IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 19, 2020, accepted August 22, 2020, date of publication August 25, 2020, date of current version September 4, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3019356

Reversible Logic Synthesis Using Binary Decision Diagrams With Exploiting Efficient Reordering Operators

BAKER K. ABDALHAQ, AHMED AWAD, (Member, IEEE), AND AMJAD HAWASH
Information and Computer Science Department, An-Najah National University, Nablus 44831, Palestine

Corresponding author: Ahmed Awad (ahmedawad@najah.edu)

This work was supported by the An-Najah National University under Grant ANN-1920-Sc006.

ABSTRACT With the continuous shrinkage of transistor sizes in very large scale integrated circuits, power consumption forms a serious concern to be tackled. With their ability to allow for zero energy dissipation, reversible circuits have been considered as promising solution to meetup with low power design requirements. Moreover, advances in their synthesis methodologies can be easily applied to quantum circuits due to the inherited reversibility of the latter. Although numerous algorithms have been proposed in the literature to synthesize reversible circuits and map them into their corresponding quantum circuits, the scalability and computational effort of such algorithms form a serious concern when synthesizing large size input functions. Binary Decision Diagram-based synthesis for reversible circuits has shown great evidence in realizing reversible circuits with low quantum cost through exploiting proper reduction rules for smaller graph size. However, the order of the variables in the decision diagram impacts its overall size, and thus, the cost of its corresponding reversible circuit. While several reordering algorithms have been proposed in this manner, their direct impact on the quantum cost has not been considered. In this article, a Binary Decision Diagram-based algorithm for reversible circuit synthesis is proposed to synthesize reversible circuits for a given Boolean function with low quantum cost through exploiting a linearized relationship between the decision diagram size and the corresponding quantum cost. Thereafter, different decision diagram reordering algorithms have been integrated with the proposed algorithm and compared in terms of their impact on the quantum cost. Experimental results show that Genetic Algorithm-based reordering for decision diagram, supported with, cycle crossover, inverse mutation, and tournament selection, results in the least quantum cost of the output circuit if compared with other algorithms due to its property in preserving the nodes of the decision diagram in their near-optimal locations during the optimiation recipe.

INDEX TERMS Reversible logic, binary decision diagram, quantum cost, crossover, mutation, selection.

I. INTRODUCTION

While advanced technology nodes continue scaling down in terms of transistor sizes, power dissipation represents a major obstacle in the development process of high performance Integrated Circuits (ICs). In accordance with Laundauer's principle, each bit loss results in energy dissipation in the form of heat. However, as reversible circuits preserve every bit value from being lost, zero energy dissipation is allowed. Therefore, reversible logic introduces a promising solution to overcome the challenges in the design of high performance computing devices [1], [2].

The associate editor coordinating the review of this manuscript and approving it for publication was Siddhartha Bhattacharyya.

In reversible logic, the number of input bits should equal the number of output bits with a bijective mapping between each input-output vector. Consequently, from an output vector, the input can be obtained and vice-versa. Such a mapping is required in wide range of applications including: cryptography, Digital Signal Processing (DSP), DNA applications, and others [3], [4]. Furthermore, the advances in reversible logic field can be well exploited in quantum computations since quantum circuits are inherently reversible. As a result, quantum computation eventually profits from the development of effective reversible circuit synthesis algorithms [5].

On the contrary of irreversible logic, fan-out and feedback paths are not allowed in reversible logic. To impose such constraints, the synthesis process of reversible circuits

significantly differs from the synthesis process of their corresponding irreversible circuits. With being cascaded together, reversible gates form a reversible circuit that realizes a required Boolean function. If the function to be realized is irreversible, it is embedded in a reversible circuit with adding some additional input bits (known as constant inputs) and additional output bits (known as garbage outputs) to preserve the required bijective mapping [6]. Fig. 1 illustrates a reversible circuit realizing the AND Boolean function using Toffoli gate. Notice that a constant input and two garbage outputs (P and Q) have been added to make the circuit reversible.

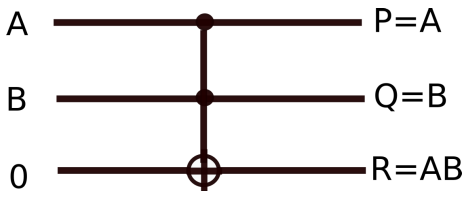


FIGURE 1. Realizing AND Boolean function using a reversible Toffoli gate with one constant input (with value of 0) and two garbage outputs (P and Q).

The cost of a reversible circuit is characterized by several metrics including: the number of reversible gates cascaded together, the number of control lines, and the number of elementary quantum gates required to transform that circuit into a quantum circuit, known as the Quantum Cost (QC). While the choice of the appropriate cost metric is mainly driven by the type of the quantum circuit, the QC has been considered as a key metric to be tackled by optimization during the synthesis process [7].

A lot of effort is evident in the literature in developing effective algorithms for reversible circuit synthesis. From the specifications of the input Boolean function to be realized, exact and heuristic based synthesis algorithms have been employed to realize the input function with low cost such as: Transformation Based Synthesis (TBS), And-Inverter Graphs (AIGs), and Lookup Table (LUT) based synthesis [8]–[10]. However, the limited scalability of such algorithms is a major barrier when synthesizing an input function with a large number of variables. On the other hand, some algorithms have been proposed to reduce the number of constant inputs and garbage outputs through detecting and eliminating redundant lines during the synthesis process [11], [12]. However, such algorithms might increase the quantum cost as well as the number of reversible gates needed to realize the required functions. Although Exact algorithms outperform others in terms of the cost of the synthesized circuits, they slowly converge for large-scale input functions. Furthermore, their memory demands rise abruptly with the size of the input function [11], [13].

Binary Decision Diagram (BDDs) have been well exploited in the field of reversible circuits synthesis, due to their efficient data structure representation of large size Boolean functions which, in turn, overcomes the large memory demands problem induced by using traditional truth table representations. BDDs have been widely used in combinational applications as well [14], [15]. In BDD-based synthesis for reversible circuits, a one-to-one mapping is applied between each BDD node and a cascade of Toffoli gates. This mapping is guided by a predefined lookup table constructed in a pre-processing stage, as proposed in [16]. This utilization of BDD in reversible circuit synthesis is beneficial in twofold: (1) The size of the resultant circuit is bounded by the size of its corresponding BDD. (2) BDD reduction rules and BDD reordering algorithms can be effectively exploited in this area, so that smaller BDD is obtained, which is more likely, to result in less cost reversible circuit. Thus, reordering BDD variables with preserving its correctness in representing a Boolean function is critical stage during the synthesis process, as proper reordering accompanied with reduction rules is expected to reduce the size of the BDD, and thus, the cost of its corresponding reversible circuit [17]. Recently, one-pass design for reversible circuits has been introduced with exploiting BDDs as well [18].

Many algorithms have been proposed in the literature to find the order of the BDD nodes that results in low BDD size once reduction rules are applied [19]. However, for such an NP-Complete problem [20], meta-heuristic based algorithms (including both swarm and evolutionary based algorithms) have demonstrated great evidence in finding near-optimal solutions if compared with other deterministic algorithms [21]–[23]. Other proposed algorithms, such as sifting [24], and dynamic programming-based [25], are faster to converge, at the cost of resulting larger BDD sizes. However, the direct impact of the various BDD reordering algorithms on the quantum cost of the synthesized reversible circuits has not been considered in the literature. In addition, there is a need to integrate efficient evolutionary operators in this optimization process to achieve less quantum cost in the obtained reversible circuits.

