]. Figure 3.7(a) and 3.7(b) shows two brother (child) gates and a parent gate in Figure 3.7(c).

(a) (b) (c)
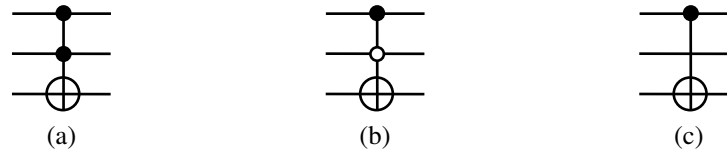
Figure 3.7: Brother (child) gates and parent gate [2]

If the target bit of one gate is not the control of the other gate or two gates have the same target bit then the position of these two gates can be interchanged which is known as a moving rule. A simplification algorithm utilizing the above deletion, merging, splitting, and moving rules was proposed in [2].

Templates and rules using both positive and negative control Toffoli gates were proposed in [4]. Templates were introduced that allow for a substitution of a cascade of (positively controlled) Toffoli gates with a single but an equivalent (negatively controlled) Toffoli gate. For example, a cascade of Toffoli gates with all possible combinations of positive

control connections can be subsumed into a single negative control Toffoli gate as shown in Figure 3.8, where the left circuit has only one 3-bit negative control Toffoli gate. The quantum cost of all 3-bit negative control Toffoli gates is 6. This gate can be represented by the cascade of 4 Toffoli gates with all possible combinations of positive control connections. The gate count for this cascade is 4 and the quantum cost is 8, which is more than the single 3-bit all negative control Toffoli gate. Initially, this transformation increases the number of gates and also the quantum cost of a circuit. However, this transformation will increase more chance to apply template matching rules with other gates. By analyzing
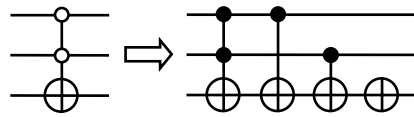
Figure 3.8: Substitution of a cascade of positive control Toffoli gates with an equivalent single negative control Toffoli gate [4]

these patterns, 7 generalized rules were proposed for post synthesis optimization to reduce both the number of gates and the quantum costs [4]. The proposed algorithm traverses the given reversible circuit and checks for any possible rules and the algorithm is iterated until no further reduction is possible. In another paper [5], the authors defined two MCT gates as adjacent gates if they have targets on the same line and differ in only one line with respect to the control connections. These adjacent gates are defined as brother (child) gates in [2] and are depicted in Figure 3.7. If two MCT gates have targets on the same line and differ in two control lines, then they are defined as distance-2 gates. The replacement rules for the distance-2 gates were proposed in [5]. The merging rules for adjacent gates were also discussed as merging rules in [2] and [32]. In [5], the authors proposed an optimization algorithm using the merging and replacement rules to optimize the circuits and showed that the algorithm performs better than the algorithm proposed in [4].

In [32], the author defined the negative/positive Toffoli gate as a Mixed Polarity Multiple Control Toffloi (MPMCT) gate and gave reduction rules that can be applied to MPMCT gates as follows:

- $T(C;x_t)T(C;x_t) \equiv I$

- $T(C;x_t)T(C \cup x_i;x_t) \equiv T(C \cup \overline{x_i};x_t)$

- $T(C \cup x_i;x_t)T(C \cup x_j;x_t) \equiv T(x_i;x_j)T(C \cup x_j;x_t)T(x_i;x_j)$

- $T(C \cup \overline{x_i};x_t)T(C \cup \overline{x_j};x_t) \equiv T(x_i;x_j)T(C \cup x_j;x_t)T(x_i;x_j)$

- $T(C \cup x_i;x_t)T(C \cup \overline{x_j};x_t) \equiv T(x_i;x_j)T(C \cup \overline{x_j};x_t)T(x_i;x_j)$

To optimize the circuit, the above reduction rules are applied with the new optimization procedure proposed in [32].

3.4 Summary

In this chapter several logic synthesis approaches for reversible circuits have been briefly described. We have also discussed various template matching techniques with examples.

Chapter 4

Template Matching with Negative Controls

In chapter 3, we have discussed several existing logic synthesis and post-synthesis optimization approaches. In this chapter the proposed templates and algorithms are described with experimental results.

4.1   Proposed Approach

Toffoli gates can appear in various ways in a circuit, as shown in Figure 4.1. The target of a Toffoli gate can be either totally different than the control line of the other gates or can be in one of the control lines of another gate. For instance, in the ESOP based synthesis approach [6] the target line of a gate is always different than the control line of the other gates. On the other hand, in the transformation based approach [19], the target of a gate can be in any of the control lines of other gates. Based on the target line, pairs of gates in a circuit can be categorized as:

- same or different size gates having the same target line, as shown in Figure 4.1(a) and 4.1(b), or

- same or different size gates having different target lines, as shown in Figure 4.1(c) and 4.1(d).

Different target line Toffoli gates can be further classified as:

(a)

(b)

(c)

(d)
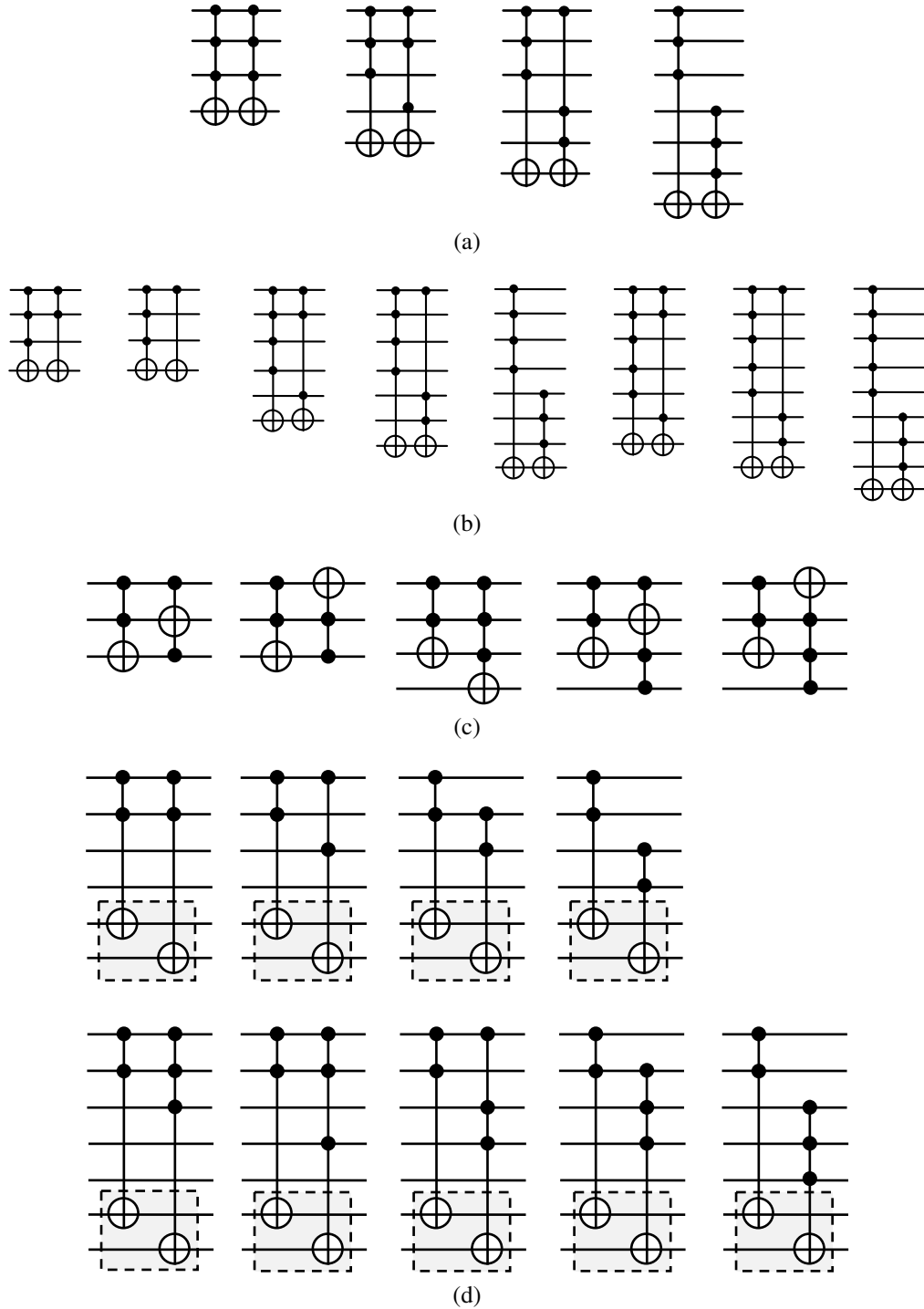
Figure 4.1: Possible ways for two Toffoli gates to appear in a circuit.

- different target line Toffoli gates having targets on any line, as shown in Figure 4.1(c), or

- different target line Toffoli gates having targets always other than control lines, as

shown in Figure 4.1(d).

In developing our templates we considered the various ways in which two Toffoli gates with the same target line can appear in a circuit, as shown in Figure 4.1(a) and 4.1(b).

We have proposed 7 templates that may be applied in various situations. Templates $1-5$ can be applied to two adjacent Toffoli gates $T_1(C_1,t_1)$ and $T_2(C_2,t_2)$ where C_i is the set of controls, $|C_1|=|C_2|$ and t_i is the target, $|t_1|=|t_2|$. In templates $1-4$, two gates share the same control line but in template 5 one of the controls of one gate is on a different line. Templates $6-7$ can be applied to two different size Toffoli gates $T_1(C_1,t_1)$ and $T_2(C_2,t_2)$ where C_i is the set of controls, $|C_1|>|C_2|$ or $|C_2|>|C_1|$ and t_i is the target, $|t_1|=|t_2|$. In template 6 the two gates may differ, but only by at most 1 line. In template 7, the difference in the size of two Toffoli gates is at least 1. In all cases we are interested in Toffoli gates that have the same target line. We propose 2 new templates for positive and negative control Toffoli gates (templates 4 and 7). Template 4 can be applied to two $\geq$ 3-bit Toffoli gates with controls on the same lines while template 7 can be applied to two different size $\geq$ 3-bit Toffoli gates. Details of each type of template are as follows.

4.1.1 Template 1

Template 1 can be applied to two adjacent CNOT gates in the case where one CNOT gate has a positive control and the other has a negative control. In this case the two CNOT gates can be replaced by a single NOT gate [4]. An example is shown in Figure 4.2 and the proof is given below.

$$T(C;x_t)T(\overline{C};x_t) \equiv T(;x_t) \tag{4.1}$$
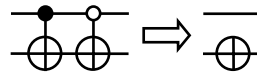
Figure 4.2: Template 1

Proof: Two CNOT gates have different controls. Thus the output of this two adja-

cent CNOT gates as: $T(C;x_t)T(\overline{C};x_t) = C \oplus x_t \oplus \overline{C} = 1 \oplus x_t = \overline{x_t} \simeq T(;x_t)$ which can be represented by a NOT gate.

