J. of Mult.-Valued Logic & Soft Computing, Vol. 29, pp. 1–23
Reprints available directly from the publisher
Photocopying permitted by license only

2017 Old City Publishing, Inc.
Published by license under the OCP Science imprint,
a member of the Old City Publishing Group.

Fast Synthesis of Reversible Circuits using a Sorting Algorithm and Optimization

ÖMERCAN SUSAM AND MUSTAFA ALTUN

Department of Electronics and Communication Engineering,
Istanbul Technical University, Turkey
E-mail: susam@itu.edu.tr; altunmus@itu.edu.tr

Accepted in final form: June 28, 2016.

This paper studies synthesis and optimization of reversible circuits composed of Toffoli gates with negative and positive control lines. The proposed synthesis algorithm performs sorting among optimal implementations of certain functions - called as *essential functions* - to implement any reversible Boolean function. Essential functions comprise very small amount of all functions. For example, to implement 3 bit circuits, 28 essential functions out of all 40320 functions are needed. The proposed optimization algorithm considers both reversible and quantum circuit costs. First, reversible cost is reduced by considering adjacent gate pairs. Then, inner quantum structures of the gates are investigated and quantum optimization is performed. The proposed algorithms are evaluated on benchmark circuits in comparison with the results in the literature.

Keywords: Design, algorithms, optimization, reversible.

1 INTRODUCTION

As Moore's Law has kept its validity for decades, not only the number of transistors on a chip but also the chip's power dissipation have continually doubled for every 18 to 24 months. Nowadays this trend has reached a critical point at which new technologies enabling low-power designs are highly desired. Reversible circuits are considered as a strong candidate technology in this regard. Reversible logic offers low-power designs [3] – zero power dissipation is theoretically possible if all operations during computation

are reversible [2] – that has motivated several emerging technologies including quantum computing that is solely based on unitary matrix based reversible operations, and DNA computing that is used to implement universal reversible gates [16]. With the advent of these technologies, reversible logic synthesis has found a renewed interest aiming on efficient algorithms to implement reversible circuits by considering run time and circuit size [7, 17]. Although finding optimal or near-optimal size circuits is an ultimate goal in synthesis, it is costly for reversible circuits. It has been achieved for all input-output combinations only up to four bits in the literature [8, 17]. For higher bit counts, it is a non-trivial problem [8]. If we consider recent experiments where 84 qubits are used [4], we easily see the impracticability of synthesizing optimal or near-optimal circuits that would take years. Motivated by these findings and with our previous work [21], we aim at a fast synthesis algorithm in this study.

For our synthesis algorithm, we face a decision of whether or not to use garbage outputs. Garbage outputs are used especially for synthesizing functions with high number of bits [22] that offers design flexibility and gate cost reduction at the expense of additional bit lines [24]. In this study, we do not use garbage outputs; we use minimum number of bit lines. Our synthesis algorithm is straightforward and fast without requiring additional bit lines. It is a well known fact that any reversible function can be implemented without a need of a garbage bit by using the NCT (**N**OT, **C**NOT, **T**offoli) gate library. Along with the NCT library, we also use the MCT (**M**ultiple **C**ontrol **T**offoli) gate library for which the number of the control lines is not limited. These libraries include gates with positive control lines. To extend our synthesis approach with negative-polarity control lines, we also use the MPMCT (**M**ixed **P**olarity **M**ultiple **C**ontrol **T**offoli) gate library.

Our synthesis process has two steps. In the first step, we use a permutation based algorithm to implement certain functions called as "essential functions". Here, the algorithm is optimal in the sense that it uses a minimum number of reversible gates. In the second step, we implement given target functions by using the essential function implementations obtained in the first step. Here, we use a fast sorting algorithm. Our overall synthesis approach is greedy as opposed to dynamic or optimal. It can also be considered as a look-up table approach with performance metrics of circuit size, synthesis time, and space required to store look-up tables. As opposed to targeting on the circuit size to find optimal or near-optimal results [8], we aim to reduce the storing space and correspondingly the synthesis time. We only store essential function implementations that comprise very small amount of all implementations. For example, to synthesize all 4 bit circuits (total of 16!), our approach needs only 120 essential function implementations. Remaining $16! - 120$ are synthesized by sorting them.

The synthesis process is followed by optimization for which we perform gate reductions by checking adjacent gate pairs. In the literature, reductions using gate pairs and templates are extensively studied for the NCT and the MPMCT libraries [6, 10, 19]. In this study, we specialize those templates and reduction rules to be applicable to the neighbour gate pairs of the circuit implementations of essential functions. We also define cost reduction techniques for the circuits based on the **NCV** (**N**OT, **C**NOT, controlled-V and controlled-$V^\dagger$) gate library.

The total time required for our synthesis and optimization processes is the time needed for synthesizing essential functions, sorting, and optimization. Because of the essential functions' fewness, and sorting-optimization algorithms' speed, the total time is very small compared to the studies in the literature.

The paper is organized as follows. In Section 2, we introduce background information of reversible circuits and quantum gates. In Section 3, we present our synthesis algorithm based on essential functions and sorting. In Section 4, we present our optimization method to reduce reversible and quantum costs. In Section 5 and 6, we give experimental results and conclusions, respectively.

2 PRELIMINARIES

2.1 Reversible Funcitons

While a conventional Boolean function always carries a one bit information (0 or 1) that is independent of the number of input bits, a **reversible Boolean function** carries information with using the same number of input and output bits. For reversible functions, each input bit combination results in a unique output bit combination; the reverse of this is also true because of the reversibility. This means that the input values can be deduced by looking at the output values of the reversible function. For example, a NOT function, implemented with a conventional inverter (logic gate), is reversible since we can undo it. On the other hand, NAND, NOR, and XOR functions with multiple inputs and a single output are irreversible. Bijection functions in mathematics is also a great example to understand reversibility. In these functions, input and output sets have the same number of elements and each element has only one counterpart in the other set.