In this article, a BDD-based reversible circuit synthesis algorithm is proposed whose objective is to synthesize a reversible circuit for a given Boolean function with low quantum cost and within a reasonable synthesis time. This algorithm is driven by a linearized relationship between the BDD size and the QC of its resultant reversible circuit. BDD reordering forms a crucial part of the proposed synthesis algorithm. Therefore, a comprehensive study for different BDD reordering algorithms in terms of their direct impact on the QC of the final synthesized reversible circuits is proposed. The compared reordering algorithms are: sifting, exact, window permutation, Simulated Annealing (SA) based, and Genetic Algorithm (GA) based reordering algorithms. Thereafter, as GA based BDD reordering algorithm outperforms others in terms of the quantum cost of the synthesized circuits, supplementary crossover and mutation operators have been integrated with this algorithm to achieve additional reduction on the BDD size, and thus, the quantum cost of the output reversible circuits. Three kinds of crossovers are compared in this context. These crossovers are: Partially Mapped Crossover (PMX), Ordered Crossover (OX), and

Cycle Crossover (CX). Then, the algorithm is supported with inverse mutation for further reduction of the cost. Finally, the impact of the selection operator in the proposed GA-based BDD reordering algorithm is verified through comparing the tournament selection with roulette wheel selection operators in terms of the QC of the output synthesized circuits.

The contributions of this work can be summarized as follows:

- A fast BDD-based reversible circuit synthesis algorithm is proposed through exploiting a linearized relationship between the BDD size and the QC of its corresponding reversible circuit.
- A comprehensive comparative study between different BDD reordering algorithms when integrated with the proposed synthesis algorithm, is conduced. This comparison includes: gate count, line number, and most importantly, the quantum cost of the synthesized reversible circuit.
- A GA based BDD reordering algorithm with supplementary crossover operators is proposed and evaluated in terms of the quantum cost and runtime.
- Supplementary simple inverse mutation is integrated with the algorithm for further enhancement.
- Tournament and roulette wheel selection operators have been integrated with the proposed algorithm and their impact on the quantum cost has been validated.

It is important to mention that in addition to the application of reversible circuit's synthesis with low quantum cost, the proposed BDD reordering algorithms whose objective is to reduce the BDD size can be applied for other applications, wherein, the BDD size is directly related to the time and space complexity of the digital system represented by the BDD outputted from the algorithm mentioned in this work. For example, in fault simulation, the fault combinations are represented using BDDs, and thus, reducing their sizes is expected to reduce the space and time complexity of this simulation. Similarly, the verification of sequential circuits exploits BDD representations for state sets. Moreover, one-pass design for reversible circuits has been introduced using BDDs. In all those applications, exploiting the proposed BDD reordering algorithm is expected to reduce the overall space and time complexities, and thus, improve the overall performance of the system [18], [26].

The rest of the paper is organized as follows: In Section II, previous work is presented. Section III describes preliminaries, reversible logic terminology, and cost metrics. The general reversible circuit synthesis algorithm is presented in Section IV with brief description for each BDD reordering algorithm with the supplementary crossover, mutation, and selection operators used in this article. Section V shows experimental results and the work conclusion is presented in Section VI.

II. PREVIOUS WORK

Optimizing reversible circuits synthesis was and still a major topic for quantum computations related researchers. Several works and algorithms related to this topic have been proposed and conducted in literature.

Transformation Based Synthesis (TBS) is one of the most recognized algorithms for reversible circuit synthesis. In this algorithm, each line of a given truth table is traversed and reversible gates are added subsequently such that the output values match the input values, without altering already considered lines. However, this algorithm cannot find solutions for large input functions. Added to that,the increased number of input variables leads to abrupt increase in memory demands [10].

Exact combination and synthesis of reversible circuits has been introduced too in [13] with the cost of slow execution of the algorithm to find one of the correct solutions. Alternative algorithms have been proposed as well. One of these is based on Boolean Satisfiability Solver (SAT) discussed in [19]. The main aim of this algorithm is to reduce the gate count of a given Toffoli gates network by examining the satisfiability of all feasible solutions. However, with large scale and complex circuits, such algorithms slowly converge. Near-optimal synthesis of reversible circuits have been tested and exploited by heuristic algorithms that are designed to find the best available solution for such problems. Genetic algorithm and particle swarm optimization are examples of such algorithms that have been discussed in [27], [28] and used to synthesis reversible functions. However, studies found that these algorithms might slowly converge on small-scale circuits and an integration with local search algorithms is needed to produce low cost reversible circuits.

Authors of the work conducted in [29] and [30] presented a method that synthesizes a network with Toffoli and Fredkin gates in two steps. First, their synthesis algorithm finds a cascade of Toffoli and Fredkin gates with no backtracking and minimal look-ahead. Next, the authors apply transformations that reduce the number of gates in the network. Transformations are accomplished via template matching with basis represented with m gates that realizes the identity function. The transformation that reduces the gate count can be applied depending on the result of template comparison with the sequence of gates to be reduced. However, this algorithm has been applied to small input functions since the entire truth table has to be stored, which abruptly increases space complexity.

The work of [31] presents a synthesis algorithm, Covering Set Partitions (CSP) without generating garbage bits. The algorithm utilizes the inherent mathematical specifications of binary numbers to construct input sequences which are ensured to lead to valid outputs. The proposed algorithm is able to generate functions of huge number of variables (up to 30 bits) in a acceptable amount of time. The authors tested the algorithm on a randomly selected subsets of all correct input sequences of binary numbers.

The work presented in [32] covers the application of Genetic algorithm to synthesize large reversible circuits by CSP. The work is concerned in discovering the set of optimal solutions taking into account minimizing circuit costs within

a huge space of solutions. The work investigates three searching algorithms: Genetic, Random and Tabu searches. The criterion used to compare the three algorithms depending on the quantum cost that is represented by the generated number of logical gates by each algorithm.

One of the major works related to synthesis process optimization is discussed in [5]. The mentioned work concentrates on generating correct permutations for a binary function symbolizing the reversible circuit to minimize the solutions search space. NCT (NOT, CNOT, Toffoli) gate based library has been counted for synthesis of reversible logic circuits. The proposed algorithm generates a cascade of Toffoli gates for some reversible description of a circuit. However, the impact on the quantum cost of the synthesized circuit has not been considered.

Investigating the influence of different graph-based intermediate representations including: And-Inverter Graph (AIG), Binary Decision Diagram (BDD), and Majority Inverter Graph (MIG), in terms of their contribution in producing efficient circuit realizations, is the work published in [33]. By this work, it has been found that it is preferable to use BDD for small functions to realize reversible circuits since it produces more compact circuits than the other two mentioned representations. However, with integrating BDD reduction rules, it is possible for BDD-based synthesis algorithms to scale well for complex input functions.

Introducing BDD to the process of synthesizing reversible circuits added a noticeable improvement that can be touched in reducing synthesized circuit size through exploting BDD reduction rules and proper variable reordering [1], [34]. According to this, the algorithm discussed in [17] introduces an enhanced version of transformation based synthesis, wherein, BDD structure is used in the reversible circuit synthesis instead of the truth table. In the work presented in [16], the authors propose a synthesis approach that deals with huge circuits whose corresponding functions contain more than a hundred of different variables. The authors present in this work a methodology to extract reversible circuits for functions based on their corresponding BDDs. The circuit is obtained using an algorithm with linear worst case behavior regarding run-time and space requirements. Furthermore, the size of the resulting circuit is bounded by the BDD size. This allows to transfer theoretical results known from BDDs to reversible circuits. Experiments show better results (with respect to the circuit cost) and a significantly better scalability in comparison to previous synthesis approaches. Another work that exploits BDD in reversible logic synthesis through decomposing the BDD version of a Boolean function into reversible logic components has been recently published in In [35].

The size of BDD is influenced by the order of the variables in its related binary function, and thus, this affects the cost of the corresponding constructed reversible circuit. Several greedy algorithms have been proposed to change the order of the binary function variables such as Sifting and Windowing algorithms. The change of variables order and rebuilding of the corresponding BDD leads to change the positions of nodes inside the BDD itself and thus, changes the size of BDD when further reduction rules are applied [24]. Another BDD reordering algorithm exploits dynamic programming through dividing the BDD into levels, and then reordering the nodes for each of those levels [25]. This algorithm is generally slower than sifting algorithm. Heuristic-based reordering algorithms have shown great evidence in reordering BDD variables with small BDD sizes. For example, Genetic Algorithm (GA) and Simulated Annealing (SA) algorithms have been well exploited in BDD reordering problem [21], [23]. Recent swarm optimizaiton algorithms have been also exploited to solve BDD reordering problem as published in [22]. However, the direct impact of BDD reordering algorithms on the cost of the synthesized reversible circuits should be investigated.