4.1.2 Template 2

If two Toffoli gates have the same controls, then the two gates negate each other. This property is known as self-reversibility [26] as introduced in section 2.7.

$$T(C;x_t)T(C;x_t) \equiv I \tag{4.2}$$
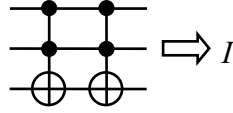
Figure 4.3: Template 2

Proof: Since the two Toffoli gates have same controls and target, the output of one Toffoli gate is the input of the next Toffoli gate. The output of the second Toffoli gate: $T(C;x_t)T(C;x_t) = C \oplus x_t \oplus C = C \oplus C \oplus x_t = 0 \oplus x_t = x_t \simeq T(;)$, which is equal to the input and the two gates negate each other.

4.1.3 Template 3

If two Toffoli gates have the same controls but one of the controls is the inverse polarity of that of the previous gate, then these two gates can be replaced by one Toffoli gate with all the common controls [2]. The expression is given in Equation 4.3 and an example is shown in Figure 4.4.

$$T(C \cup x_i;x_t)T(C \cup \overline{x_i};x_t) \equiv T(C;x_t) \tag{4.3}$$
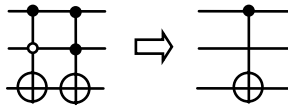
Figure 4.4: Template 3

Proof: As shown in Figure 4.4, the two Toffoli gates have controls of opposite polarity in one of the controls. The output of this cascade is: $T(C \cup x_i; x_t)T(C \cup \overline{x_i}; x_t) = Cx_i \oplus x_t \oplus C\overline{x_i} = C(x_i \oplus \overline{x_i}) \oplus x_t = C \oplus x_t = C \oplus x_t \simeq T(C; x_t)$, which includes all the common controls and the target.

4.1.4 Template 4

If two n-bit $(n \geq 3)$ Toffoli gates have controls on the the same lines but two (i.e. x_i, x_j) of the controls have different polarity, then the two n-bit $(n \geq 3)$ gates can be replaced by two CNOT gates and one $(n-1)$-bit $(n \geq 2)$ Toffoli gate. Equations (4.4a) and (4.4b) formalize this, while Figure 4.5 illustrates two possible ways to apply this template.

$$T(C \cup x_i \cup \overline{x_j}; x_t)T(C \cup \overline{x_i} \cup x_j; x_t) \equiv T(x_i; x_j)T(C \cup x_j; x_t)T(x_i; x_j) \qquad (4.4a)$$

$$T(C \cup \overline{x_i} \cup \overline{x_j}; x_t)T(C \cup x_i \cup x_j; x_t) \equiv T(x_i; x_j)T(C \cup \overline{x_j}; x_t)T(x_i; x_j) \qquad (4.4b)$$
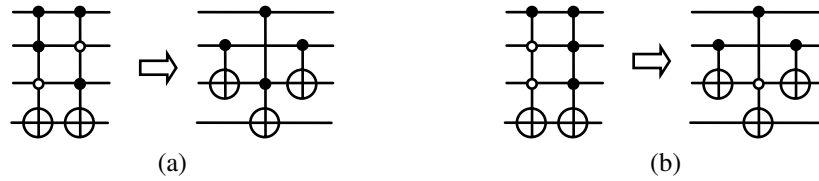
Figure 4.5: Template 4

Proof: As shown in Figure 4.5 and in Equations (4.4a) and (4.4b), the output of these two cascades can be represented by two CNOT gates and one $(n-1)$-bit Toffoli gate as proved in Equations (4.5a) and (4.5b).

$$T(C \cup x_i \cup \overline{x_j}; x_t)T(C \cup \overline{x_i} \cup x_j; x_t) = Cx_i\overline{x_j} \oplus x_t \oplus C\overline{x_i}x_j$$

$$= C(x_i\overline{x_j} \oplus \overline{x_i}x_j) \oplus x_t$$

$$= C(x_i \oplus x_j) \oplus x_t$$

$$\simeq T(x_i; x_j)T(C \cup x_j; x_t)T(x_i; x_j) \qquad (4.5a)$$

28

$$T(C \cup \overline{x_i} \cup \overline{x_j}; x_t) T(C \cup x_i \cup x_j; x_t) = C\overline{x_i x_j} \oplus x_t \oplus Cx_i x_j$$

$$= C(\overline{x_i x_j} \oplus x_i x_j) \oplus x_t$$

$$= C(x_i \overline{x_j} \oplus \overline{x_j} \oplus x_i x_j) \oplus x_t$$

$$= C(x_i x_j \oplus x_i \oplus \overline{x_j} \oplus x_i x_j) \oplus x_t$$

$$= C(x_i \oplus \overline{x_j}) \oplus x_t$$

$$\simeq T(x_i; x_j) T(C \cup \overline{x_j}; x_t) T(x_i; x_j) \tag{4.5b}$$

4.1.5 Template 5

This template can be applied to two Toffoli gates of the same size where one of the controls is on a different line. In this case the two Toffoli gates can be replaced by two CNOT gates and one Toffoli gate [32]. The three situations in which this may occur are formally described in Equations (4.6a), (4.6b), and (4.6c) and illustrated in Figure 4.6.

$$T(C \cup x_i; x_t) T(C \cup x_j; x_t) \equiv T(x_i; x_j) T(C \cup x_j; x_t) T(x_i; x_j) \tag{4.6a}$$

$$T(C \cup \overline{x_i}; x_t) T(C \cup \overline{x_j}; x_t) \equiv T(x_i; x_j) T(C \cup x_j; x_t) T(x_i; x_j) \tag{4.6b}$$

$$T(C \cup \overline{x_i}; x_t) T(C \cup x_j; x_t) \equiv T(x_i; x_j) T(C \cup \overline{x_j}; x_t) T(x_i; x_j) \tag{4.6c}$$

Proof: The above threes cases can be represent by two CNOT gates and one Toffoli gate.

(a)   (b)   (c)
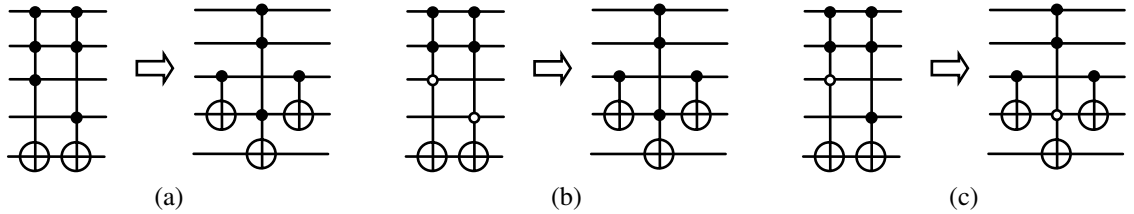
Figure 4.6: Template 5

The proof of these three cases is shown in Equations (4.7a), (4.7b), and (4.7c).

$$
\begin{aligned}
T(C \cup x_i; x_t) T(C \cup x_j; x_t) &= C x_i \oplus x_t \oplus C x_j \\
&= C(x_i \oplus x_j) \oplus x_t \\
&\simeq T(x_i; x_j) T(C \cup x_j; x_t) T(x_i; x_j) \quad\quad \text{(4.7a)}
\end{aligned}
$$

$$
\begin{aligned}
T(C \cup \overline{x_i}; x_t) T(C \cup \overline{x_j}; x_t) &= C \overline{x_i} \oplus x_t \oplus C \overline{x_j} \\
&= C(\overline{x_i} \oplus \overline{x_j}) \oplus x_t \\
&= C(x_i \oplus 1 \oplus x_j \oplus 1) \oplus x_t \\
&= C(x_i \oplus x_j) \oplus x_t \\
&\simeq T(x_i; x_j) T(C \cup x_j; x_t) T(x_i; x_j) \quad\quad \text{(4.7b)}
\end{aligned}
$$

$$
\begin{aligned}
T(C \cup \overline{x_i}; x_t) T(C \cup x_j; x_t) &= C \overline{x_i} \oplus x_t \oplus C x_j \\
&= C(\overline{x_i} \oplus x_j) \oplus x_t \\
&= C(x_i \oplus 1 \oplus x_j) \oplus x_t \\
&= C(x_i \oplus \overline{x_j}) \oplus x_t \\
&\simeq T(x_i; x_j) T(C \cup \overline{x_j}; x_t) T(x_i; x_j) \quad\quad \text{(4.7c)}
\end{aligned}
$$

4.1.6 Template 6

If the size of two Toffoli gates differs by 1 and all the controls except the additional control in the larger gate are on the same lines, then this sequence of gates can be replaced by a Toffoli gate of the same size as the larger gate [32]. The two situations are described in Equations (4.8a) and (4.8b) and illustrated in Figure 4.7.

$$
T(C; x_t) T(C \cup x_i; x_t) \equiv T(C \cup \overline{x_i}; x_t) \quad\quad \text{(4.8a)}
$$

$$
T(C; x_t) T(C \cup \overline{x_i}; x_t) \equiv T(C \cup x_i; x_t) \qu\quad\quad \text{(4.8b)}
$$

Proof: As shown in Figure 4.7, the two different sized Toffoli gates have controls on the

(a)   (b)
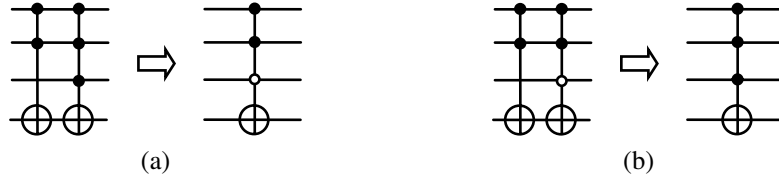
Figure 4.7: Template 6

same lines except the additional control of the larger gate. The above two scenarios can be represented by a Toffoli gate of the same size as the larger gate as proved below and shown in Equations (4.9a) and (4.9b).

$$T(C;x_t)T(C\cup x_i;x_t) = C\oplus x_t \oplus Cx_i$$

$$= C(1\oplus x_i)\oplus x_t$$

$$= C\overline{x_i}\oplus x_t$$

$$\simeq T(C\cup\overline{x_i};x_t) \tag{4.9a}$$

$$T(C;x_t)T(C\cup\overline{x_i};x_t) = C\oplus x_t \oplus C\overline{x_i}$$

$$= C(1\oplus\overline{x_i})\oplus x_t$$

$$= Cx_i\oplus x_t$$

$$\simeq T(C\cup x_i;x_t) \tag{4.9b}$$

4.1.7 Template 7

This template can be applied to two different sized n-bit $(n \geq 3)$ Toffoli gates as described in Equations (4.10aa)-(4.10db) and illustrated in Figure 4.8.