2.2 Reversible Gates & Circuits

A reversible function can be realized by a **reversible circuit** consisting of reversible gates [15]. In this study we use three types of gate libraries that are the NCT (**N**OT, **C**NOT, **T**offoli), the MCT (**M**ultiple **C**ontrol **T**offoli), and

the MPMCT (**M**ixed **P**olarity **M**ultiple **C**ontrol **T**offoli) libraries. Definition of the gates:

- **N**OT: a 1-bit gate performing 1 bit NOT operation in any case.
- **C**NOT: a 2-bit gate performing 1 bit NOT operation on its target bit iff its control bit is 1.
- **T**offoli: a 3-bit gate performing 1 bit NOT operation on its target bit iff its control bits are both 1.
- **M**ultiple **C**ontrol **T**offoli: an n-bit gate, $n = 1,\ 2,\ 3,\ 4,$ ..., performing 1 bit NOT operation on its target bit iff both of its control bits are 1.
- **M**ixed **P**olarity **M**ultiple **C**ontrol **T**offoli: an n-bit gate, $n = 1,\ 2,\ 3,\ 4,$ ..., performing 1 bit NOT operation on its target bit iff all of its positive control bits are 1 and all of its negative control bits are 0.

Note that an MCT gate becomes a NOT, a CNOT, and a Toffoli gate if $n = 1$, $n = 2$, and $n = 3$, respectively. Circuit representations of the gates are given in Figure 1 where symbols $\bullet$, $\circ$, and $\oplus$ denote positive control, negative control, and target lines, respectively.

2.3 Circuit Cost

In this study, we calculate **circuit costs** in two different ways. For the first one, we consider each gate cost in the NCT, the MCT, and the MPMCT libraries as one that is the **"Reversible Cost"**.

Definition 1. ***Reversible cost*** *is the total number of reversible gates used in a given circuit.*

For the second one, we consider sub-gates (elementary gates or elementary quantum operations [1]). We take each sub-gate cost in the **NCV**-111 (**N**OT,

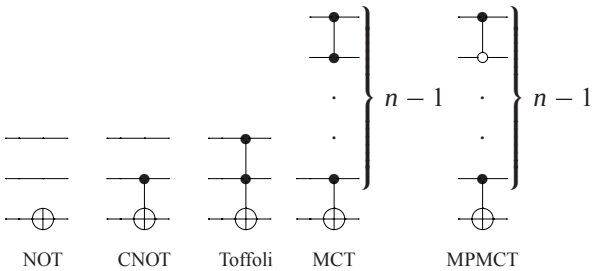

FIGURE 1
Circuit representations of NOT, CNOT, Toffoli, MCT, and MPMCT gates.

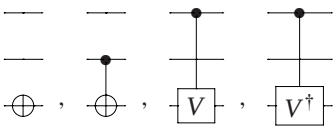

FIGURE 2
Circuit representations of NOT, CNOT, controlled-V and controlled-$V^\dagger$ gates.

CNOT, controlled-V and controlled-$V^\dagger$) library as one to find the **"Quantum Cost"**.

Definition 2. ***Quantum cost*** *is the total number of elementary quantum operations (sub-gates) used to realize a given circuit. [23]*

We use the NCV-111 library among different quantum libraries/metrics since it is a well accepted one [11]. In this metric, "111" refers that each NOT, CNOT, controlled-V and controlled-$V^\dagger$ gate has a cost of one. Figure 2 illustrates NCV quantum gates that operate on quantum bits (qubits). As opposed a conventional bit being either 0 or 1, a qubit is a linear superposition of the states $|0\rangle$ and $|1\rangle$. General representation is $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ where α and β are complex numbers satisfying that $|\alpha|^2 + |\beta|^2 = 1$.

Using the NCV-111 library necessitates to use 4 possible states for a qubit [19]:

$$
\begin{aligned}
|0\rangle &= 1|0\rangle + 0|1\rangle &&= \begin{bmatrix} 1 & 0 \end{bmatrix} \\
|+\rangle &= \frac{1-i}{2}|0\rangle + \frac{1+i}{2}|1\rangle &&= \begin{bmatrix} \frac{1-i}{2} & \frac{1+i}{2} \end{bmatrix} \\
|1\rangle &= 0|0\rangle + 1|1\rangle &&= \begin{bmatrix} 0 & 1 \end{bmatrix} \\
|-\rangle &= \frac{1+i}{2}|0\rangle + \frac{1-i}{2}|1\rangle &&= \begin{bmatrix} \frac{1+i}{2} & \frac{1-i}{2} \end{bmatrix}
\end{aligned}
$$

that results in input-output transition tables for NOT, V, and $V^\dagger$ gates given in Table 1.

Figure 3 shows a Toffoli gate realization using NCV gates. Quantum costs of gates in the NCT and MPMCT libraries, up to 3 bits, are given in Figure 4 [1, 12, 25].

NOT	
Input	Output
$\|0\rangle$	$\|1\rangle$
$\|+\rangle$	$\|-\rangle$
$\|1\rangle$	$\|0\rangle$
$\|-\rangle$	$\|+\rangle$

V	
Input	Output
$\|0\rangle$	$\|+\rangle$
$\|+\rangle$	$\|1\rangle$
$\|1\rangle$	$\|-\rangle$
$\|-\rangle$	$\|0\rangle$

$V^\dagger$	
Input	Output
$\|0\rangle$	$\|-\rangle$
$\|+\rangle$	$\|0\rangle$
$\|1\rangle$	$\|+\rangle$
$\|-\rangle$	$\|1\rangle$

TABLE 1
Input-output transition tables for NOT, V, and $V^\dagger$ gates.

3 SYNTHESIS

Our synthesis algorithm is greedy as opposed to dynamic or optimal. It has two steps that are implementing essential functions and performing sorting. After achieving essential function implementations with the NCT, MCT, and MPMCT gate libraries, we sort them to implement any target function. In sorting, our algorithm determines which essential function implementations are needed and how to efficiently use them.