In this work, a comparative study has been conducted for BDD reordering algorithms including: sifting, window, exact, genetic-based, and simulated annealing based reordering, in terms of the quantum cost of generated reversible circuit. The conducted experimental results of this work show that meta heuristic-based BDD reordering algorithms outperform other algorithms in terms of the overall synthesized circuit cost with slightly additional run time if compared with other greedy algorithms. Thereafter, further enhancement is achieved through integrating proper selection, mutation, and crossover operators with the genetic based reordering for BDD. This includes comparing several kinds of crossover operators including: Partially Mapped Crossover (PMX), Ordered Crossover (OX) and Cycle Crossover (CX). When integrated with inverse mutation, CX achieves additional quantum cost reduction. This leads to further analysis for the impact of selection operator through conducting a comparison between tournament and roulette wheel selection. Integrating all those operators have reduced the cost of the output circuits if compared with other BDD reordering algorithms.

III. PRELIMINARIES

The following subsections provide a compact specification for the basic terminology related to reversible circuit synthesis as well as to the formulation for its related cost metrics.

A. REVERSIBLE LOGIC RELATED TERMINOLOGIES

A reversible function $f : B^n \rightarrow B^n$ such that $B \in \{0, 1\}$ can be realized by $n \times n$ reversible gate, where a bijective mapping is applied on f for every n-bit input into n-bit output vectors. A reversible circuit $G = (g_0, g_1, \cdots, g_{m-1})$ is considered a cascade of reversible gates that implement a reversible function.

The development of reversible gates and circuits considered after Toffoli proposed reversible logic gates in the work presented in [36]. In a reversible logic gate, a unique input always is associated with a unique output and vice versa (bijection). NOT, Feynman, Fredkin, Peres, and Clifford gates are all examples of reversible gates that have been proposed over the latest decades [6], [37].

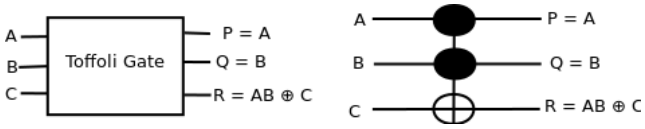


FIGURE 2. Representations of Toffoli gate where the control lines are represented by *A* and *B* while the target line is represented by *C*. The symbol ⊕ means a bit wise XOR operation.

A sample Toffoli gate is composed of both a target line and a some of control lines (as depicted in Fig. 2). The target line will be signaled to be inverted if all control lines are signaled to 1. Otherwise, it remains unchanged [1]. Table 1 shows the truth table that illustrates the implementation of the Toffoli gate shown in Fig. 2, where values of A, B and C represent the input lines and P, Q, and R represent the output lines.

TABLE 1. Truth table for Toffoli gate shown in Fig. 2.

Input			Output		
A	B	C	P	Q	R
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

Toffoli gates are widely used reversible gates in the literature due to their simplicity in structure [38]. Most of related works depend on Toffoli gate in their tries of optimizing the reversible circuits synthesis. Therefore, in this work, Toffoli gates are considered in both the proposal and the experiments.

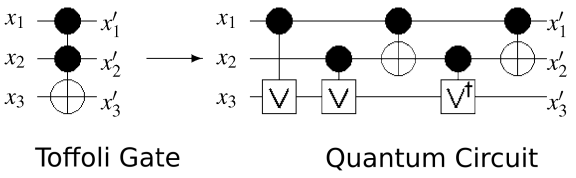


FIGURE 3. A a 3 × 3 Toffloli gate quantum circuit.

B. COST METRICS OF REVERSIBLE LOGIC

The reversible circuit Quantum Cost (QC) is defined as the number of basic quantum operations required to synthesize the corresponding reversible gate. A Toffoli gate g with C positive control lines, its QC is formulated as in eq.(1) [38]. Fig. 3 depicts the quantum circuit realization for a 3 × 3 Toffloli gate with QC equals to 5.

$$QC(g) = 2^{C+1} - 3 \tag{1}$$

The Quantum Cost (QC) for a reversible circuit is computed by summing up the QC for each reversible gate composing it, as formulated in eq.(2) [1]. For example, the QC for the reversible circuit shown in Fig. 4 equals to $5 * (2^{1+1} - 3) + 2^{2+1} - 3 = 10$

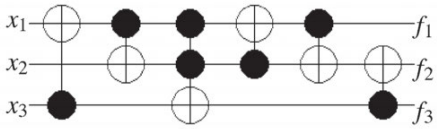


FIGURE 4. A reversible circuit that consists of 5 Toffoli gate with a total QC of 10.

$$QC(G) = \sum_{\forall g_i \in G} QC(g) \tag{2}$$

The cost for the synthesized reversible circuit might be driven by the type of the generated quantum circuit. For example, Near-term devices (NISQ quantum circuits) are evaluated in terms of the number of qubits. On the other hand, fault-tolerant quantum computers are evaluated in terms of the number and depth of T-gates. However, for generalization, the cost of a reversible circuit is characterized by its corresponding QC in this article, due to its impact on the effort needed to convert it into a quantum circuit.

C. BINARY DECISION DIAGRAM TERMINOLOGY

A Binary Decision Diagram (BDD) is an easily manipulated compact data structure to represent a Boolean function with less memory requirements than that of a truth table. Formally, it is a directed acyclic graph consisting of terminal and non-terminal nodes. A terminal node has a value of 0 or 1 while a non terminal node is augmented by one of the input variables of the Boolean function. A BDD is constructed through recursively applying Shannon decomposition as given in eq.(3), where x_i represents an input variable, f represents the Boolean function to be represented, f_{x_i} represents the value of the Boolean function f when $x_i = 1$ (known as positive cofactor), and $f_{\bar{x}_i}$ represents the value of the function f when the variable $x_i = 0$ (known as negative cofactor) [39].

$$f = \bar{x}_i f_{\bar{x}_i} + x_i f_{x_i} \tag{3}$$

As an example, consider the function $f = x_1.x_2 + x_3$, Fig. 5 illustrates the BDD for this function when Shanon decomposition is recursively applied. Notice that each complete path in the BDD represents a row in the truth table of the function. For example, if the input $x_1x_2x_3 = 101$, then following this path in the BDD leads to terminal node 0, which is the output of this Boolean function.

The size of a BDD is defined as the number of its non-terminal nodes. As an example, the BDD shown in Fig. 5 has a size of 3. However, BDD reduction rules have been proposed to reduce the size of the BDD while preserving its correctness in representing a given Boolean function. Shared nodes and complemented edges are examples of such reduction rules [40].

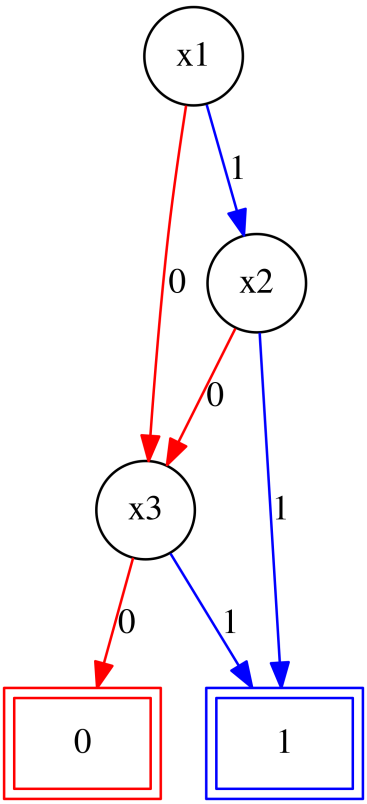


FIGURE 5. BDD representation of the Boolean function $f = x_1x_2 + x_3$.

The order in which the input variables of the Boolean function appears in the BDD impacts the BDD size when further reduction rules are applied. As an example, when the BDD of the Boolean function $f = x_1 \cdot x_2 + x_3 \cdot x_4 + \cdots + x_{n-1} \cdot x_n$ is represented in the order shown in Fig. 6-a, its size is given in a complexity of $O(n)$. Whereas, if the order of variables has been changed to be as in Fig. 6-b, the size of the BDD will be in a complexity of $O(2^n)$ [41].