$$T(C\cup x_i\cup x_j;x_t)T(C\cup x_k;x_t) \equiv T(x_i\cup x_j;x_k)T(C\cup x_k;x_t)T(x_i\cup x_j;x_k) \tag{4.10aa}$$

$$T(C\cup x_i\cup x_j;x_t)T(C\cup\overline{x_k};x_t) \equiv T(x_i\cup x_j;x_k)T(C\cup\overline{x_k};x_t)T(x_i\cup x_j;x_k) \tag{4.10ab}$$

$$T(C\cup\overline{x_i}\cup x_j;x_t)T(C\cup x_k;x_t) \equiv T(\overline{x_i}\cup x_j;x_k)T(C\cup x_k;x_t)T(\overline{x_i}\cup x_j;x_k) \tag{4.10ba}$$

$$T(C\cup\overline{x_i}\cup x_j;x_t)T(C\cup\overline{x_k};x_t) \equiv T(\overline{x_i}\cup x_j;x_k)T(C\cup\overline{x_k};x_t)T(\overline{x_i}\cup x_j;x_k) \tag{4.10bb}$$
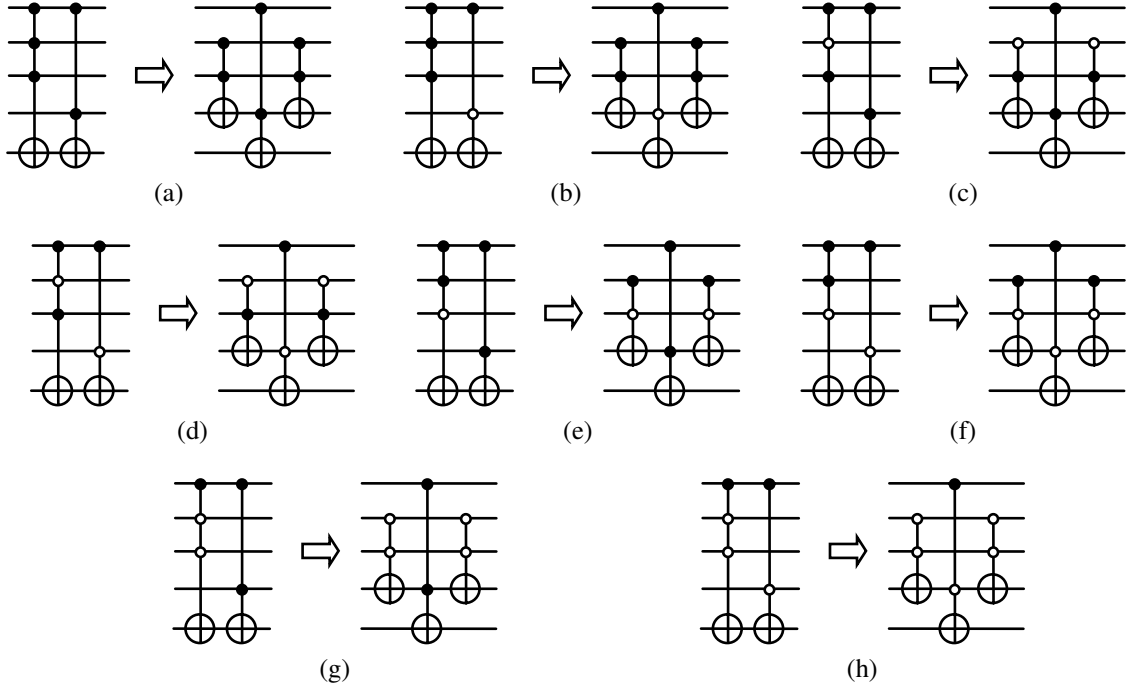
Figure 4.8: Template 7

$$T(C\cup x_i\cup\overline{x_j};x_t)T(C\cup x_k;x_t)\equiv T(x_i\cup\overline{x_j};x_k)T(C\cup x_k;x_t)T(x_i\cup\overline{x_j};x_k) \tag{4.10ca}$$

$$T(C\cup x_i\cup\overline{x_j};x_t)T(C\cup\overline{x_k};x_t)\equiv T(x_i\cup\overline{x_j};x_k)T(C\cup\overline{x_k};x_t)T(x_i\cup\overline{x_j};x_k) \tag{4.10cb}$$

$$T(C\cup\overline{x_i}\cup\overline{x_j};x_t)T(C\cup x_k;x_t)\equiv T(\overline{x_i}\cup\overline{x_j};x_k)T(C\cup x_k;x_t)T(\overline{x_i}\cup\overline{x_j};x_k) \tag{4.10da}$$

$$T(C\cup\overline{x_i}\cup\overline{x_j};x_t)T(C\cup\overline{x_k};x_t)\equiv T(\overline{x_i}\cup\overline{x_j};x_k)T(C\cup\overline{x_k};x_t)T(\overline{x_i}\cup\overline{x_j};x_k) \tag{4.10db}$$

Proof: The proofs of all the above cases are given in Equations (4.1a)-(4.1h).

$$T(C\cup x_i\cup x_j;x_t)T(C\cup x_k;x_t)=Cx_ix_j\oplus x_t\oplus Cx_k$$

$$=C(x_ix_j\oplus x_k)\oplus x_t$$

$$\simeq T(x_i\cup x_j;x_k)T(C\cup x_k;x_t)T(x_i\cup x_j;x_k) \tag{4.1a}$$

$$T(C\cup x_i\cup x_j;x_t)T(C\cup\overline{x_k};x_t)=Cx_ix_j\oplus x_t\oplus C\overline{x_k}$$

$$=C(x_ix_j\oplus\overline{x_k})\oplus x_t$$

$$\simeq T(x_i\cup x_j;x_k)T(C\cup\overline{x_k};x_t)T(x_i\cup x_j;x_k) \tag{4.1b}$$

32

$$T(C \cup \overline{x_i} \cup x_j; x_t)T(C \cup x_k; x_t) = C\overline{x_i}x_j \oplus x_t \oplus Cx_k$$

$$= C(\overline{x_i}x_j \oplus x_k) \oplus x_t$$

$$\simeq T(\overline{x_i} \cup x_j; x_k)T(C \cup x_k; x_t)T(\overline{x_i} \cup x_j; x_k) \qquad (4.1c)$$

$$T(C \cup \overline{x_i} \cup x_j; x_t)T(C \cup \overline{x_k}; x_t) = C\overline{x_i}x_j \oplus x_t \oplus C\overline{x_k}$$

$$= C(\overline{x_i}x_j \oplus \overline{x_k}) \oplus x_t$$

$$\simeq T(\overline{x_i} \cup x_j; x_k)T(C \cup \overline{x_k}; x_t)T(\overline{x_i} \cup x_j; x_k) \qquad (4.1d)$$

$$T(C \cup x_i \cup \overline{x_j}; x_t)T(C \cup x_k; x_t) = Cx_i\overline{x_j} \oplus x_t \oplus Cx_k$$

$$= C(x_i\overline{x_j} \oplus x_k) \oplus x_t$$

$$\simeq T(x_i \cup \overline{x_j}; x_k)T(C \cup x_k; x_t)T(x_i \cup \overline{x_j}; x_k) \qquad (4.1e)$$

$$T(C \cup x_i \cup \overline{x_j}; x_t)T(C \cup \overline{x_k}; x_t) = Cx_i\overline{x_j} \oplus x_t \oplus C\overline{x_k}$$

$$= C(x_i\overline{x_j} \oplus \overline{x_k}) \oplus x_t$$

$$\simeq T(x_i \cup \overline{x_j}; x_k)T(C \cup \overline{x_k}; x_t)T(x_i \cup \overline{x_j}; x_k) \qquad (4.1f)$$

$$T(C \cup \overline{x_i} \cup \overline{x_j}; x_t)T(C \cup x_k; x_t) = C\overline{x_i}\overline{x_j} \oplus x_t \oplus Cx_k$$

$$= C(\overline{x_i}\overline{x_j} \oplus x_k) \oplus x_t$$

$$\simeq T(\overline{x_i} \cup \overline{x_j}; x_k)T(C \cup x_k; x_t)T(\overline{x_i} \cup \overline{x_j}; x_k) \qquad (4.1g)$$

$$T(C \cup \overline{x_i} \cup \overline{x_j}; x_t)T(C \cup \overline{x_k}; x_t) = C\overline{x_i}\overline{x_j} \oplus x_t \oplus C\overline{x_k}$$

$$= C(\overline{x_i}\overline{x_j} \oplus \overline{x_k}) \oplus x_t$$

$$\simeq T(\overline{x_i} \cup \overline{x_j}; x_k)T(C \cup \overline{x_k}; x_t)T(\overline{x_i} \cup \overline{x_j}; x_k) \qquad (4.1h)$$

4.2 Basic Template Matching Algorithm

We propose a basic algorithm to apply the above templates. This algorithm maintains two separate gate lists; the original list of gates, and a new list of gates that at the end of the algorithm will replace the original list. The basic template matching process is performed as follows:

Consider two adjacent gates g_1 and g_2 from the gate list of a circuit.

1. if g_1 and g_2 have the same target line then we begin searching for templates

    (a) if g_1 and g_2 match any of the templates then replace g_1 and g_2 with the equiva-
        lent gates from that template (i.e. g_1', g_2', ...) and add the new gates at the end
        of the new gate list and then move on to consider the next two gates (i.e. g_3 and
        g_4) in the original gate list; go to step 1.

    (b) if no match is found then add g_1 at the end of the new gate list, g_2 and g_3 become
        the gates under consideration; go to step 1.

2. else add g_1 and g_2 at the end of new gate list and consider the next two gates (i.e. g_3
   and g_4) in the original gate list; go to step 1.

This algorithm is iterated until no further reduction is possible in quantum cost i.e., after each iteration the quantum cost of the new gate list is compared to the quantum cost of the old gate list. If there is a reduction in quantum cost, then the new gate list becomes the old gate list, and a new iteration begins.

Consider the example shown in Figure 4.9(a) where the input circuit has 7 gates. The quantum cost of the circuit is 15. Consider the first two gates from the circuit and apply templates according to the algorithm. Since the first two gates have two different targets, we consider the next two gates (second and third gate). The second and third gates also have two different targets, so we move on to the next two gates (third and fourth gate). As shown in Figure 4.9(b), these two gates have the same target line, so then we search for templates and apply template 5. In this way we consider all of the gates as we search for templates. Now all adjacent pairs of remaining gates have different target lines, thus there are no possibilities to apply templates using this approach. The output circuit is shown in Figure 4.9(c).