3.1 Essential Functions and Their Implementations

Definition 3. *An **essential function** is a reversible Boolean function such that its truth table has exactly two rows having different input and output bit values, i.e., all other rows have identical inputs and outputs.*

Consider a truth table such that input and output bit values are all identical. If we interchange any two output rows without changing others, the result is an essential function. Table 2 shows three 3-bit essential functions; bold rows represent interchanged rows. The total number of essential functions for a certain bit size n can be derived as

$$\binom{2^n}{2} = \frac{2^n!}{2!(2^n-2)!} = 2^{n-1} * (2^n - 1) \tag{1}$$

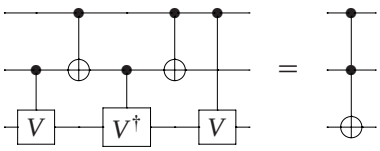

FIGURE 3
Implementation of a Toffoli gate using the NCV library; quantum cost is 5.

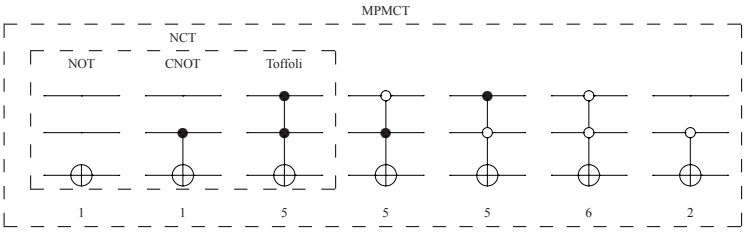

FIGURE 4
Reversible gates of the NCT and the MPMCT libraries up to 3 bits and their quantum costs below them.

that is favourably low compared to the total number of $2^n!$ functions. Table 3 compares the formulas for different bit sizes up to 6.

The reason of using essential functions is consistent with our main goal of achieving a fast greedy algorithm as opposed to exhaustive search based

f_{E1}	
Input	Output
cba	c'b'a'
000	**001**
001	**000**
010	010
011	011
100	100
101	101
110	110
111	111

f_{E2}	
Input	Output
cba	c'b'a'
000	000
001	**111**
010	010
011	011
100	100
101	101
110	110
111	**001**

f_{E3}	
Input	Output
cba	c'b'a'
000	000
001	001
010	**110**
011	011
100	100
101	101
110	**010**
111	111

TABLE 2
Truth tables of 3-bit essential functions f_{E1}, f_{E2}, and f_{E3} .

Bit Size	# of Essential Functions	# of Total Functions
2	6	24
3	28	40320
4	120	20922789888000
5	496	2.613308e + 35
6	2016	1.268869e + 89

TABLE 3
Number of essential functions and total number of functions according to bit size.

algorithms. In general, it is desired for a greedy algorithm that each local synthesis problem should be efficiently solved, preferably with an optimal solution and local solutions should be easily integrated in order to solve the global problem. For our algorithm, local and global synthesis problems correspond to implementing essential and given target reversible functions, respectively. Optimal implementation of an essential function is relatively fast since only two input bit assignments result in different output values that allows us to eliminate considerable amount of cases. More importantly, discussed in the next section "Sorting", implementations of essential functions can be directly integrated; for a given reversible function, the total number of the required essential function is always the same but the ordering of their implementation is arbitrary.

We use a simple algorithm to synthesize essential functions with minimal circuit sizes considering their reversible costs. Our algorithm is based on permutation trials. At first, essential functions are determined and the essential function library is created. Then, optimal circuit implementations of the functions are achieved. Each essential function has three different optimal implementations corresponding to three different gate libraries NCT, MCT, and MPMCT. Permutation trials start circuits having a reversible cost of one (one gate) and continue by gradually increasing the cost until obtaining an optimal solution. The optimal circuit implementation, which may not be a unique solution, is stored and the algorithm picks the next essential function to be realized. Figure 5 represents a flow chart diagram of the algorithm.

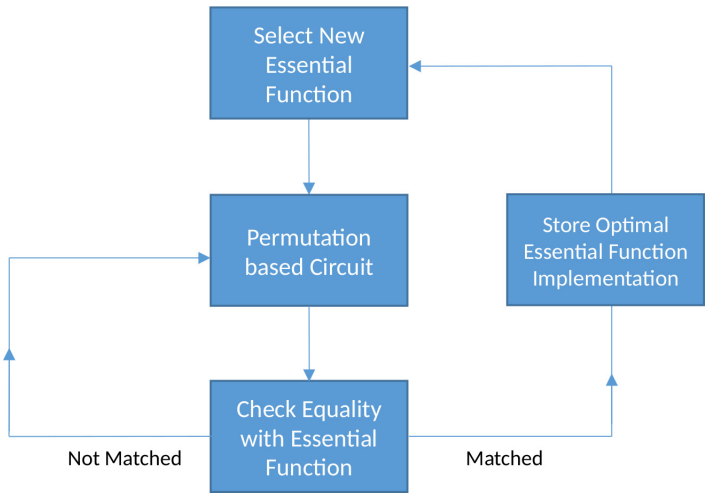

FIGURE 5
Flow chart of the algorithm for essential function implementations.

3.2 Sorting

Sorting process begins after the essential function implementations are stored. An example of this process is shown in Figure 6. Here, f_1 and f_2 are essential functions used to obtain a target function F. The truth table of F has 8 rows; 4 of them are mismatched, having different input and output bit values. To match them, pairwise interchanging of output rows is done by the essential functions f_1 and f_2. Bold lines in the functions' truth tables indicate the interchanged rows. Since every distinct essential function, for this case f_1 or f_2, only deals with two rows without changing others, the ordering of the implementations of essential functions can be arbitrary. It means that F could also be implemented by changing the circuit order of f_1 and f_2.

Fixing mismatched rows is indeed a sorting problem among output rows. We investigate different sorting algorithms in reversible circuit design perspective and choose a selection sort algorithm that uses row by row checking of mismatched input and output bits. Row by row checking and fixing can be efficiently done by reversible circuits, in our case by circuit implementations of essential functions. However, operations like dividing and sliding used in other sorting algorithms necessitates relatively larger circuits. For example, a merge sort algorithm divides a target set into two subsets and

(a)

f_1	
In	Out
cba	cba
000	000
001	001
010	010
011	**101**
100	100
101	**011**
110	110
111	111

+

f_2	
In	Out
cba	cba
000	**100**
001	001
010	010
011	011
100	**000**
101	101
110	110
111	111

=

F	
In	Out
cba	cba
000	100
001	001
010	010
011	101
100	000
101	011
110	110
111	111

(b)

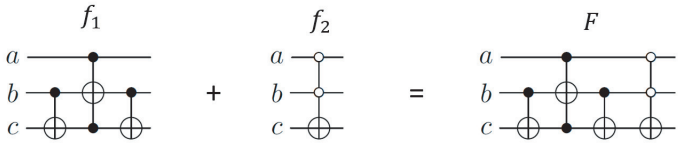

FIGURE 6
Implementations of essential functions f_1, f_2, and the target function F: (a) truth tables, (b) circuits.