D. PROBLEM FORMULATION

Given an input Boolean function f to be realized with a reversible circuit, the objective is to find a reversible circuit G that implements the function f with low Quantum Cost (QC) within a synthesis time that does not exceed a predefined upper bound. This problem is formulated in eq.(4) where the input Boolean function to be realized is denoted by $f(X)$ such that $X = x_1, x_2, \cdots, x_n$ and $QC(G)$ represents the QC of the circuit solution G whose synthesis time τ should not exceed a predefined upper bound τ_U.

$$\begin{aligned} &\underset{G}{\text{minimize}}\ QC(G) \\ &\text{subject to } G \text{ realizes } f(X) \\ &\tau \leq \tau_U \end{aligned} \tag{4}$$

To synthesize a reversible circuit using BDD-based approach, each node in the BDD is substituted by a cascade of Toffoli gates following a pre-defined lookup table [16].

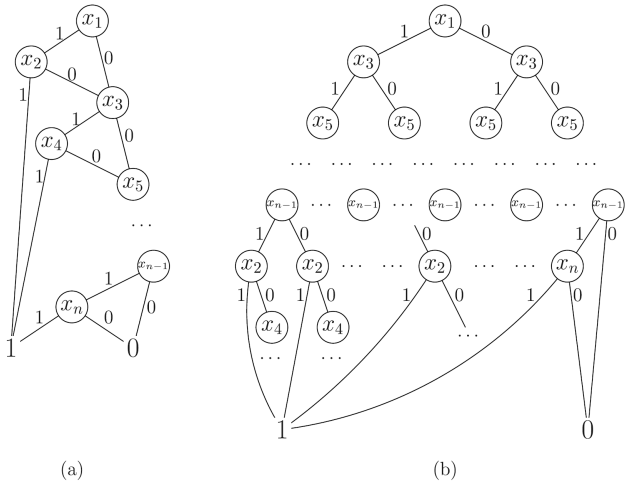


FIGURE 6. Two BDD representations of the function $f = x_1 \cdot x_2 + x_3 \cdot x_4 + \cdots + x_{n-1} \cdot x_n$ (a) Order 1 with a complexity of n. (b) Order 2 with a complexity of $O(2^n)$ [41].

With such a one-to-one mapping, the QC of the synthesized reversible circuit is expected to increase with the size of its companion BDD.

To investigate the impact of the BDD size on the QC of the synthesized reversible circuit, QC has been recorded versus the BDD size for several benchmarks. Fig. 7 illustrates the plot which shows that the QC can be linearly correlated with the BDD size. The correlation coefficient has been found to be around 0.985. Hence, BDD size forms a reasonable evaluation criterion for this optimization problem.

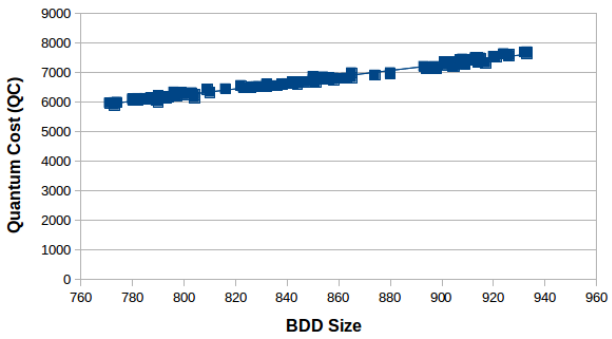


FIGURE 7. BDD size versus QC. The plot shows a roughly linear relation with a correlation coefficient of 0.955. All results were obtained using Revkit simulator [3].

Therefore, the optimization problem given in eq.(4) can be roughly formulated as follows: Given an input Boolean function $f(X)$, the objective is to find the best order $\pi(X)$ of the variables in the BDD representation that results in the least BDD size upon applying reduction rules. This formulation is given in eq.(5) where π represents the order of input variables in the BDD representation and $\psi(\pi)$ represents the size of the resultant BDD of that order when reduction rules are applied. The BDD size should not exceed maximum

value of $\alpha|X|$, where α is a predefined constant. This upper bound is important to avoid generating huge BDDs for the algorithms driven by random permutations. For example, the BDD shown in Fig. 6-a is a solution represented as $\pi(X) = (x_1, x_2, x_3, \ldots, x_n)$ and $\psi(\pi) = n$. It is important to mention that with this solution representation (variable order/permutation) the solution space size for this optimization problem equals to $n!$.

$$\begin{aligned} &\underset{\pi(X)}{\text{minimize}}\ \psi(\pi(X)) \\ &\textit{subject to}\ \psi(\pi(X)) \leq \alpha * |X| \\ &\tau \leq \tau_U \end{aligned} \tag{5}$$

IV. FLOW OF REVERSIBLE CIRCUIT SYNTHESIS & REORDERING ALGORITHMS OF BINARY DECISION DIAGRAM

Once a function is represented by a BDD, each BDD node is mapped to a cascape of Toffoli gates following its location in the BDD and the location of its neighbors. This mapping is conducted following a pre-defined lookup table. Fig. 8 illustrates some cases of such mapping [16].

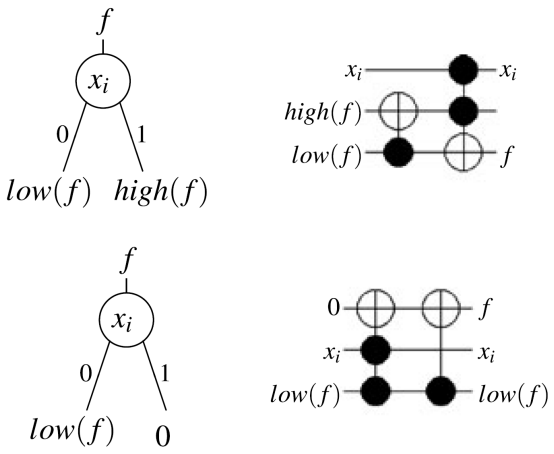


FIGURE 8. Mapping BDD nodes into a cascade of Toffoli gates [16].

A. ALGORITHM OF REVERSIBLE LOGIC SYNTHESIS

Fig. 9 illustrates the general flowchart of the proposed synthesis algorithm for reversible circuits. It consists of the following phases:

1) Initialization phase: In this phase, the initial BDD of the input Boolean function to be realized, is constructed.
2) Optimization phase: This phase is the core of this work. It applies the following steps iteratively until the stop conditions of the algorithm are satisfied:
   - Apply a BDD reordering algorithm to effectively reduce the number of nodes in the BDD. This part will be explained in details subsequently.
   - Apply reduction rules on the ordered BDD. Those rules include shared nodes and complemented edges reduction.

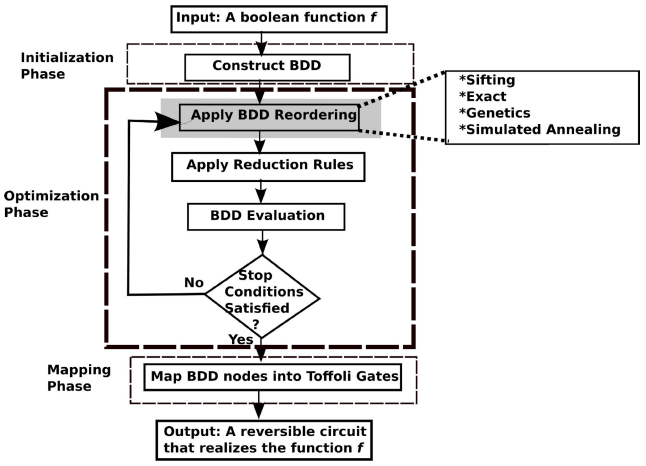


FIGURE 9. BDD-based reversible circuit synthesis algorithm.

   - Output a BDD with a satisfactory size to the mapping phase.
3) Mapping phase: This phase applies the algorithm published in [16] on the BDD resultant from the optimization phase. Each BDD node is substituted by a cascade of Toffoli gates following the mapping table published in [16].