(a) An example input circuit

(b) Applying template 5

(c) Output circuit
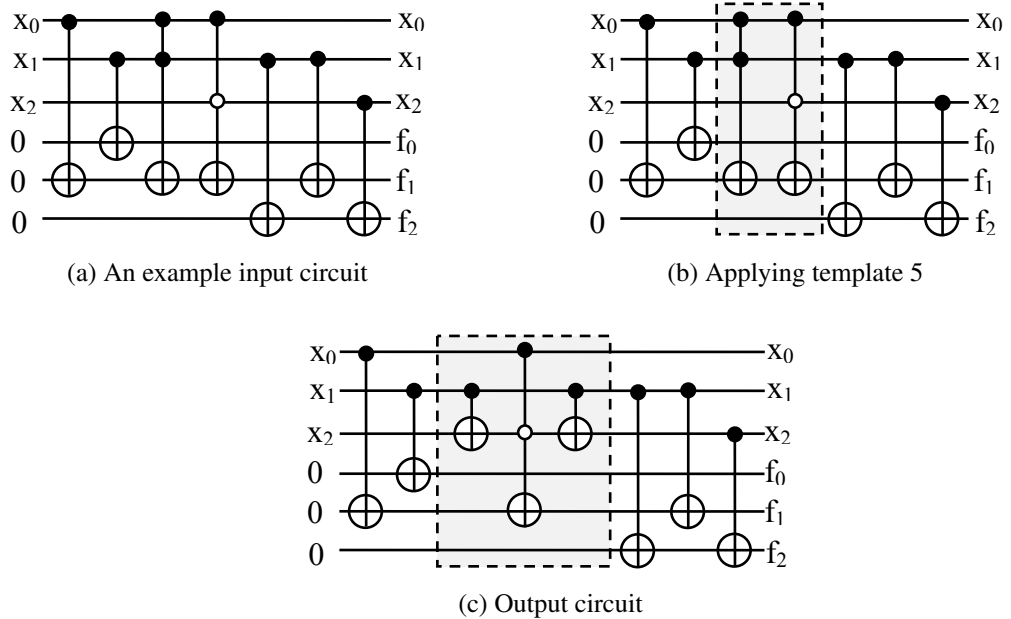
Figure 4.9: Illustration of basic template matching algorithm

4.2.1 Experimental Results and Discussion

The proposed templates along with the basic algorithm were implemented in Java. The programs have been run on an Intel Core 2 Duo CPU T6670 @ 2.20GHz×2 system running Ubuntu 13.04 with 2GB main memory for 110 benchmark circuits. These benchmarks were obtained from RevLib [37] and preprocessed by applying the improved shared cube synthesis approach [22]. All the resulting circuits are verified using QMDD (Quantum Multiple-valued Decision Diagrams) [20]. Using QMDD, we compare the resulting circuits (after applying templates) with the input circuits, in order to ensure that the behavior of the circuit has not been modified. The running time is negligible and the results are listed in Table 4.1. Table 4.1 compares the outputs obtained in the current experiment to the results from the improved shared cube synthesis approach in terms of quantum cost and gate count. In this table PrevGC/PrevQC refers to the gate count/quantum cost obtained from the circuit generated by the improved shared cube synthesis approach, while NewGC/NewQC refers to the new gate count/quantum cost as computed from the circuits generated from the basic template matching post-processing approach. The proposed templates reduce the quantum

cost of circuits 8.76% on average. col4_135 is the best reported circuit in terms of reduction in quantum cost. Among 110 circuits 40 of them showed improvement.

Table 4.1: Applying templates using basic algorithm

Circuit	PrevGC[22]	NewGC	GCImp.(%)	PrevQC[22]	NewQC	QCImp.(%)
co14_135	14	21	-50.00	3472	1750	49.60
4mod5_8	4	4	0.00	21	13	38.10
cm85a_127	48	60	-25.00	2206	1434	35.00
add6_92	153	170	-11.11	5135	4075	20.64
majority_176	5	6	-20.00	133	106	20.30
fredkin_3	7	8	-14.29	15	12	20.00
0410184_85	218	249	-14.22	7636	6330	17.10
adr4_93	41	45	-9.76	645	538	16.59
rd32_19	6	7	-16.67	25	22	12.00
clip_124	78	85	-8.97	3824	3386	11.45
x2_223	23	24	-4.35	433	385	11.09
cm82a_126	17	18	-5.88	126	115	8.73
alu_9	4	5	-25.00	40	37	7.50
f51m_159	327	348	-6.42	28382	26292	7.36
max46_177	42	45	-7.14	4524	4210	6.94
root_197	48	51	-6.25	1811	1688	6.79
alu4_98	454	473	-4.19	41127	38701	5.90
9symml_91	52	57	-9.62	10943	10393	5.03
sym9_71	52	57	-9.62	10943	10393	5.03
5xp1_90	58	59	-1.72	786	750	4.58
rd73_69	43	44	-2.33	856	820	4.21
sym10_207	83	88	-6.02	15640	15006	4.05
ex3_152	4	5	-25.00	76	73	3.95
rd84_70	68	70	-2.94	2329	2241	3.78
alu2_96	78	80	-2.56	4369	4217	3.48
example2_156	78	80	-2.56	4369	4217	3.48
tial_214	459	472	-2.83	43412	42032	3.18
misex3_180	854	868	-1.64	49076	47670	2.86
cycle10_2_61	42	43	-2.38	1273	1237	2.83
life_175	50	52	-4.00	6074	5960	1.88
urf2_73	479	483	-0.84	8742	8582	1.83
dist_144	94	95	-1.06	3700	3652	1.30
misex3c_181	822	828	-0.73	49720	49152	1.14
urf3_75	1501	1512	-0.73	53157	52553	1.14
urf1_72	960	963	-0.31	23769	23649	0.50
ex1010_155	1675	1678	-0.18	52788	52594	0.37
hwb9_65	1011	1013	-0.20	23471	23405	0.28
sqr6_204	54	55	-1.85	583	582	0.17
hwb7_15	233	234	-0.43	3015	3012	0.10
urf5_76	210	211	-0.48	5364	5359	0.09
Average			**-7.73**			**8.76**

4.3 Summary

In this chapter we have described the various possible ways that Toffoli gates appear in a circuit. Seven templates have been discussed, two of them are new. A basic algorithm was also proposed and discussed with an example. The experimental results were also discussed with a list of the benchmark circuits used in our experiments.

Chapter 5

Improved Template Matching Algorithm

In the previous chapter, we discussed several templates with a basic template matching algorithm. This chapter presents improved template matching algorithms along with use of a moving rule. Section 5.2 describes the basic template matching process along with the moving rule. An improved algorithm is presented in section 5.3.

5.1 Gate Rearrangements

Gate rearrangements in a circuit increases the possibilities for matching more templates and can lead to further optimization. Gate rearrangements have usually been performed based on the moving rule [11]. The moving rule preserves the functional behavior of a circuit while moving gates within the circuit. In the example circuit shown in Figure 5.1(a), the gate count for this circuit is 7 and the quantum cost is 15. After rearranging gates and applying templates the gate count of the new circuit is 7, while the quantum cost has been reduced to 11. The gate rearrangements and templates applied are shown in Figure 5.1. The basic template matching techniques along with the moving rule are described below.

5.1.1 Moving Rule

Two adjacent gates $g(C_1, t_1)$ and $g(C_2, t_2)$ in a reversible circuit can be interchanged iff $C_1 \cap t_2 = \emptyset$ and $C_2 \cap t_1 = \emptyset$, i.e. the target of each gate is not a control of the other gate [11]. From Figure 4.9(a) and 5.1(b) it can be seen that, according to the moving rule the first CNOT gate $T(x_0; f_1)$ can pass the second CNOT gate $T(x_1; f_0)$ because the target of the first gate is not the control of the second gate. This movement allows the application of

(a) An example input circuit

(b) Moving gate

(c) Applying template 6

(d) Applying template 5

(e) Output circuit
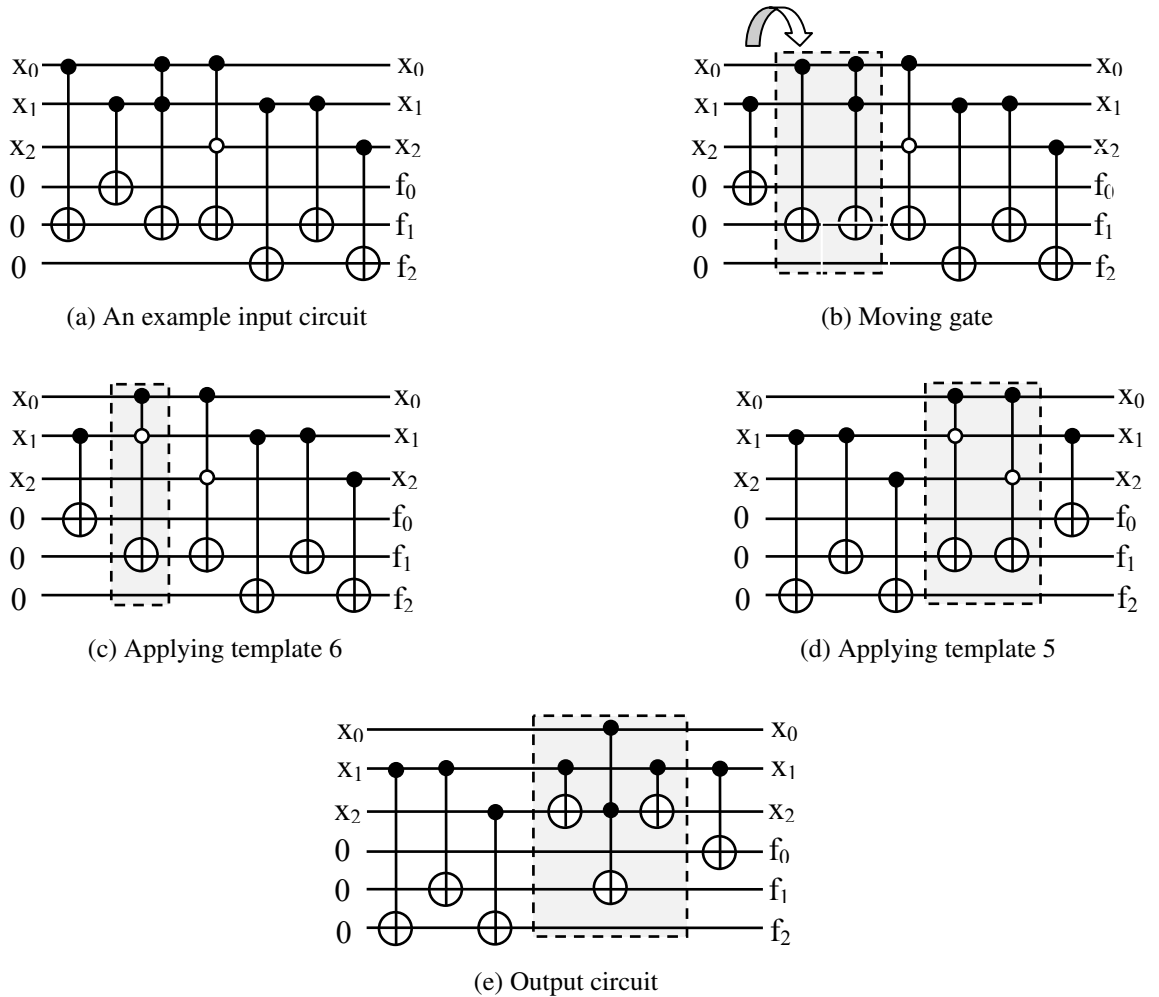
Figure 5.1: Illustration of applying moving rule

template 6 on gates 2 and 3 and generates a new Toffoli gate with positive and negative controls.