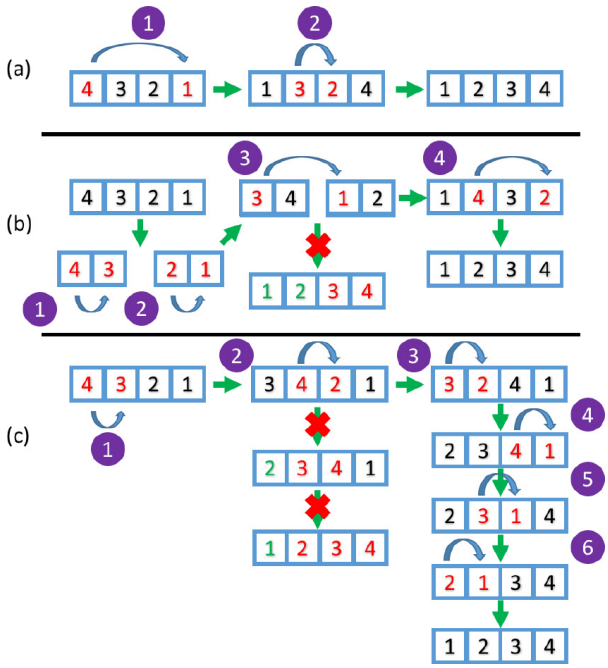

FIGURE 7
Sorting process with essential functions: (a) selection sort, (b) merge sort, (c) insertion sort. Numbers associated with arrows represent the number of essential functions needed. Crossed arrows show the operations that cannot be directly achieved by using essential functions.

sorts them separately. After subset sorting is completed, they are merged and the same procedure repeats until finding a final solution. Here, each of these operations, especially the dividing operation, is quite costly. Similarly, sliding process used in an insertion sort algorithm has the same handicap. Figure 7 illustrates differences between sorting algorithms by showing the number of required essential function implementations. For this specific example, the selection/merge/insertion algorithms require 2/4/6 essential function implementations, respectively.

Given a target function F with m mismatched rows, the selection sort algorithm first fixes one of the m rows and continues to fix the rows one by one. Here, the number of possible solutions is upper bounded by $m!$. For all of these solutions, the total number of interchanging operations or corresponding essential functions is constant, as stated by the following theorem.

Theorem 1. *Consider a target function F with mismatched rows in its truth table. In order to implement F, the required number of essential functions is always the same, regardless the sequence of mismatched rows to be fixed.*

F				
000	0	0	0	0
111	7	1	1	1
001	1	7	2	2
010	2	2	7	3
100	4	4	4	4
101	5	5	5	5
110	6	6	6	6
011	3	3	3	7
Essential Functions	1-7	2-7	3-7	-
Function Cost	4	4	1	-
Total Cost	9			

(a)

F				
000	0	0	0	0
111	7	3	1	1
001	1	1	3	2
010	2	2	2	3
100	4	4	4	4
101	5	5	5	5
110	6	6	6	6
011	3	7	7	7
Essential Functions	7-3	1-3	2-3	-
Function Cost	1	2	2	-
Total Cost	5			

(b)

TABLE 4
Two different solutions with different costs to synthesize F using a selection sort algorithm.

Before proving the theorem we elucidate it with an example shown in Table 4. In the table, mismatched and fixed rows are represented by red and green colors, respectively; there are 4 mismatched rows initially. Table 4(a) shows a solution with a sorting sequence that starts at the first row and progresses to the bottom one by one. Table 4(b) represents a solution with a sorting sequence that results in the smallest reversible cost. As stated in Theorem 1 both solutions use the same number of 3 essential functions.

We use the following definition and lemma for the proof of the theorem.

Definition 4. *A **closed group** of mismatched rows is defined such that removal of any row(s) from the group makes it unmatchable, i.e., at least one of the matching elements exists only once.*

Example of closed groups are shown in Figure 8. Here, there are total of 8 mismatched rows that cannot constitute a single closed group since removing the top 5 or bottom 3 rows does not make it unmatchable. However, "Group a" and "Group b" are closed groups.

Lemma 1. *Consider a closed group of k mismatched rows. The number of required essential functions to match all k rows is* $k - 1$.

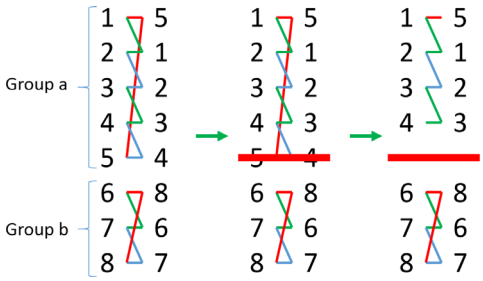

FIGURE 8
Two different closed groups. Removing the fifth row from "Group a" results in unmatchable rows: 4 on the left side and 5 on the right are unmatchable.

Proof. The proof is by contradiction. In sorting process, each essential function fixes either one or two mismatched rows. Additionally, the last mismatched row pair is always fixed using a single essential function. Suppose that $k-2$ or fewer essential functions are used; it means that in sorting process, excluding the last two mismatched rows, one essential function is used (at least once) to fix two mismatched rows. This row pair is indeed a closed group that can not exist initially (before sorting starts); if it did, the initial closed group would consist of multiple closed groups that is against the definition of a closed group in Definition 4. Therefore this row pair should be formed during the sorting process that results in an initial distinct closed group as illustrated in Figure 9. In other words, if it happens then the initial closed group consists of multiple closed groups. Again this is against the definition of a closed group. So we have a contradiction. □

The proof of Theorem 1 is stated below.