B. BINARY DECISION DIAGRAM REORDERING ALGORITHMS

The following provides a breif description for the most widely used BDD reordering algorithms:

1) SIFTING ALGORITHM

This greedy algorithm is based on finding the optimum position for a variable, assuming all other variables to be fixed. The variable is exchanged with its successor until it becomes the next to the last variable in the Directed Acyclic Graph (DAG). Then it is moved up to the top of the DAG. The position for the best DAG size is remembered and the variable is moved down to the optimum position. If there are n variables in the DAG, then there are n potential positions for a variable, including its current position. Among those n positions, the sub-goal employed by the shifting algorithm is to find the spot which minimizes the size of the DAG. The shift algorithm has the advantage that a variable can move a long distance in the ordering. However, the DAG size can increase significantly after the first few variables swap, and then eventually reduce below the starting point. The swaps are based on the best position seen regardless of any increase in the intermediate DAG size [24].

2) WINDOW PERMUTATION ALGORITHM

The idea behind this algorithm is to choose a level I in the DAG and exhaustively searching all $k!$ permutations for the k adjacent variable starting at level I. This costs $k! - 1$ pairwise exchanges followed by up to $k(k-1)/2$ pairwise exchanges to restore the best permutation discovered. This mechanism is repeated for all levels in the DAG until no improvement

$x_1, x_2, x_3, x_4, x_5, x_6, x_7$	initial
$x_1, x_3, x_2, x_4, x_5, x_6, x_7$	swap(x_2, x_3)
$x_1, x_3, x_4, x_2, x_5, x_6, x_7$	swap(x_2, x_4)
$x_1, x_4, x_3, x_2, x_5, x_6, x_7$	swap(x_3, x_4)
$x_1, x_4, x_2, x_3, x_5, x_6, x_7$	swap(x_3, x_2)
$x_1, x_2, x_4, x_3, x_5, x_6, x_7$	swap(x_4, x_2)
$x_1, x_2, x_3, x_4, x_5, x_6, x_7$	swap(x_4, x_3)
$x_1, x_3, x_2, x_4, x_5, x_6, x_7$	swap(x_2, x_3)
$x_1, x_3, x_4, x_2, x_5, x_6, x_7$	swap(x_2, x_4)

FIGURE 10. Window permutation algorithm with a window size of 3.

in the DAG size is acquired. Fig. 10 illustrates the variable permutations that are explored when the size of the window equals to 3 starting at variable x_2. Size permutations are explored with 5 adjacent variable swaps, and then 3 additional variable swaps are used to restore the best permutation (this is the worst case).

During the algorithm execution, the current optimal permutation level is marked. This mark is reset when a new more optimal permutation is found for any of the preceding $k-1$ levels. When no more level are marked, the window permutation algorithm cannot improve the DAG size.

3) EXACT ALGORITHM BASED BINARY DECISION DIAGRAMS REORDERING

This dynamic programming algorithm takes into account that the cost of the variables on the initial levels of the BDD relies only on their ordering, without any attention to the order of the remaining variables. Because of this, the minimal cost ordering (in terms of the BDD size) is computed for the variables in the first k levels. Later on, a truth table with folding is repeatedly applied to the remaining set of variables to determine the best order of the variables that results a small size BDD. The disadvantage of this approach is that it is computationally severe from runtime point of view [25].

4) GENETIC ALGORITHM BASED BINARY DECISION DIAGRAMS REORDERING

This algorithm can be categorised with the algorithms that deals with the local search ones. Each solution is symbolized by a sequence of variables with some order. The algorithm initiates by producing random alternations (permutations) of the included variables. Sifting algorithm -as a local minimization mechanism- is applied on each solution in which a solutions generation is kept. The number of produced solutions in each iteration is relative to the number variables used as input for a given function. The algorithm acts randomly in the process of selection of two solutions such that the probability of selecting a solution is relative to its related BDD size. If a solution with relatively small size in an iteration, there is a probability to be chosen several times. After selecting two solutions, new ones are produced by mixing the parent ones [23]. Enhancing this algorithm by combining suitable selections, mutation process, and applying crossover operators suitable for BDD reordering issue, will be explained thereafter.

5) SIMULATED ANNEALING BASED BINARY DECISION DIAGRAMS REORDERING

Simulated annealing with an enhanced neighborhood determination is used to find the optimal ordered BDD. The variable neighborhood ordering is realized as the set of all variable orderings that can be generated with a single replacement or jump operation. The cooling schedule (annealing) is defined as: The initial transitions are always admitted which resulting in a random start alternation (permutation). Simultaneously, the size of the recognized ordered BDDs is used to compute an initial temperature. During the annealing process, a specific number of admitted switching are made at each temperature. The temperature is then decreased using a dynamic mechanism. The process ends when no more improvements can be applied [21].

C. IMPROVED GENETIC ALGORITHM BASED BINARY DECISION DIAGRAMS REORDERING

The solution space consists of all possible orders of variables in the BDD. The objective of GA optimization phase is to find the order of the variables that results in the least BDD size. It is important to mention that on the contrary of Traveling Salesman Problem (TSP), a solution $\pi_1 = (x_1, x_2, x_3)$ has a different cost of the solution rotated $\pi_2 = (x_3, x_2, x_1)$ because each solution results in a BDD whose size depends on the proceeding reduction rules. Every permutation is a different solution in BDD. Therefore, genetic crossover and mutation operators that preserve the optimal locations of some variables are expected to result in outputting BDDs with smaller sizes.

Algorithm 1 illustrates the GA for BDD reordering problem. First of all, the initial population is prepared as a random set of permutations for the variables of the BDD. For each permutation, the BDD is constructed and sifting algorithm [24] is applied on it as a local search to accelerate the convergence of the algorithm. As long as a predefined maximum number of iterations has not been reached, two parents are selected using selection operator. Those two parents are combined through applying a major crossover operator. Thereafter, either a another type of crossover or mutation operator is applied. The decision whether to apply a secondary crossover or mutation is made randomly, following a probabilistic mixing factor, denoted by γ. BDD for each offspring is constructed followed by applying reduction rules. Each solution is then evaluated in terms of the BDD size. If a solution results in a BDD size that is less than the largest BDD size that has been found so far in the population, it will replace the permutation resulting

Algorithm 1 GA-Based BDD Reordering Algorithm

```
procedure BDD-Reordering
    initPop ← generateRandomPopulation()
    I_max ← c * #var
    iter ← 0
∀p ∈ initPop
    bdd ← constuctBDD(p)
    bdd ← applySift(bdd)
while iter < I_max:
    p_max ← chooseBddWithLargestSize(initPop)
    p_i, p_j ← applySelection(initPop)
    r ← chooseRandomBetween(0.0, 1.0)
    if r ≤ γ then
        o_i, o_j ← applyCrossOver(p_i, p_j)
    else
        o_i, o_j ← applyOtherCrossOver(p_i, p_j)
    end if
    bdd_i, bdd_j ← constuctBDD(o_i, o_j)
    bdd_i, bdd_j ← applyReductionRules(bdd_i, bdd_j)
    bdd_i, bdd_j ← applySift(bdd_i, bdd_j)
    o_min ← chooseOffSpringWithSmallerBDDSize
(bdd_i, bdd_j)
    if cost(o_min) < cost(p_max) then
        p_max ← o_min.
        initPop ← updatePop(initPop, p_max).
    end if
end procedure
```

in the largest BDD size in the population. Once the algorithm stops, a BDD with a small size is outputted.

The GA optimization might require several iterations to find an acceptable solution for input Boolean function with large number of variables. On the other hand, fewer iterations might be required for input functions with limited number of variables. Therefore, the stop condition for the algorithm states that the maximum number of iterations is a multiple of the number of input variables.

1) CROSSOVER OPERATORS

Crossover operator might result in a newborn chromosome (permutation of BDD variables) which is better than the parents during the evolution process. In this article, three crossover operators have been chosen to be applied in BDD reordering. This includes: Partially Mapped Crossover (PMX), Ordered Crossover (OX), and Cycle Crossover (CX). It is important to mention that PMX has been in the literature for BDD-reordering problem. However, CX and OX have been chosen because they are expected to be less destructive for the solutions, and preserve the order of the variables that has been found to be near-optimal in the evolution process [42].

a: ORDERED CROSSOVER (OX)

Given two parent permutations for the order of variables in the BDD, two random crossover cutting points are selected in the two parents. Those points will partition each parent permutation into left, middle, and right portions. Now each child is constructed through preserving the middle portion of its corresponding parent, while the left and right portions are placed in the child in the order in which they appear in the other parent [42]. Fig. 11 illustrates an example of OX.