5.2 Basic Algorithm with Moving Rule

The template matching process along with use of the moving rule is performed as follows:

Consider two gates g_1 and g_2 from the gate list of a circuit.

1. if g_1 and g_2 have the same target line then we can check for template matches:

    (a) if g_1 and g_2 match any of the templates then replace g_1 and g_2 with the equiva-

lent gates from that template (i.e. g'_1, g'_2, ...) and add the new gates to the new gate list and then move on to consider the next two gates (i.e. g_3 and g_4) in the original gate list; go to step 1.

(b) if no match is found for any template then apply the moving rule:

    i. if g_1 can pass g_2 then interchange g_1 and g_2; add g_2 into the new gate list, g_1 and g_3 become the gates under consideration; go to step 1.

    ii. else add g_1 and g_2 to the new gate list and consider the next two gates from the original gate list (i.e. g_3 and g_4); go to step 1.

2. else apply moving rule to g_1 and g_2

(a) if g_1 can pass g_2 then interchange g_1 and g_2; add g_2 into the new gate list, g_1 and g_3 become the gates under consideration; go to step 1.

(b) else add g_1 and g_2 to the new gate list and consider the next two gates from the original gate list (i.e. g_3 and g_4) in the circuit; go to step 1.

This algorithm is iterated until no further reduction is possible in quantum cost i.e., after each iteration the quantum cost of the new gate list is compared to the quantum cost of the old gate list. If there is a reduction in quantum cost, then the new gate list becomes the old gate list, and a new iteration begins.

5.2.1 Experimental Results and Discussion

The proposed algorithm with the moving rule has been implemented in Java and tested on the same platform for the same set of circuits as the first experiment. The resulting circuits were again verified using QMDD [20]. The run time is again negligible. Results are listed in Table 5.1. Table 5.1 compares the outputs obtained in the current experiment to the results from the improved shared cube synthesis approach in terms of quantum cost and gate count. In this table PrevGC/PrevQC refers to the gate count/quantum cost obtained from the circuit generated by the improved shared cube synthesis approach, while

NewGC/NewQC refers to the new gate count/quantum cost as computed from the circuits generated from our template matching post-processing. The proposed templates reduce the quantum cost of circuits 16.93% on average which is an improvement of almost double that of the previous experiment. 86 of the 110 circuits generated by the improved shared cube synthesis approach [22] showed improvement in quantum cost after applying our templates. As in the previous experiment col4_135 is the best reported circuit in terms of reduction in quantum cost. The next best reported circuit is cm85a_127 where the quantum cost reduction (44.15%) is higher than the previous (35%). decod24-enable_32 has the third best reduction in quantum cost (41.38%) although we did not obtain any improvement in the previous experiment. Gate count average (15.91%) also improved compared with the previous gate count average (-7.73%). table3_209 exhibited the greatest reduction in gate count, at 71.33%. 4mod5_8, fredkin_3, adr4_93, x2_223, miller_5, sqn_203, and pcler8_190 showed no changes in gate count, but significant reductions in quantum cost.

Table 5.1: Applying templates using basic algorithm with moving rule

Circuit	PrevGC [22]	NewGC	GCImp.(%)	PrevQC [22]	NewQC	QCImp.(%)
co14_135	14	21	-50.00	3472	1750	49.60
cm85a_127	48	54	-12.50	2206	1232	44.15
decod24-enable_32	9	5	44.44	29	17	41.38
bw_116	287	94	67.25	637	387	39.25
4mod5_8	4	4	0.00	21	13	38.10
decod24_10	9	4	55.56	16	10	37.50
C7552_119	89	32	64.04	399	250	37.34
decod_137	89	32	64.04	399	250	37.34
ham15_30	114	46	59.65	263	183	30.42
ham7_29	37	17	54.05	67	47	29.85
rd73_69	43	52	-20.93	856	619	27.69
add6_92	153	159	-3.92	5135	3714	27.67
clip_124	78	80	-2.56	3824	2803	26.70
fredkin_3	7	7	0.00	15	11	26.67
mod5d2_17	15	12	20.00	38	28	26.32
0410184_85	218	256	-17.43	7636	5662	25.85
plus127mod8192_78	36	31	13.89	803	602	25.03

Table 5.1: Applying templates using basic algorithm with moving rule

Circuit	PrevGC [22]	NewGC	GCImp.(%)	PrevQC [22]	NewQC	QCImp.(%)
adr4_93	41	41	0.00	645	489	24.19
apla_107	72	40	44.44	1683	1277	24.12
dc1_142	31	18	41.94	127	97	23.62
mod5mils_18	11	12	-9.09	30	23	23.33
max46_177	42	52	-23.81	4524	3540	21.75
3_17_6	11	9	18.18	28	22	21.43
cycle10_2_61	42	46	-9.52	1273	1004	21.13
sym6_63	13	16	-23.08	721	571	20.80
plus63mod8192_80	35	31	11.43	847	672	20.66
majority_176	5	6	-20.00	133	106	20.30
sym10_207	83	105	-26.51	15640	12990	16.94
cm42a_125	42	17	59.52	161	134	16.77
pm1_192	42	17	59.52	161	134	16.77
graycode6_11	12	10	16.67	12	10	16.67
cm151a_129	26	25	3.85	769	642	16.51
dc2_143	51	39	23.53	1084	906	16.42
sqrt8_205	22	23	-4.55	466	393	15.67
root_197	48	44	8.33	1811	1528	15.63
hwb7_15	233	118	49.36	3015	2551	15.39
hwb6_14	92	52	43.48	839	711	15.26
x2_223	23	23	0.00	433	367	15.24
sao2_199	41	33	19.51	3767	3203	14.97
aj-e11_81	18	11	38.89	74	63	14.86
urf2_73	479	254	46.97	8742	7453	14.74
wim_220	23	14	39.13	139	119	14.39
urf5_76	210	115	45.24	5364	4614	13.98
miller_5	9	9	0.00	29	25	13.79
inc_170	75	32	57.33	892	769	13.79
sqn_203	37	37	0.00	1346	1171	13.00
mlp4_184	80	67	16.25	2496	2174	12.90
hwb8_64	480	261	45.63	8195	7158	12.65
5xp1_90	58	44	24.14	786	687	12.60
cu_141	28	20	28.57	781	687	12.04
rd32_19	6	7	-16.67	25	22	12.00
cm82a_126	17	14	17.65	126	111	11.90

Table 5.1: Applying templates using basic algorithm with moving rule

Circuit	PrevGC [22]	NewGC	GCImp.(%)	PrevQC [22]	NewQC	QCImp.(%)
f51m_159	327	359	-9.79	28382	25020	11.85
in0_162	245	115	53.06	7949	7014	11.76
f2_158	14	13	7.14	112	99	11.61
alu4_98	454	483	-6.39	41127	36371	11.56
9symml_91	52	63	-21.15	10943	9729	11.09
sym9_71	52	63	-21.15	10943	9729	11.09
misex1_178	42	22	47.62	332	296	10.84
rd84_70	68	70	-2.94	2329	2079	10.73
table3_209	701	201	71.33	18606	16630	10.62
life_175	50	58	-16.00	6074	5429	10.62
misex3_180	854	576	32.55	49076	43865	10.62
alu3_97	72	56	22.22	1986	1780	10.37
ham3_28	6	5	16.67	10	9	10.00
mod5adder_66	28	26	7.14	353	318	9.92
tial_214	459	490	-6.75	43412	39133	9.86
hwb9_65	1011	554	45.20	23471	21173	9.79
alu2_96	78	82	-5.13	4369	3942	9.77
example2_156	78	82	-5.13	4369	3942	9.77
urf1_72	960	524	45.42	23769	21497	9.56
dist_144	94	82	12.77	3700	3348	9.51
sqr6_204	54	50	7.41	583	528	9.43
sf_232	4	5	-25.00	32	29	9.38
urf3_75	1501	941	37.31	53157	48218	9.29
ex1010_155	1675	775	53.73	52788	48110	8.86
dk17_145	34	27	20.59	1014	930	8.28
misex3c_181	822	584	28.95	49720	45785	7.91
alu_9	4	5	-25.00	40	37	7.50
plus63mod4096_79	32	28	12.50	676	634	6.21
4mod7_26	12	11	8.33	84	79	5.95
pcler8_190	18	18	0.00	323	308	4.64
ex3_152	4	5	-25.00	76	73	3.95
one-two-three_27	8	5	37.50	38	37	2.63
ex2_151	7	8	-14.29	146	143	2.05
cm163a_133	35	27	22.86	546	536	1.83
Average			**15.91**			**16.93**

5.3 Improved Template Matching Algorithm

The algorithm described in section 4.2 considered two consecutive gates in the circuit and applied templates. It didn't consider gate movement in the circuit. However the algorithm described in section 5.2 made use of a moving rule along with the basic algorithm to apply templates. Neither of the previously mentioned algorithms searched for the templates that offered the best match in terms of reduction in quantum cost. In this section we propose an improved algorithm which will consider rank while applying templates. The next section describes the various quantum cost savings for each template, which is used for the ranking of the templates in this algorithm. For instance, consider two gates (g_i and g_{i+1}, $(1 \leq i \leq n)$ where i is the index of a gate and n is the number of gates in a circuit) from a circuit. If g_i and g_{i+1} have the same target, then the algorithm searches for templates and sets the rank for this pair of gates as described in Table 5.2. In the next step, if g_i can pass g_{i+1}, then we consider g_i and the next gate after g_{i+1} (i.e. g_{i+2}) from the circuit. If g_i and g_{i+2} have the same target, then the algorithm searches for templates and sets the rank for these two gates. Now we compare the new rank with the previous rank and update the rank and the pair of gates (i.e. g_i and g_{i+2}) to apply templates if the new rank is less than the previous rank. Next, if gate g_i can also pass g_{i+2}, then we consider g_i with the next gate of g_{i+2} and check conditions to apply templates and update the rank and the pair of gates to apply templates. After considering g_i with all the other gates in the circuit, we get the best rank for g_i and g_j, $(i+1 \leq j \leq n)$. We then apply the template on g_i and g_j and replace the gates with the new set of gates (i.e. g'_i, g'_j, ….). In this way, the algorithm searches for the pair of gates g_i, g_j and corresponding template that offers the best possible reduction in quantum cost.