Proof. Suppose that F has m mismatched rows in its truth table. Also suppose that these rows are represented in g closed groups. Here, m and g are

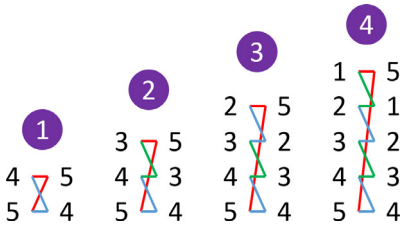

FIGURE 9
A closed group of two rows (in 1) always results in a distinct closed group (in 2, 3, 4, ...) using essential functions in backward direction.

fixed numbers for a certain function. Using Lemma 1, we know that the number of essential functions required for each closed group is the number of each group's rows minus 1. Therefore, the total number of the required essential functions is fixed and always $m - g$. □

Comparing the two solutions given in Table 4, as expected from Theorem 1, both uses the same number of essential functions. However, the type of essential functions and related costs are quite different. Total costs of 3 essential functions are 5 and 9 for the solutions in (a) and (b), respectively. This underlines the importance of selecting an adequate sequence of mismatched rows for a given target function. For this purpose, we offer an approach called as "pick smallest". We evaluate our approach by comparing it with "top-down" and "optimal" approaches explained below. Worst case complexities for the approaches are calculated by considering the time spent on row checking procedure. Note that m represents the total number of mismatched rows upper bounded by 2^n for a bit size of n.

Pick Smallest - ***Proposed***

Our approach checks all mismatched rows and their related costs. The lowest cost, assigned to a certain mismatched row, is selected and the corresponding mismatched row is fixed. The checking process is repeated until there is no mismatched rows left. Since our approach needs to check all mismatched rows in each checking, the time complexity is $\mathcal{O}(m^2)$.

Top-Down

Top-down approach fixes mismatched rows one by one from top to bottom. Thus, its time complexity is $\mathcal{O}(m)$. Indeed, this approach represents all of the approaches that use a certain sequence for all target functions. For example, an approach with checking rows one by one from bottom to top will have the same complexity and very close run time and average cost values as the top-down approach has.

Optimal

An optimal sequence can be found by checking all possible $m!$ sequences. Therefore, the time complexity is $\mathcal{O}(m!)$.

While the top-down approach uses the same sequence of mismatched rows for every target function, the proposed and the optimal approaches use different sequences for different target functions that results in better reversible costs. Table 5 compares the approaches considering total run times and average reversible costs for all 3-bit functions. Although the reversible cost of our

	Pick Smallest - Proposed	Top-Down	Optimal
Reversible Cost	15.75	16.69	14.48
Time (s)	0.48	0.47	310
Complexities	$\mathcal{O}(m^2)$	$\mathcal{O}(m)$	$\mathcal{O}(m!)$

TABLE 5
Sequence comparison regarding total run times and average reversible costs for all 3-bit functions.

approach is slightly larger than that of the optimal approach, our approach overwhelms the others considering the runtime performances.

4 OPTIMIZATION

The synthesis process explained in Section 3 is followed by the optimization for which we perform reductions by checking adjacent gate pairs. If a gate pair with a lower cost is found for a certain part of the circuit then the cost reduction is achieved. We construct our gate pairs in two ways that are for reducing reversible and quantum circuit costs. In both ways, the proposed gate pairs include only two gates. More complex optimization techniques based on template matching algorithms with considering higher number of gates could have been constructed. This would slightly improve the circuit cost, but at the same time severely harm our main purpose of achieving a fast algorithm.

4.1 Reversible Cost Optimization

As a nature of the proposed sorting sequence, essential functions are used one after the other that might form non-optimal joint circuit parts consisting of two gates. When this formation is found, we replace the circuitry with its optimal equivalent. For this purpose, we construct all possible gate pair forms as shown in Figure 10. These are simple and easily applicable gate pairs that are not costly in terms of the runtime. Note that the number of the joints is one fewer than the number of essential functions. For each joint, there might be two or even more gate reductions that is explicitly shown in Figure 11.

4.2 Quantum Cost Optimization

After optimizing the reversible costs using the gate pairs based on the NCT, MCT, and MPMCT libraries, here we perform a second optimization process that aims to reduce the quantum cost. We offer gate pairs based on NCV quantum gates. For this purpose, we first implement reversible gates in optimal sizes. We show that Toffoli gates with two positive control lines

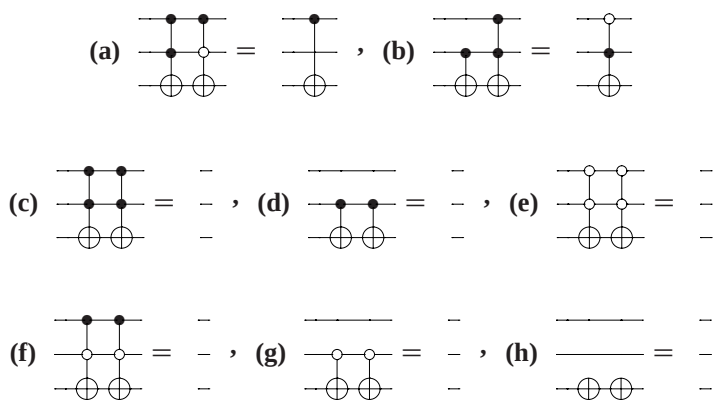

FIGURE 10
All possible two-gate gate pair forms (up to 3 bits) for reversible cost reduction: (a) and (b) for one-gate reduction, (c) through (h) for two-gate reduction.