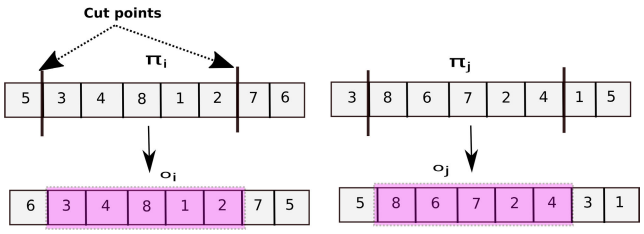


FIGURE 11. Ordered crossover where π_i, π_j represent two parent permutations and o_i, o_j represent their corresponding offsprings. Notice that the swash of consecutive variables between the cut points remains the same for each offspring as its corresponding parent.

b: CYCLE CROSSOVER (CX)

This crossover identifies a cycle between the two parent permutations. The formed cycle starting from the first parent is used then to fill the portions of the first offspring while the remaining parts are filled from the second parent following their order in that permutation. Similarly, the cycle formed starting from the second parent will be copied to the second offspring while the first parent will be used to fill the missing parts [43]. Fig. 12 illustrates an example of CX. Notice that the cycle formed starting with the parent permutation π_i is as follows: $1 \rightarrow 5 \rightarrow 2 \rightarrow 4 \rightarrow 7 \rightarrow 1$. All the parts of this cycle will be copied from permutation π_i to offspring o_i in their original positions in π_i.

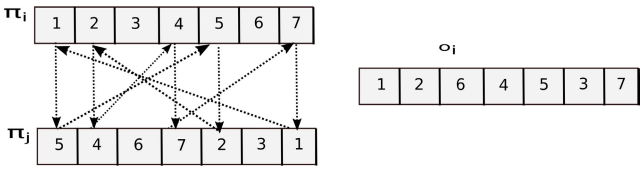


FIGURE 12. Cycle crossover where π_i, π_j represent two parent permutations and o_i represents the offspring resultant from a cycle starting with the first part of parent π_i. Notice that the dashed arrows represent how a cycle is constructed.

c: PARTIALLY-MAPPED CROSSOVER (PMX)

This crossover type defines two cut points in the two parents. The portion of variable positions between the two cut points will be exchanged between both parents to simultaneously generate portions of the two offsprings. The remaining parts of the offsprings will be determined using the mapping relation between both parents [43]. Fig. 13 illustrates PMX example with the following mapping relations resultant from the exchange process: $6 \leftrightarrow 3, 7 \leftrightarrow 4, 2 \leftrightarrow 5, 1 \leftrightarrow 6$

2) MUTATION OPERATOR

The purpose of mutation is to increase the exploration of the GA. This operator is applied to a single permutation to

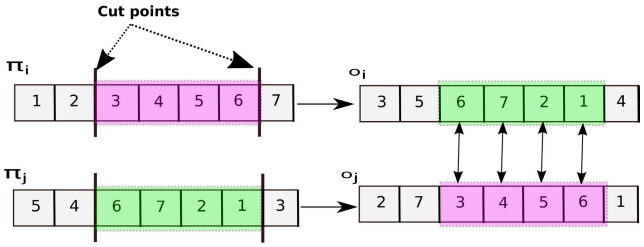


FIGURE 13. Partially mapped crossover where π_i, π_j represent two parent permutations and o_i, o_j represents the offsprings.

get a new permutation whose BDD can have a smaller size. In this article, inversion mutation is exploited, due to its simplicity during the optimization process. Inversion mutation chooses two random cut points. The portions of the permutation between those cut points are simply reversed. Fig. 14 illustrates inversion mutation.

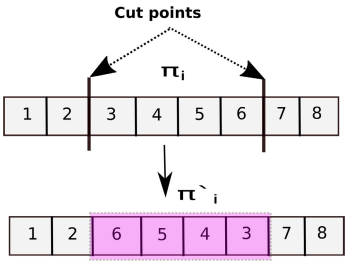


FIGURE 14. Inversion mutation where π_i represents a permutation and π_i' represents the new permutation after inversion.

3) SELECTION OPERATORS

Selection operator aims to choose the permutations in the population that will survive for the next iteration in the evolutionary process. To guarantee preserving BDDs with small sizes for the next iterations alongside with preserving population diversity for the upcoming crossover/mutation operators, probabilistic selection operators are exploited in this article. This includes: roulette wheel and enhanced tournament selections.

a: ROULETTE WHEEL SELECTION

This selection is fitness based. That is, the probability to select a permutation for the next generation is proportional to the size of its corresponding BDD. However, a BDD with very small size compared to the others might dominate the next generation, which limits the diversity of the population, and thus, lacks the ability of exploring permutations that result in smaller BDD sizes.

b: ENHANCED TOURNAMENT SELECTION

This selection is ordered based. N-way tournament selection groups the population randomly into clusters of N solutions each. Clusters could overlap. The fittest solution among each cluster is selected for the next generation.

In this work, an enhanced version of the tournament selection to preserve the elitism in the evolutionary process for each population without loosing the diversity has been proposed. This version is called N-way tournament-p, wherein, the best p% of the population (in terms of their fitness) are included in the selection process. By this way, the selection pressure of the BDDs is controlled such that any BDD whose size worsens after crossover/mutation operators is moved to the (100-p)% of the population, and thus, excluded from the tournament in that iteration.

D. IMPROVED GENETIC ALGORITHM COMPLEXITY ANALYSIS

The problem of finding the minimum BDD size is NP-complete, therefore, no known algorithm finds the global minimum in polynomial time. The algorithm does not guarantee to find the global minimum. It tries to find a sub-optimal solution within a reasonable time frame. The size of the problem is defined by the number of variables. The number of variables affects the execution time in several ways. The first way is the effect on search space size. The algorithm is all about searching for the variables' permutation that yields the smallest BDD size. As mentioned before, the algorithm is divided into two parts: global algorithm using GA and local search using sifting. Both parts search for the best permutation. The search space increases by order of $n!$ where n represents the number of variables. Bigger search space means harder to find the desired solution. Therefore, it is natural to make more iterations when a harder problem is expected. So the stopping criterion used was the exhausting number of iterations that is $c*n$ where c denotes a pre-defined constant. Inside the main loop, the process of building BDD and applying reduction (pruning) rules are repeated. This part is also affected by the number of variables but not as direct as the search space. Besides the number of variables, the number of product terms in the Boolean function also has an important influence on the size of the resulting BDD. For a small number of terms, the number of nodes is small and for a big number of terms also the number of nodes is small because there will be more chance to apply reduction rules. At some point in the middle, number of nodes is the biggest [26].

E. ALGORITHM LIMITATIONS

With the assumption of direct relationship between BDD size and the Quantum Cost (QC) of the resultant circuit (that is, the more nodes in a BDD, the more reversible gates in the output circuit, and thus a larger QC), a limitation is noticed in the proposed algorithm as there might be some cases wherein the QC is not proportional with the corresponding BDD size. To overcome such limitation, the BDD can be constructed and its companion reversible circuit is synthesized in each iteration. However, such an approach requires a complete BDD construction followed by synthesis in each iteration, which abruptly increases time and memory demands.

TABLE 2. Comparison between deterministic BDD reordering algorithms.