5.3.1 Quantum Cost Savings in Templates

The 7 templates described in section 4.1 can be applied in various situations. We assign rank to the templates based on the quantum cost savings for each template. The template

ranking strategies are summarized in Table 5.2 with an example for each case and described below.

Template 1

Template 1 can be applied to two adjacent CNOT gates. Alternatively, the cascade of two CNOT gates can be replaced by a single NOT gate. If m and p is the quantum cost of the NOT and CNOT gate, respectively, then the quantum cost is reduced from $2p$ to m.

Template 2

Template 2 can be applied to two n-bit Toffoli gates with the same controls. In this case the two gates negate each other and the gate count and quantum cost savings is 100%.

Template 3

In template 3, two n-bit Toffoli gates are replaced by one $(n-1)$-bit Toffoli gate. If the quantum cost of a n-bit Toffoli gate is x and that of the $(n-1)$-bit Toffoli gate is y, then the quantum cost is reduced from $2x$ to y.
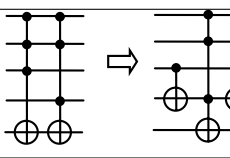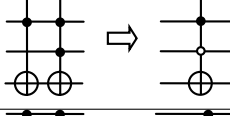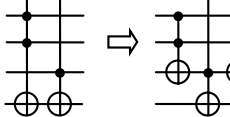
Template 4

Template 4 can be applied to two n-bit Toffoli gates with the conditions described in section 4.1.4 where $n \geq 3$. The cascade is replaced by two CNOT gates and one $(n-1)$-bit Toffoli gate where $n \geq 2$. If x is the quantum cost of an n-bit Toffoli gate, p is the quantum cost of a CNOT gate, and y is the quantum cost of an $(n-1)$-bit Toffoli gate, then the quantum cost is reduced from $2x$ to $2p + y$ and the template is given a rank of 4.

Template 5

In template 5, the cascade of two n-bit Toffoli gates are replaced by two CNOT gates and one n-bit Toffoli gate. If x and p is the quantum cost of an n-bit Toffoli gate and a CNOT gate, respectively, then the quantum cost is reduced from $2x$ to $2p + x$.

Table 5.2: Quantum cost savings of different templates

Templates	QC savings	Example	Example QC savings	Rank
Template 1	$2p$ to m		2 to 1	2
Template 2	$2x$ to 0		26 to 0	1
Template 3	$2x$ to y		26 to 5	3
Template 4	$2x$ to $2p+y$		26 to 7	4
Template 5	$2x$ to $2p+x$		26 to 15	5
Template 6	$y+x$ to x		18 to 13	6
Template 7	$x+y$ to $2q+y$		18 to 15	7

Template 6

Template 6 can be applied to two different size Toffoli gates with the conditions described in section 4.1.6. If the quantum cost of an n-bit Toffoli gate is x and an $(n-1)$-bit Toffoli gate is y, then the quantum cost is reduced from $y+x$ to x and the template is given a rank of 6.
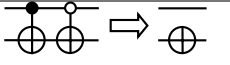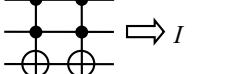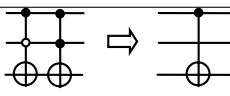
Template 7

Template 7 can be applied to the various situations discussed in section 4.1.7. If x is the quantum cost of an n-bit Toffoli gate, q is the quantum cost of a 3-bit Toffoli gate, and y is the quantum cost of an $(n-1)$-bit Toffoli gate, then the quantum cost is reduced from $x+y$ to $2q+y$ and the template is assigned a rank of 7.
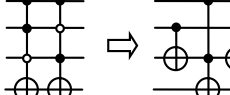
As an example of how these rankings are used, we consider the subcircuit of an arbitrary

circuit shown in Figure 5.2(a) where the subcircuit has three Toffoli gates. All the gates have the same target line. We can see that gates 1 and 2 satisfy the conditions to apply template 4. Now we set rank 4 for gates 1 and 2 and save it. According to the moving rule described in section 5.1.1, gate 1 can pass gate 2 since none of the controls of either gate is the target of the other gate. As shown in Figure 5.2(b), we consider gates 1 and 3 and we can see that these two gates satisfy the conditions to apply template 7. We set rank 7 for gates 1 and 3. Now we compare the new rank (7) with the previous rank (4) and we can see that the new rank (7) is higher than the previous rank. Thus we apply template 4 on gates 1 and 2 and replace with the new set of gates as shown in Figure 5.2(c). In each iteration for each gate in a circuit and considering the moving rule we compare the rank and take the best pair that offers the best savings in quantum cost after applying different templates. In the next section we describe the algorithm.

(a) An example input circuit

(b) Moving gate

(c) Applying template 4
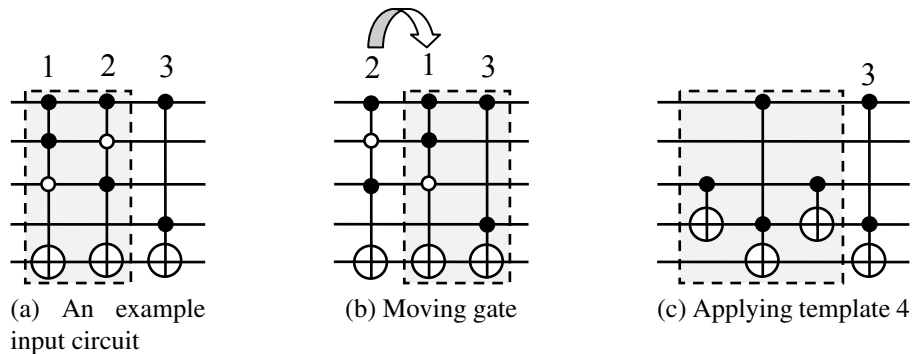
Figure 5.2: Illustration of improved template matching algorithm

5.3.2 Improved Template Matching Algorithm

The improved template matching process is performed as follows: The *Improved_Template_Matching* algorithm takes a circuit as an input, where *gate_list* represents the set of gates and *n* is the the number of gates (*input_gate_list*) in a circuit. This algorithm maintains two separate gate lists; the original list of gates, and a new list of gates that at the end of the algorithm will replace the original list. The algorithm *Process_Circuit(temp_gate_list)* takes *temp_gate_list* as input and returns *new_gate_list* as out-

put after each iteration. The algorithm assigns *new_gate_list* to the *temp_gate_list* and is iterated if the quantum cost of the *new_gate_list* is not equal to the quantum cost of the *temp_gate_list* or the gate count of the *new_gate_list* is not equal to the *temp_gate_list*.

Algorithm 1 Improved Template Matching

 1: **procedure** IMPROVED_TEMPLATE_MATCHING(*input_gate_list*)
 2:     **if** (*input_gate_list* is not empty) **then**
 3:         *temp_gate_list* ← *input_gate_list*
 4:         *iteration* ← 0
 5:         *new_gate_list* = *Process_Circuit*(*temp_gate_list*)
 6:         **if** (*new_qc*! = *temp_qc*||*new_gc*! = *temp_gc*) **then**
 7:             *temp_gate_list* ← *new_gate_list*
 8:             + + *iteration*
 9:             *Improved_Template_Matching*(*temp_gate_list*)
10:         **end if**
11:     **end if**
12: **end procedure**

Algorithm 2 Improved Template Matching

 1: **procedure** PROCESS_CIRCUIT(*gate_list*)
 2:     **for** (*i* ← 1 to *n*) **do** ▷ *n* is the number of gates (gate_list) in a circuit in each iteration
 3:         **if** (*gate*[*i*] is not flagged) **then**
 4:             *Process_Gates*(*i*, *last_gate*, *gate*[*i*])
 5:         **end if**
 6:     **end for**
 7: **end procedure**

For each gate in a circuit, the algorithm *Process_Circuit* calls the function *Process_Gates(i,last_gate,gate[i])* where *i* is the index of the gate in a circuit, *last_gate* is a Boolean variable to check whether gate *i* is the last gate of the circuit, and *gate*[*i*] represents the ith gate in the circuit. The algorithm *Process_Gates* saves the *temp_rank* for each pair of gates and compares with *rank* and updates the *rank* and the pair of gates to apply templates when *temp_rank* is less than *rank*. The algorithm assigns *flag* to the gates of a circuit after applying templates. The function *Process_Gates* sets flag variables for the two gates and returns immediately when two gates follow the self-reversible property, which is the best case. For other cases, the function *Process_Gate* checks and applies templates to

the two gates based on the given conditions and sets flag variables for the two gates. If two gates (i.e. *gate[i]* and *gate[j]*) do not match with any templates then the *gate[j]* is added to the new gate list and the algorithm considers the gate (i.e. *gate[j+1]*) from the circuit and continues the iterations for each gate in a circuit.