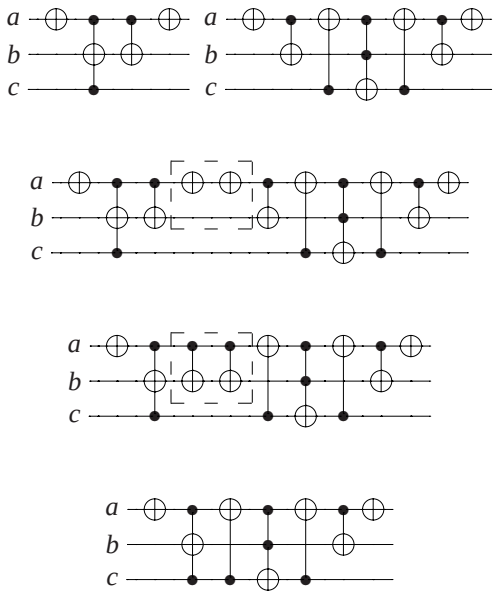

FIGURE 11
Reduction with identical neighbour gates, 4 gates removed from the circuit.

(Toffoli), negative-positive control lines (Mixed Polarity Toffoli), and two negative control lines (Negative Toffoli) require 5, 5, and 6 gates, respectively, in their optimal implementations with NCV [5]. Next, we search for non-optimal joint circuit parts consisting of two gates and replace them with

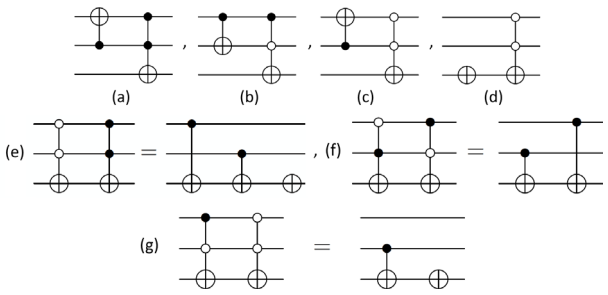

FIGURE 12
All possible 3-bit gate pair forms for quantum cost reduction.

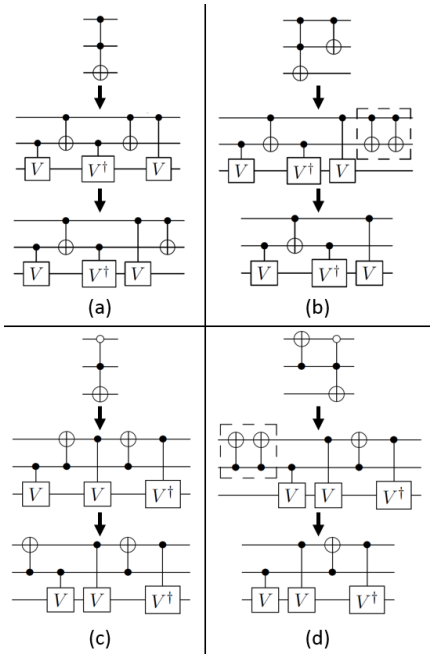

FIGURE 13
(a) Toffoli gate with its inner quantum structure. (b) gate pair using a Toffoli gate. (c) Mixed Polarity Toffoli gate with its inner quantum structure. (d) gate pair using a Mixed Polarity Toffoli gate.

their optimal equivalents. We construct all possible 7 gate pair forms as shown in Figure 12. The gate pairs in Figure 12(a) through (d) result in a cost reduction of two, corresponding to two NOT gates. Examples shown in Figure 13 fall into this category. The gate pairs in Figure 12(e), (f), and (g) result in a cost reduction of 8. To elucidate this, consider the circuits in

12(e). The two-gate circuit and its equivalent optimal three-gate circuit have quantum costs of 11 (6+5) and 3 (1+1+1), respectively, that results in a cost reduction of 8 (11-3).

Since there are different optimal implementations of Toffoli gates, we need to select proper ones to achieve a maximum quantum cost reduction. Figure 13 summarizes our optimization process. First, Toffoli gates are expanded to their quantum circuit structures and proper optimal implementations are selected (Figure 13(a), Figure 13(c). Then scanning process begins to find conditions having identical neighbour gates; the process repeats until there is no such condition left (Figure 13(b), Figure 13(d).

5 EXPERIMENTAL RESULTS

Implementations are realized in C. All experiments run on a 3.40 GHz Intel Core i7 CPU (only single core used) with 8.00 GB memory.

In this section run times are calculated by excluding times needed to synthesize essential functions. The reason is that we synthesize essential functions and store them once that is a fixed time.

Table 6 shows results of our synthesis method in comparison with the optimal/near-optimal methods in the literature considering all 3-bit reversible functions. Reversible costs are calculated by counting each gate cost as one using the NCT and the MPMCT libraries. Quantum costs are calculated by counting each gate cost as one using the NCV library. As a reminder, Figure 4 summarizes quantum cost values for all gates from the NCT and the MPMCT libraries up to 3 bits. In Table 6, the numbers under the library names represent the number of functions implemented with corresponding reversible costs. For example, the first row tells that 26 reversible gates are used to implement 10 different functions using the NCT library. In the table, an average quantum cost of 13.88 is calculated by directly applying cost conversions presented in Figure 4 [20]. Indeed, this value can be improved with applying our quantum optimization method; we improve it by 5.61%, from 13.88 to 13.10.

Table 6 demonstrates that using the MPMCT library instead of the conventional NCT library not only improve reversible and quantum costs, as expected, but also reduces the run times. From Theorem 1, we know that the number of essential functions to implement a specific target function is always same that is independent of the used libraries. It means that the time required for sorting and optimization steps is almost same for both libraries – both steps have a linear time relationship with the number of essential functions. However, the time required to implement essential functions is different; it is smaller for the MPMCT due to having smaller circuits.

Reversible Cost	**Proposed**		Optimal/Near-Optimal	
	NCT	MPMCT	NCT [20]	MPMCT [18]
26	10			
25	31			
24	145			
23	238			
22	682			
21	1031			
20	1625			
19	2720	9		
18	3129	11		
17	4022	156		
16	4383	224		
15	4179	582		
14	4126	1422		
13	3528	2388		
12	3104	3690		
11	2389	5509		
10	1772	6493		
9	1203	5906		
8	818	5007	577	
7	531	3966	10253	
6	322	2623	17049	3236
5	181	1400	8921	20480
4	80	623	2780	13282
3	43	232	625	2925
2	18	66	102	369
1	9	12	12	27
0	1	1	1	1
Average Reversible Cost	14.82	9.50	5.86	4.57
Average Quantum Cost	30.82	25.35	13.88	NA
Total Time (s)	0.47	0.26	40	NA

TABLE 6
All 3-bit reversible function implementations: the number of implemented functions with certain reversible costs for the proposed and the optimal/near-optimal algorithms.