Benchmark	Window Permutation				Sifting Algorithm				Exact Algorithm			
	Line#	Gate#	QC	Runtime	Line#	Gate#	QC	Runtime	Line#	Gate#	QC	Runtime
4mod5-8	7	8	24	0	7	8	24	0	7	8	24	0.01
9symml-91	27	62	206	0	27	62	206	0	27	62	206	0
alu-9	9	16	48	0	7	9	29	0	7	9	29	0.01
apex2-101	1650	5883	19211	0.09	498	1746	5922	0.1	392	1291	4363	0.13
apex5-104	2131	6262	22210	0.02	1026	2914	10358	0.02	1023	2876	10216	0.05
bw-116	84	306	926	0	87	307	943	0	84	306	926	0
cordic-138	52	104	356	0.01	52	101	325	0.01	50	95	311	2.26
decod24-10	6	11	27	0	6	11	27	0	6	11	27	0
e64-149	1350	3850	8990	0.01	195	387	907	0.01	192	378	886	0.02
ex5p-154	205	645	1837	0.01	206	647	1843	0.01	206	647	1843	0.02
ham7-29	18	50	94	0	21	61	141	0	21	61	141	0.01
ham15-30	42	140	248	0.37	45	153	309	0.4	42	142	270	6.09
hwb5-13	27	85	265	0	28	88	276	0	28	88	276	0
hwb6-14	46	157	513	0	46	159	507	0	46	159	1928	0
hwb7-15	73	277	905	0	73	281	909	0	76	278	910	0
hwb8-64	115	444	1468	0.01	112	449	1461	0.01	114	440	1452	0.01
hwb9-65	172	692	2300	0.01	170	699	2275	0.01	169	679	2239	0.04
mini-alu-84	10	20	60	0	10	20	60	0	10	20	60	0
mod5adder-66	29	102	314	0	32	96	292	0	29	94	302	0
pdc-191	619	2087	6599	0.04	619	2080	6500	0.04	619	2087	6599	34.08
plus63mod4096-79	23	49	89	0.03	23	49	89	0.03	23	49	89	0.09
plus63mod8192-80	25	53	97	0.07	25	53	97	0.08	25	53	97	0.24
plus127mod8192-78	25	54	98	0.07	25	54	98	0.08	25	54	98	0.25
rd53-68	13	34	98	0	13	34	98	0	13	34	98	0.01
rd73-69	25	73	217	0	25	73	217	0	25	73	217	0.01
rd84-70	34	104	304	0	34	104	304	0.01	34	104	304	0.01
spla-202	548	1667	4831	0.03	489	1709	5925	0.03	483	1687	5863	40.59
sym6-63	14	29	93	0	14	29	93	0	14	29	93	0
sym9-71	27	62	206	0	27	62	206	0	27	62	206	0
xor5-195	6	8	8	0	6	8	8	0	6	8	8	0.01
Average	**247.07**	**777.80**	**2421.40**	**0.03**	**131.60**	**415.10**	**1348.30**	**0.03**	**127.43**	**396.13**	**1336.03**	**2.80**

Runtime unit:second, zero means very small time, QC: Quantum Cost.

V. EXPERIMENTAL TESTS & RESULTS

The previously described BDD reordering algorithms (sifting, window permutation, exact, genetic, and simulated annealing) have been evaluated in terms of the quantum cost of the synthesized reversible circuits. Furthermore, different variation and selection operators in the proposed genetic algorithm have been compared as well.

All experiments were conducted using Intel Xeon machine with 8 cores and a total RAM of 15.51 GB using Revkit simulator [3]. In this simulator, the published algorithm in [16] is implemented to map BDD nodes into a network of Toffoli gates. The CUDD[1] library for decision diagram operations is integrated in this simulator.

For evaluation, the compared algorithms have been applied on a number of public benchmarks[2] with different input sizes. Each benchmark is provided to the simulator as a PLA file (including target circuit truth table). Several common logic circuits (such as adders, ALUs, mods, and others) are included in those benchmarks.

It is important to mention that for stochastic algorithms (genetic and simulated annealing based BDD reordering), each experiment has been executed 100 times and the results of those experiments have been averaged. The maximum number of iterations has been set to $3 * n$ where n represents the number of variables in the input Boolean function that has to be realized using reversible circuitry. Moreover, the mutation-crossover mixing factor γ (used in Algorithm 1) has been set to 0.5. For tournament selection, 3-way tournament selection has been implemented with $p = 50\%$ of the worst individuals to be excluded from the tournament.

[1]http://hackage.haskell.org/package/cudd

[2]http://www.revlib.org/functions.php

A. COMPARISON BETWEEN BINARY DECISION DIAGRAMS REORDERING ALGORITHMS

Table 2 shows the line# (number of input/output variables after synthesis), gate count (number of Toffoli gates in the synthesis), QC, and the synthesis time for the following deterministic algorithms: window-permutation (window size has been set to 4), sifting, and exact algorithms.

As shown in Table 2, sifting algorithm outperforms window permutation algorithm in terms of QC by 44% in average and other evaluation metrics with almost the same runtime. This is reasonable because sifting algorithm considers the entire BDD during its greedy approach when reordering the nodes while windows permutation algorithm considers only the permutations of the different orders within the predefined window.

When applying exact algorithm, it slightly outperforms sifting algorithm since this algorithm considers more possible permutations of BDD's nodes ordering in a dynamic programming manner. However, sifting algorithm is almost 93× faster than the exact algorithm since the latter one has to test many solutions.

Table 3 shows a comparison between two stochastic based reordering algorithms. This includes both genetic algorithm (with roulette-wheel selection and PMX crossover) and simulated annealing. Both algorithms outperform the

TABLE 3. Comparison between genetic and simulated annealing based reordering algorithms.

Benchmark	Simulated Annealing				Genetic Algorithm			
	Line#	Gate#	QC	Runtime	Line#	Gate#	QC	Runtime
4mod5-8	7	8	24	0.01	7	8	24	0.0005
9symml-91	27	62	206	0.01	27	62	206	0.0205
alu-9	7	9	29	0	7	9	29	0.0092
apex2-101	267	810	2714	0.86	282	885	2747.78	6.533
apex5-104	878	2779	10221	2.23	1015	2874	9839.09	5.3136
bw-116	86	313	937	0.01	86	313	931.34	0.0101
cordic-138	52	101	325	0.07	49	94	311.44	0.3375
decod24-10	6	11	27	0	6	11	27	0.0006
e64-149	192	378	886	0.99	192	378	886	2.2166
ex5p-154	206	647	1843	0.06	206	647	1970	0.0328
ham7-29	21	61	141	0.01	21	61	141	0.0103
ham15-30	45	153	309	0.53	42	142	270	0.4439
hwb5-13	28	88	276	0	27	84	268	0.0101
hwb6-14	46	159	507	0.01	44	155	503	0.0106
hwb7-15	73	281	909	0.02	76	278	910	0.0207
hwb8-64	112	449	1461	0.04	114	440	1552	0.0409
hwb9-65	170	699	2275	0.06	169	679	2239	0.0714
mini-alu-84	10	20	60	0	10	20	60	0.0005
mod5adder-66	30	98	302	0.01	29	96	301.52	0.0101
pdc-191	619	2087	6599	0.51	619	2087	6598.02	0.7155
plus63mod4096-79	23	49	89	0.05	23	49	89	0.0483
plus63mod8192-80	25	53	97	0.1	25	53	97	0.0882
plus127mod8192-78	25	54	98	0.1	25	54	98	0.0921
rd53-68	13	34	98	0	13	34	98	0.0045
rd73-69	25	73	217	0	25	73	217	0.0105
rd84-70	34	104	304	0.01	34	104	304	0.0201
spla-202	482	1686	5858	0.35	482	1686	5960.72	0.5097
sym6-63	14	29	93	0	14	29	93	0.0101
sym9-71	27	62	206	0	27	62	206	0.0203
xor5-195	6	8	8	0	6	8	8	0.0017
Average	**118.53**	**378.83**	**1237.30**	**0.20**	**123.40**	**382.50**	**1232.83**	**0.55**

Runtime unit:second, zero means very small time, QC: Quantum Cost

deterministic algorithms in terms of QC and other evaluation metrics due to their random behavior in finding the near-optimal order of the BDD nodes. Simulated annealing has reduced the QC by around 7% if compared with the exact algorithm. In addition, it is 14× faster than the exact algorithm.