Algorithm 3 Applying templates and moving rule

1: **procedure** PROCESS_GATES($i, last\_gate, gate[i]$)
2:     $rank \leftarrow MAX\_VALUE$
3:     **if** (!$last\_gate$) **then**
4:         **for** $j \leftarrow i+1$ to n **do**
5:             **if** ($gate[i].target == gate[j].target$ && $gate[j]$ is not flagged) **then**
6:                 **if** ($apply\_moving\_rule(gate[i], gate[j])$) **then**
7:                     $temp\_rank \leftarrow check\_templates(gate[i], gate[j])$
8:                     **if** ($temp\_rank < rank$) **then**
9:                         $rank \leftarrow temp\_rank$
10:                     **end if**
11:                     **if** ($rank == 1$) **then**   ▷ template 2
12:                         $template\_one(gate[i], gate[j])$
13:                         $gate[i] = true$
14:                         $gate[j] = true$
15:                         **return**
16:                     **end if**
17:                 **end if**
18:             **end if**
19:         **end for**
20:         **if** ($rank == 2$) **then**   ▷ template 1
21:             $template\_two(gate[i], gate[j])$
22:             $gate[i] = true$
23:             $gate[j] = true$
24:         **else if** ($rank == 3$ || $rank == 5$) **then**   ▷ template 3 or 5
25:             $template\_three\_plus\_five(gate[i], gate[j])$
26:             $gate[i] = true$
27:             $gate[j] = true$
28:         **else if** ($rank == 4$) **then**   ▷ template 4
29:             $template\_four(gate[i], gate[j])$
30:             $gate[i] = true$
31:             $gate[j] = true$
32:         **else if** ($rank == 6$) **then**   ▷ template 6
33:             $template\_six(gate[i], gate[j])$

34:     $gate[i] = true$
35:     $gate[j] = true$
36:    **else if** ($rank == 7$) **then**   ▷ template 7
37:     $template\_seven(gate[i], gate[j])$
38:     $gate[i] = true$
39:     $gate[j] = true$
40:    **else**
41:     add $gate[i]$ to the new $gate\_list$
42:     $gate[i] = true$
43:    **end if**
44: **else**
45:   add $gate[i]$ to the new $gate\_list$
46:   $gate[i] = true$
47: **end if**
48: **end procedure**

5.3.3 Experimental Results and Discussion

The ImprovedTemplateMatchingAlgorithm has been implemented in Java and the experiments run on the same platform as we did in the first two experiments. The evaluation of the improved template matching technique is performed on three different test suites. All circuits generated in our experiments have been verified using the QMDD-based verification approach [20]. The run time of all the tested benchmarks was under $81,314$ milliseconds (ms) (urf3_75), and our experiments showed that each circuit was iterated over at most 7 times (f51m_233).

The first test suite consists of the same number of circuits from the improved shared cube synthesis approach [22] that is used in the previous two experiments. In this experiment, 86 circuits showed improvement where 56 of them showed more improvement than the approach described in the previous section. The results are summarized in Table 5.3. For the circuits listed in Table 5.3, the average quantum cost reduction is 23.80% compared to the improved shared cube synthesis approach. In this experiment, for the 86 benchmark circuits the average quantum cost reduction is 21.34% while in the second experiment, the average quantum cost reduction was 15.80%.

Table 5.3: Applying templates using improved template matching algorithm

Circuit	PrevGC [22]	GC 5.1	NewGC	PrevQC [22]	QC 5.1	QCImp.(%) 5.1	NewQC	QCImp.(%)
cm85a_127	48	54	55	2206	1232	44.15	1024	53.58
4mod5_8	4	4	5	21	13	38.10	10	52.38
adr4_93	41	41	38	645	489	24.19	330	48.84
bw_116	287	94	77	637	387	39.25	332	47.88
C7552_119	89	32	29	399	250	37.34	210	47.37
decod_137	89	32	29	399	250	37.34	210	47.37
0410184_85	218	256	257	7636	5662	25.85	4631	39.35
add6_92	153	159	154	5135	3714	27.67	3267	36.38
rd73_69	43	52	51	856	619	27.69	581	32.13
ham15_30	114	46	45	263	183	30.42	179	31.94
clip_124	78	80	85	3824	2803	26.70	2726	28.71
3_17_6	11	9	7	28	22	21.43	20	28.57
sym6_63	13	16	17	721	571	20.80	521	27.74
sqrt8_205	22	23	24	466	393	15.67	339	27.25
urf2_73	479	254	272	8742	7453	14.74	6395	26.85
dc1_142	31	18	19	127	97	23.62	93	26.77
dc2_143	51	39	38	1084	906	16.42	794	26.75
cycle10_2_61	42	46	45	1273	1004	21.13	934	26.63
max46_177	42	52	54	4524	3540	21.75	3324	26.53
hwb8_64	480	261	270	8195	7158	12.65	6172	24.69
sqn_203	37	37	39	1346	1171	13.00	1035	23.11
hwb7_15	233	118	119	3015	2551	15.39	2332	22.65
sao2_199	41	33	33	3767	3203	14.97	2917	22.56
urf1_72	960	524	545	23769	21497	9.56	18445	22.40
ex1010_155	1675	775	810	52788	48110	8.86	41027	22.28
hwb9_65	1011	554	573	23471	21173	9.79	18363	21.76
aj-e11_81	18	11	10	74	63	14.86	58	21.62
urf3_75	1501	941	967	53157	48218	9.29	42226	20.56
root_197	48	44	44	1811	1528	15.63	1452	19.82
dist_144	94	82	80	3700	3348	9.51	2985	19.32
cm42a_125	42	17	17	161	134	16.77	130	19.25
pm1_192	42	17	17	161	134	16.77	130	19.25
5xp1_90	58	44	43	786	687	12.60	637	18.96
hwb6_14	92	52	49	839	711	15.26	683	18.59
sym10_207	83	105	107	15640	12990	16.94	12752	18.47
f51m_159	327	359	366	28382	25020	11.85	23168	18.37
9symml_91	52	63	69	10943	9729	11.09	9027	17.51
sym9_71	52	63	69	10943	9729	11.09	9027	17.51
miller_5	9	9	8	29	25	13.79	24	17.24
sqr6_204	54	50	47	583	528	9.43	484	16.98
alu4_98	454	483	507	41127	36371	11.56	34155	16.95
in0_162	245	115	103	7949	7014	11.76	6630	16.59
mlp4_184	80	67	61	2496	2174	12.90	2092	16.19
urf5_76	210	115	107	5364	4614	13.98	4500	16.11
misex3_180	854	576	583	49076	43865	10.62	41231	15.99
misex3c_181	822	584	602	49720	45785	7.91	41837	15.85
tial_214	459	490	512	43412	39133	9.86	36747	15.35
rd84_70	68	70	70	2329	2079	10.73	1976	15.16
table3_209	701	201	193	18606	16630	10.62	15862	14.75
inc_170	75	32	34	892	769	13.79	768	13.90
life_175	50	58	60	6074	5429	10.62	5277	13.12
alu3_97	72	56	55	1986	1780	10.37	1730	12.89
alu2_96	78	82	85	4369	3942	9.77	3837	12.18
example2_156	78	82	85	4369	3942	9.77	3837	12.18
mod5adder_66	28	26	27	353	318	9.92	317	10.20
4mod7_26	12	11	8	84	79	5.95	76	9.52
Average						**17.03**		**23.80**

Our second test suite consists of 29 benchmark circuits obtained from the RevLib library [37]. Compared with the optimization technique proposed in [32], 10 of them showed improvement, 14 of them remained same and 5 of them did not show any improvement in quantum cost. The results are given in Table 5.4, sorted based on the quantum cost improvement. The column circuit is the name of the circuit, the column RevLibGC/QC represents the gate count and quantum cost of the benchmark circuits. The gate count and quantum cost of these circuits are calculated based on the values in Table 2.3. The PrevQC column lists the quantum cost from [32]. The column NewGC/QC represents the gate count and quantum cost of the improved template matching algorithm. For the circuits listed in Table 5.4 that showed improvement, the average quantum cost reduction is 18.14%.

Table 5.4: Comparison of [32] and the improved template matching algorithm

Circuit	RevLibGC [37]	RevLibQC [37]	PrevQC [32]	NewGC	NewQC	QCImp.(%)
rd32-v0_67	2	8	12	2	8	33.33
rd32-v1_69	3	9	13	3	9	30.77
rd73_141	14	64	76	14	64	15.79
sys6-v0_144	15	62	72	15	62	13.89
sym9_147	21	94	108	21	94	12.96
rd84_143	21	98	112	21	98	12.50
4gt5_76	13	29	27	12	24	11.11
4gt13_90	14	34	32	13	29	9.38
4mod5-v0_19	5	13	11	4	10	9.09
one-two-three-v1_99	8	36	34	9	33	2.94
4gt11_82	12	16	16	12	16	0.00
4mod5-v0_21	6	19	19	6	19	0.00
miller_11	5	17	17	5	17	0.00
mod5d2_70	8	16	13	9	13	0.00
mod5mils_65	5	13	11	5	11	0.00
mod5mils_71	5	13	11	5	11	0.00
rd32-v0_66	4	12	12	4	12	0.00
rd32-v1_68	5	13	13	5	13	0.00
rd53_133	12	128	104	13	104	0.00
rd73_140	20	76	76	20	76	0.00
rd84_142	28	112	112	28	112	0.00
sym9_146	28	108	108	28	108	0.00
sys6-v0_111	20	72	72	20	72	0.00
toffoli_double_4	2	10	7	3	7	0.00
ham15_107	132	1831	1434	134	1446	-0.84
alu-v2_31	13	101	91	13	95	-4.40
4mod5-v0_18	9	25	19	10	22	-15.79
sym6_145	36	777	297	48	408	-37.37
sym9_148	210	4368	798	276	2189	-174.31
Average						**18.14**

The third test suite consists of 45 benchmark circuits obtained from RevLib [37] and the