Comparing our synthesis algorithm with the optimal/near-optimal ones in Table 6, we see that our run times are always better at the cost of circuit size. This is an expected result. Here, an important point is that our algorithm can effectively work for higher bits. However, the optimal/near-optimal synthesis method is not practically applicable even for 5 bit circuits. To further justify the speed of our algorithm, we compare our synthesis method with considerably fast implementation techniques in the literature as follows.

We consider the Transformation-Based Synthesis (**TBS**) technique proposed by [13]. We compare our method with the TBS method in two

Bit Size	# of Essential Functions = 2		# of Essential Functions = 3	
	Proposed	TBS [13]	**Proposed**	TBS [13]
2	4.00	3.00	5.22	2.44
3	6.69	6.83	8.82	7.64
4	9.47	13.14	15.58	21.86
5	12.20	23.97	21.07	45.68
Total Average	11.86	22.70	16.01	27.43

TABLE 7
Average reversible costs for our method and the TBS method considering all functions (total of 13799) up to 5 bits to be implemented with 2 and 3 essential functions.

Name	Bit Size	**Proposed**				TBS [13]		
		# of Essential Functions	Reversible Cost	Quantum Cost	Time (s)	Reversible Cost	Quantum Cost	Time (s)
4b15g_1	4	11	50	238	$0.116.10^{-3}$	25	103	0.01
4b15g_2	4	12	60	258	$0.206.10^{-3}$	27	149	0.01
4b15g_3	4	12	50	230	$0.113.10^{-3}$	30	148	0.01
4b15g_4	4	11	47	215	$0.161.10^{-3}$	30	152	0.01
4b15g_5	4	10	48	208	$0.187.10^{-3}$	31	145	0.01
3_17	3	4	16	32	$0.068.10^{-3}$	12	28	0.07
4_49	4	10	49	197	$0.196.10^{-3}$	35	205	0.01
nth_prime3_inc	3	5	14	42	$0.138.10^{-3}$	7	17	0.01
nth_prime4_inc	4	12	48	256	$0.139.10^{-3}$	27	171	0.01
nth_prime5_inc	5	30	160	1534	$0.408.10^{-3}$	72	832	0.02
ham3	3	3	9	19	$0.141.10^{-3}$	6	10	0.01
hwb4	4	8	41	173	$0.217.10^{-3}$	29	151	0.01
hwb5	5	24	149	1183	$0.387.10^{-3}$	79	805	0.02

TABLE 8
Benchmark synthesis in comparison with the TBS method.

different ways using the MCT gate library. First, we consider small functions to be implemented by our method and the TBS method. We generate all functions up to 5 bits that can be implemented with 2 or 3 essential functions. Then, we obtain average reversible costs using our method and the TBS method; we do not consider run times since they are negligibly small. Table 7 shows the results. Examining the numbers, we can state that our synthesis method performs better compared to the TBS. Our second way of comparison is based on the benchmarks from Maslov's website [9]. Results are given in Table 8. It can be seen that, our synthesis times are always at least hundred times smaller than those obtained using the TBS approach at the cost of the circuit size.

We also consider the Quantum Multiple-valued Decision Diagrams (**QMDD**) approach proposed by [14]. We compare our method with the QMDD method by using the MPMCT library; we use the same benchmarks

Name	Bit Size	**Proposed**				QMDD [14]		
		# of Essential Functions	Reversible Cost	Quantum Cost	Time (s)	Reversible Cost	Quantum Cost	Time (s)
4b15g_1	4	11	37	360	$0.096 .10^{-3}$	21	146	0.03
4b15g_2	4	12	44	400	$0.101 .10^{-3}$	19	161	0.07
4b15g_3	4	12	36	334	$0.191 .10^{-3}$	-	-	-
4b15g_4	4	11	29	294	$0.089 .10^{-3}$	20	135	0.01
4b15g_5	4	10	28	274	$0.090 .10^{-3}$	20	112	0.01
3_17	3	4	12	38	$0.068 .10^{-3}$	7	17	0.04
4_49	4	10	38	284	$0.196 .10^{-3}$	16	116	0.03
nth_prime3_inc	3	5	11	33	$0.063 .10^{-3}$	-	-	-
nth_prime4_inc	4	12	32	284	$0.115 .10^{-3}$	16	133	0.01
nth_prime5_inc	5	30	116	2318	$0.322 .10^{-3}$	-	-	-
ham3	3	3	7	19	$0.039 .10^{-3}$	7	19	0.01
hwb4	4	8	32	252	$0.091 .10^{-3}$	21	151	0.01
hwb5	5	24	112	1664	$0.230 .10^{-3}$	49	806	0.02

TABLE 9
Benchmark synthesis in comparison with the QMDD method. Dashes are put if no solution is found in 15-minute run.

Bit Size	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Number of Functions	10^6												10^3					
Average Time (s)	<0.0001								<0.001			0.002	0.003	0.007	0.010	0.031	0.064	0.140

TABLE 10
Average run times of the proposed algorithm to synthesize functions with different bit sizes.

as used in Table 8. Results are given in Table 9 that again approves the speed of our algorithm. Note that three functions (4b15g_ 3, nth_ prime3_ inc, and nth_ prime4_ inc) could not be synthesized by the QMDD method in a sufficient time duration, selected as 15 minutes.

In order to evaluate the scalability performance of our method for higher bits, we consider functions with bit sizes between 1-21. We randomly generate one million and one thousand functions for bit sizes between 1-15 and 16-21, respectively. Table 10 shows the results. Examining the numbers, we can see an almost linear relationship between the run times and the number of truth table rows. This is an expected result since our synthesis approach directly deals with truth table rows. Therefore, we expect that increasing bit sizes by 1, beyond 21 bits, results in doubled run times. Using this assumption, average run times for 29-bit and 33-bit functions are achieved below 1

minute and 15 minutes, respectively. Of course, this expectation can not be directly used for certain types of functions. For instance, if different functions with different bit sizes need same number of essential functions then corresponding synthesis run times are expected to be close.