Table 5.5: Comparison of [4], [5] and the improved template matching algorithm

Circuit	RevLibGC [37]	RevLibQC [37]	GC [4]	QC [4]	GC [5]	QC [5]	NewGC	NewQC	QCImp1(%)	QCImp2(%)
sym9_148	210	4368	154	3668	143	3433	276	2189	40.32	36.24
sym6_145	36	777	31	647	23	517	48	408	36.94	21.08
max46_240	107	5444	51	4498	52	4538	112	3632	19.25	19.96
add6_196	229	6455	179	6005	167	5534	244	4581	23.71	17.22
clip_206	174	6731	111	6535	109	6119	173	5462	16.42	10.74
life_238	107	6766	57	5740	57	5744	99	5210	9.23	9.30
sym9_193	129	14193	58	12747	63	13090	124	11929	6.42	8.87
9symml_195	129	14193	58	12747	62	13026	124	11929	6.42	8.42
alu4_201	1063	55388	523	46413	529	46764	1013	43097	7.14	7.84
mux_246	35	1078	20	804	20	804	42	742	7.71	7.71
tial_265	1041	56203	516	47145	522	47556	981	44394	5.84	6.65
apla_203	80	3438	74	3438	64	3024	78	2839	17.42	6.12
f51m_233	663	37400	358	33333	355	32882	637	30981	7.06	5.78
dc2_222	75	1886	55	1789	53	1688	78	1592	11.01	5.69
mod5adder_306	96	292	84	281	70	270	72	265	5.69	1.85
hwb5_300	88	276	80	270	67	259	67	255	5.56	1.54
in2_236	405	23802	283	23146	250	20600	394	20289	12.34	1.51
alu2_199	157	5654	87	4776	87	4611	147	4561	4.50	1.08
cycle10_293	78	202	74	199	57	186	56	184	7.54	1.08
frg1_234	212	15265	130	14737	130	14702	210	14815	-0.53	-0.77
cu_219	40	1148	31	1054	29	954	37	962	8.73	-0.84
cm150a_210	53	1096	38	822	38	822	46	830	-0.97	-0.97
sqn_258	76	2122	50	2041	51	1887	76	1923	5.78	-1.91
dist_223	185	7601	126	7288	122	6631	177	6796	6.75	-2.49
rd73_312	73	217	65	214	53	200	55	205	4.21	-2.50
rd84_313	104	304	95	301	71	275	71	285	5.32	-3.64
misex3c_243	1721	115190	1188	111258	1049	96064	1711	101512	8.76	-5.67
ham7_299	61	141	46	135	34	121	36	128	5.19	-5.79
sqr6_259	81	1033	66	1034	53	876	87	946	8.51	-7.99
rd53_131	28	119	12	104	12	104	22	113	-8.65	-8.65
table3_264	1012	80039	744	79326	577	61412	934	67157	15.34	-9.35
misex3_242	1752	119177	1199	115637	1043	99119	1749	108621	6.07	-9.59
alu3_200	94	2632	79	2512	69	2162	87	2405	4.26	-11.24
ham15_298	153	309	100	290	91	266	82	297	-2.41	-11.65
5xp1_194	85	1406	65	1327	59	1155	85	1290	2.79	-11.69
in0_235	338	20031	245	18999	218	16985	330	18988	0.06	-11.79
pm1_249	35	377	31	354	24	275	30	312	11.86	-13.45
inc_237	93	2140	72	2104	63	1745	92	2114	-0.48	-21.15
sf_275	11	51	6	44	5	41	11	51	-15.91	-24.39
dc1_220	39	416	33	419	26	249	39	416	0.72	-67.07
rd32_273	20	116	8	112	7	67	20	116	-3.57	-73.13
C7552_205	80	1728	79	1745	23	623	74	1250	28.37	-100.64
decod_217	80	1728	79	1745	21	613	74	1250	28.37	-103.92
sf_274	19	155	7	145	6	38	19	155	-6.90	-307.89
sf_276	16	152	8	146	6	27	16	152	-4.11	-462.96
Average									**13.19**	**9.40**

results are compared with two existing template matching techniques from [4] and [5]. The

results are listed in Table 5.5 sorted in order of quantum cost improvement. The column

RevLibGC/QC represents the gate count and quantum cost of the circuit obtained from

RevLib. The gate count and quantum cost of these circuits are calculated based on the

values in Table 2.3. The column NewGC/QC represents the gate count and quantum cost of

the improved template matching algorithm. Compared to [4] and [5], the average quantum

cost reduction for the top 19 circuits is 13.19% and 9.40%, respectively. sym9_148 is the best reported circuit in this list and worst in the previous list. From Table 5.4 we can see that the quantum cost of sym9_148 is 798 [32] which is much lower than all previously obtained results [4, 5, 32] and the results from the proposed approach. The same situation is also observed for the circuit sym6_145.

5.4 Summary

At the beginning of this chapter we discussed the basic template matching algorithm along with the moving rule and an example. The improved template matching algorithm is also discussed with an example. The results are compared with improved shared cube synthesis and three other existing template matching techniques.

Chapter 6

Conclusion

6.1 Conclusion

Research on reversible logic synthesis has attracted much attention due to the many possible applications. The circuits obtained from different logic synthesis approaches may not be optimal and post-synthesis optimization techniques are used to improve the gate count and quantum cost of the circuits. Template matching is one post-synthesis technique used to improve the gate count and quantum cost of a circuit. The proposed approach considers both positive and negative control Toffoli gates to develop templates. In developing templates we consider the various ways in which two Toffoli gates with the same target line can appear in a circuit. We propose two new templates for positive and negative control Toffoli gates (template 4 and template 7). Template 4 can be applied to two $\geq$ 3-bit Toffoli gates with controls on the same line while template 7 can be applied to two different size $\geq$ 3-bit Toffoli gates. In our proposed template matching algorithm we rank the templates based on the savings in quantum cost and select templates to apply based on their rank. 86 of the 110 circuits generated by the improved shared cube synthesis approach [23] show improvement in quantum cost after applying templates. Results show that the proposed templates can reduce quantum cost up to 53.58% (on average, 21.34%).

6.2 Future Work

Future work may pursue several avenues related to this work, including identifying additional templates, particularly for Toffoli gates with different target lines, and also im-

proving the template matching algorithm. Of course, the issue of template matching with negative controls has not yet been thoroughly studied, and as we pursue this work a broader investigation will also be required.

All positive control Toffoli gate templates of size 7 and some templates of size 9 are listed in [16]. Other templates for Toffoli gates with positive and negative controls were proposed in [2, 4, 5, 32]. Finding and classifying the complete set of templates for positive and negative control Toffoli gate is another direction for further research. Garbage/line count reduction through the application of templates is also a possible research direction.

Bibliography

[1] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, November 1973.

[2] Xueyun Cheng, Zhijin Guan, Wei Wang, and Lingling Zhu. A simplification algorithm for reversible logic network of positive/negative control gates. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, pages 2442–2446, May 2012.

[3] R. Cuykendall and D. R. Andersen. Reversible optical computing circuits. *Optics Letters*, 12(7):542544, 1987.

[4] K. Datta, G. Rathi, R. Wille, I. Sengupta, H. Rahaman, and R. Drechsler. Exploiting negative control lines in the optimization of reversible circuits. In Gerhard W. Dueck and D.Michael Miller, editors, *Reversible Computation*, volume 7948 of *Lecture Notes in Computer Science*, pages 209–220. Springer Berlin Heidelberg, 2013.

[5] K. Datta, I Sengupta, and H. Rahaman. A post-synthesis optimization technique for reversible circuits exploiting negative control lines. *Computers, IEEE Transactions on*, PP(99):1–1, 2014.

[6] K. Fazel, M. A. Thornton, and J. E. Rice. Esop-based Toffoli gate cascade generation. In *Communications, Computers and Signal Processing, 2007. PacRim 2007. IEEE Pacific Rim Conference on*, pages 206–209, 2007.

[7] E. Fredkin and T. Toffoli. Conservative logic. *Int'l J. of Theoretical Physics*, 21:219–253, 1982.

[8] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 44(1.2):261–269, 2000.

[9] M. M. Mano and M. D. Ciletti. *Digital Design With An Introduction to the Verilog HDL*. Pearson Education, 2013.

[10] D. Maslov. Reversible logic synthesis benchmarks page, http://www.cs.uvic.ca/˜dmaslov/.

[11] D. Maslov. *Reversible Logic Synthesis*. PhD thesis, University of New Brunswick, 2003.

[12] D. Maslov and G. W. Dueck. Reversible cascades with minimal garbage. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 23(11):1497–1509, 2004.

[13] D. Maslov and G. W. Dueck. Reversible cascades with minimal garbage. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 23(11):1497–1509, 2004.

[14] D. Maslov, G. W. Dueck, and D. M. Miller. Fredkin/toffoli templates for reversible logic synthesis. In *Computer Aided Design, 2003. ICCAD-2003. International Conference on*, pages 256–261, Nov 2003.

[15] D. Maslov, G. W. Dueck, and D. M. Miller. Toffoli network synthesis with templates. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(6):807–817, 2005.

[16] D. Maslov, G. W. Dueck, and D. M. Miller. Techniques for the synthesis of reversible toffoli networks. *ACM Trans. Des. Autom. Electron. Syst.*, 12(4), September 2007.

[17] D. Maslov, G. W. Dueck, D. M. Miller, and C. Negrevergne. Quantum circuit simplification and level compaction. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(3):436–444, March 2008.

[18] R. C. Merkle. Reversible electronic logic using switches. *Nanotechnology*, 4:2140, 1993.

[19] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Design Automation Conference, 2003. Proceedings*, pages 318–323, 2003.

[20] D. M. Miller and M. A. Thornton. Qmdd: A decision diagram structure for reversible and quantum circuits. In *Multiple-Valued Logic, 2006. ISMVL 2006. 36th International Symposium on*, pages 30–30, May 2006.

[21] D. M. Miller, R. Wille, and R. Drechsler. Reducing reversible circuit cost by adding lines. In *Multiple-Valued Logic (ISMVL), 2010 40th IEEE International Symposium on*, pages 217–222, 2010.

[22] N. M. Nayeem. Synthesis and Testing of Toffoli Circuits. Master's thesis, University of Lethbridge, 2011.

[23] N. M. Nayeem and J. E. Rice. Improved ESOP-based synthesis of reversible logic. In *Proceedings of the 2011 Reed-Muller Workshop*, pages 57–62, 2011.

[24] N. M. Nayeem and J. E. Rice. A shared-cube approach to ESOP-based synthesis of reversible logic. *Facta Universitatis Series: Electronic Engineering*, 24(3):385–402, December 2011.

[25] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.

[26] W. D. Pan and M. Nalasani. Reversible logic. *Potentials, IEEE*, 24(1):38–41, Feb 2005.

[27] A. Peres. Reversible logic and quantum computers. *Physical Review A*, 32(6):3266–3276, 1985.

[28] M. Z. Rahman and J. E. Rice. Templates for positive and negative control toffoli networks. In Shigeru Yamashita and Shin-ichi Minato, editors, *Reversible Computation*, volume 8507 of *Lecture Notes in Computer Science*, pages 125–136. Springer International Publishing, 2014.

[29] Y. Sanaee. Generating Toffoli networks from ESOP expressions. Master's thesis, University of New Brunswick, 2010.

[30] Y. Sanaee and G. W. Dueck. Generating toffoli networks from esop expressions. In *Communications, Computers and Signal Processing, 2009. PacRim 2009. IEEE Pacific Rim Conference on*, pages 715–719, Aug 2009.

[31] Y. Sanaee and G. W. Dueck. Esop-based toffoli network generation with transformations. In *Multiple-Valued Logic (ISMVL), 2010 40th IEEE International Symposium on*, pages 276–281, May 2010.

[32] Z. Sasanian. *Technology Mapping and Optimization for Reversible and Quantum Circuits*. PhD thesis, University of Victoria, 2012.

[33] C. E. Shannon. A symbolic analysis of relay and switching circuits. *American Institute of Electrical Engineers, Transactions of the*, 57(12):713–723, Dec 1938.

[34] F. Somenzi. Cudd: CU decision diagram package release 2.3.1. University of Colorado at Boulder, 2001.

[35] T. Toffoli. Reversible computing. Tech Memo LCS/TM-151, MIT Lab for Comp. Sci, 1980.

[36] R. Wille and R. Drechsler. Bdd-based synthesis of reversible logic for large functions. In *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pages 270–275, 2009.

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