6 CONCLUSION

In this paper, we present new synthesis and optimization methods for reversible circuits. We propose a fast synthesis algorithm that implements any given reversible Boolean function using the NCT, the MCT, and the MPMCT libraries. Instead of an exhaustive search on every given function, our algorithm creates a library of essential functions and performs sorting. We also propose reversible and quantum cost optimization techniques by considering adjacent gate pairs. The proposed algorithms are evaluated on benchmark circuits in comparison with the results in the literature.

ACKNOWLEDGMENTS

This work is supported by The Scientific and Technological Research Council of Turkey (TUBITAK) 1002 Project # 215E268 and the Scientific Research Projects Agency of Istanbul Technical University (ITU-BAP).

REFERENCES

[1] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. (1995). Elementary gates for quantum computation. *Physical Review A*, 52(5):3457.

[2] Charles H Bennett. (1973). Logical reversibility of computation. *IBM journal of Research and Development*, 17(6):525–532.

[3] Antoine Bérut, Artak Arakelyan, Artyom Petrosyan, Sergio Ciliberto, Raoul Dillenschneider, and Eric Lutz. (2012). Experimental verification of landauer/'s principle linking information and thermodynamics. *Nature*, 483(7388):187–189.

[4] Zhengbing Bian, Fabian Chudak, William G Macready, Lane Clark, and Frank Gaitan. (2013). Experimental determination of ramsey numbers. *Physical review letters*, 111(13):130505.

[5] Kamalika Datta, Gaurav Rathi, Robert Wille, Indranil Sengupta, Hafizur Rahaman, and Rolf Drechsler. (2013). Exploiting negative control lines in the optimization of reversible circuits. In *Reversible Computation*, pages 209–220. Springer.

[6] Kamalika Datta, Indranil Sengupta, and Hafizur Rahaman. (2015). A post-synthesis optimization technique for reversible circuits exploiting negative control lines. *Computers, IEEE Transactions on*, 64(4):1208–1214.

[7] Rolf Drechsler and Robert Wille. (2012). Reversible circuits: Recent accomplishments and future challenges for an emerging technology. In *Progress in VLSI Design and Test*, pages 383–392. Springer.

[8] Oleg Golubitsky and Dmitri Maslov. (2012). A study of optimal 4-bit reversible toffoli circuits and their synthesis. *Computers, IEEE Transactions on*, 61(9):1341–1353.

[9] Dmitri Maslov, G Dueck, and Nathan Scott. (2005). Reversible logic synthesis benchmarks page. *Online: http://www. cs. uvic. ca/˜ dmaslov*.

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

[11] Dmitri Maslov and D Michael Miller. (2007). Comparison of the cost metrics through investigation of the relation between optimal ncv and optimal nct three-qubit reversible circuits. *IET Computers & Digital Techniques*, 1(2):98–104.

[12] Dmitri Maslov, Christina Young, D Michael Miller, and Gerhard W Dueck. (2005). Quantum circuit simplification using templates. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 1208–1213. IEEE.

[13] D Michael Miller, Dmitri Maslov, and Gerhard W Dueck. (2003). A transformation based algorithm for reversible logic synthesis. In *Proceedings of the 40th annual Design Automation Conference*, pages 318–323. ACM.

[14] D Michael Miller and Mitchell A Thornton. (2006). Qmdd: A decision diagram structure for reversible and quantum circuits. In *Multiple-Valued Logic, 2006. ISMVL 2006. 36th International Symposium on*, pages 30–30. IEEE.

[15] Michael A Nielsen and Isaac L Chuang. (2010). *Quantum computation and quantum information*. Cambridge university press.

[16] Ron Orbach, Françoise Remacle, RD Levine, and Itamar Willner. (2012). Logic reversibility and thermodynamic irreversibility demonstrated by dnazyme-based toffoli and fredkin logic gates. *Proceedings of the National Academy of Sciences*, 109(52):21228–21233.

[17] Mehdi Saeedi and Igor L Markov. (2013). Synthesis and optimization of reversible circuits - a survey. *ACM Computing Surveys (CSUR)*, 45(2):21.

[18] Eleonora Schonborn, Kamalika Datta, Robert Wille, Indranil Sengupta, Hafizur Rahaman, and Rolf Drechsler. (2014). Optimizing dd-based synthesis of reversible circuits using negative control lines. In *Design and Diagnostics of Electronic Circuits & Systems, 17th International Symposium on*, pages 129–134. IEEE.

[19] Nathan O Scott and Gerhard W Dueck. (2008). Pairwise decomposition of toffoli gates in a quantum circuit. In *Proceedings of the 18th ACM Great Lakes symposium on VLSI*, pages 231–236. ACM.

[20] Vivek V Shende, Aditya K Prasad, Igor L Markov, and John P Hayes. (2003). Synthesis of reversible logic circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 22(6):710–722.

[21] Omercan Susam and Mustafa Altun. (2014). An efficient algorithm to synthesize quantum circuits and optimization. In *Electronics, Circuits and Systems (ICECS), 2014 21st IEEE International Conference on*, pages 570–573. IEEE.

[22] Robert Wille and Rolf Drechsler. (2009). Bdd-based synthesis of reversible logic for large functions. In *Proceedings of the 46th Annual Design Automation Conference*, pages 270–275. ACM.

[23] Robert Wille, Mehdi Saeedi, and Rolf Drechsler. (2010). Synthesis of reversible functions beyond gate count and quantum cost. *arXiv preprint arXiv:1004.4609*.

[24] Robert Wille, Mathias Soeken, D Michael Miller, and Rolf Drechsler. (2014). Trading off circuit lines and gate costs in the synthesis of reversible logic. *Integration, the VLSI Journal*, 47(2):284–294.

[25] Robert Wille, Mathias Soeken, Nils Przigoda, and Rolf Drechsler. (2012). Exact synthesis of toffoli gate circuits with negative control lines. In *Multiple-Valued Logic (ISMVL), 2012 42nd IEEE International Symposium on*, pages 69–74. IEEE.