Genetic algorithm outperforms the exact algorithm by 8% in terms of the QC with being 5× faster. Genetic algorithm generally outperforms all the other algorithms for several reasons: (1) Sifting is exploited as a local search which increases the exploitation in this algorithm. (2) PMX is used as crossover operator which increases the exploration for finding near-optimal order of the BDD nodes. However, with integrating such operators, the genetic algorithm becomes slower if compared with simulated annealing and other algorithms.

B. COMPARISON BETWEEN SINGLE VARIATION OPERATORS

The genetic algorithm based reordering with integrating four variation operators including: PMX, CX, and OX, and inversion mutation have been implemented.

Table 4 shows the QC for the synthesized circuits when applying only one type of operators in the BDD reordering. As shown in the results, OX slightly outperforms PMX since it preserves the locations of some nodes in their optimal locations. Inversion mutation increases exploration in the search space which increases the opportunity of finding BDD with smaller size, and thus, less QC. CX outperforms all the other operators due to its property in preserving BDD nodes in their optimal locations in the offspring. CX results in around 4% reduction in QC if compared with the genetic algorithm published in [23].

The variations in runtime between different operators are small. However, it is observed that for large scale benchmarks (such as apex5-104 circuit), longer runtime is required for reversible synthesis. In this circuit, both PMX and OX have longer runtime. The larger QC (which indicates larger BDD), the more runtime is expected, since it takes longer time to map each node into a cascade of Toffoli gates.

C. COMPARISON BETWEEN MIXED OPERATORS

Table 5 shows the QC and the runtime for the synthesized reversible circuits with mixing crossover and mutation operators for BDD reordering. In each iteration in the GA optimization, either inversion mutation or other crossover (PMX, CX, and OX) is applied. The probability to choose the mutation is 50%.

Mixing operators has generally resulted in less QC. As shown in the results, mixing CX with inversion mutation outperforms all the other algorithms in terms of QC. This result is reasonable since CX preserves some portions of the permutations in their optimal locations during the optimization process while the inversion increases the exploration of finding better solutions in the search space. This creates a good balance between exploitation and exploration. Therefore, CX with inversion has outperformed the PMX (used in CUDD library) by around 5% in terms of the QC. In addition,

TABLE 4. Comparison between single operators.

Benchmark	Inversion Mutation		OX		PMX		CX	
	QC	Runtime	QC	Runtime	QC	Runtime	QC	Runtime
4mod5-8	24.24	0	24.24	0	24	0	24	0
9symml-91	208.06	0	208.06	0.02	206	0.021	206	0.02
alu-9	29.29	0	29.29	0.01	29	0	29	0.01
apex2-101	2170.2	0.09	2760.58	6.66	2747.78	6.53	1731.80	4.59
apex5-104	9927.7	0.02	9849.15	4.97	9839.09	5.31	9839.75	4.60
bw-116	942.19	0	941.42	0.01	931.34	0.01	932.86	0.01
cordic-138	314.26	0.01	313.91	0.38	311.44	0.34	311.2	0.31
decod24-10	27.27	0	27.27	0	27	0	27	0
e64-149	894.86	0.01	894.86	2.43	886	2.22	886	2.19
ex5p-154	1861.43	0.01	1861.43	0.053	1970	0.033	1843	0.04
ham7-29	142.41	0	142.41	0.01	141	0.01	141	0.01
ham15-30	272.70	0.37	272.70	0.46	270	0.44	270	0.42
hwb5-13	270.68	0	270.68	0.01	268	0.01	268	0.01
hwb6-14	508.03	0	508.03	0.01	503	0.01	503	0.01
hwb7-15	919.1	0	919.1	0.03	910	0.02	910	0.02
hwb8-64	1466.52	0.01	1466.52	0.051	1552	0.04	1452	0.05
hwb9-65	2261.39	0.01	2261.39	0.082	2239	0.07	2239	0.08
mini-alu-84	60.6	0	60.6	0	60	0	60	0
mod5adder-66	304.18	0	304.36	0.02	301.52	0.01	300.99	0.01
pdc-191	6664.99	0.04	6664.99	1.00	6598.02	0.72	6599	0.67
plus63mod4096-79	89.89	0.03	89.89	0.06	89	0.05	89	0.05
plus63mod8192-80	97.97	0.07	97.97	0.11	97	0.09	97	0.09
plus127mod8192-78	98.98	0.07	98.98	0.12	98	0.09	98	0.09
rd53-68	98.98	0	98.98	0.01	98	0	98	0.01
rd73-69	219.17	0	219.17	0.01	217	0.01	217	0.01
rd84-70	307.04	0	307.04	0.02	304	0.02	304	0.02
spla-202	5919.53	0.03	5919.13	0.66	5960.72	0.51	5861.17	0.45
sym6-63	93.93	0	93.93	0.01	93	0.01	93	0.01
sym9-71	208.06	0	208.06	0.02	206	0.02	206	0.02
xor5-195	8.08	0	8.08	0.01	8	0	8	0.01
Average	**1213.72**	**0.026**	**1230.74**	**0.57**	**1232.83**	**0.55**	**1188.16**	**0.46**

QC: Quantum Cost, Runtime Unit: Second , zero means very small time

generating smaller BDD sizes reduces the runtime as well since less number of BDD nodes has to be mapped to Toffoli gates. Thus, it can be seen that mixing CX with inversion mutation results in shorter runtime if compared with other algorithms.

TABLE 5. Comparison with mixed operators.

Benchmark	OX with Inversion		PMX with Inversion		CX with Inversion	
	QC	Runtime	QC	Runtime	QC	Runtime
4mod5-8	24.24	0	24.24	0	24	0
9symml-91	208.04	0.02	208.06	0.0202	206	0.02
alu-9	8317.25	0.01	29.29	0.01	29	0.01
apex2-101	1399.42	6.85	1866.95	6.58	1220.12	5.63
apex5-104	9410	5.35	9885.98	5.40	9841.52	4.92
bw-116	204	0.01	942.74	0.01	933.09	0.01
cordic-138	313.79	0.37	314.51	0.34	311.83	0.32
decod24-10	27.27	0	27.27	0	27	0
e64-149	894.77	2.55	894.86	2.48	886	2.33
ex5p-154	1861.25	0.06	1861.43	0.06	1843	0.05
ham7-29	142.40	0.02	142.41	0.01	141	0.01
ham15-30	272.67	0.46	272.7	0.44	270	0.43
hwb5-13	270.65	0.01	270.68	0.01	268	0.01
hwb6-14	507.98	0.01	508.03	0.01	503	0.01
hwb7-15	919.01	0.023	919.1	0.021	910	0.02
hwb8-64	1466.38	0.047	1466.52	0.041	1452	0.04
hwb9-65	2261.16	0.08	2261.39	0.07	2239	0.08
mini-alu-84	60.60	0.01	60.6	0	60	0
mod5adder-66	303.68	0.02	304.54	0.01	300.73	0.01
pdc-191	6664.34	1.082	6664.99	1.083	6599	0.88
plus63mod4096-79	89.88	0.063	89.89	0.06	89	0.05
plus63mod8192-80	97.96	0.11	97.97	0.10	97	0.1
plus127mod8192-78	98.97	0.12	98.98	0.11	98	0.1
rd53-68	98.97	0.01	98.98	0.01	98	0.01
rd73-69	219.14	0.011	219.17	0.01	217	0.01
rd84-70	307.01	0.02	307.04	0.02	304	0.02
spla-202	5918.87	0.67	5919.43	0.65	5861.44	0.55
sym6-63	93.92	0.010	93.93	0.01	93	0.01
sym9-71	208.04	0.02	208.06	0.02	206	0.02
xor5-195	8.08	0.01	8.08	0.01	8	0
Average	**1422.32**	**0.60**	**1202.26**	**0.59**	**1171.19**	**0.52**

QC: Quantum Cost, Runtime Unit: Second, zero means very small time

D. COMPARISON BETWEEN SELECTION OPERATORS

Selection operator plays an important role in the size of the obtained BDD after reordering. Table 6 shows a comparison between roulette wheel and 3-way tournament-50 selections when both integrated with CX and inversion mutation in the genetic algorithm for BDD reordering in the reversible circuit synthesis.

Tournament selection slightly outperforms roulette wheel selection in terms of QC. This clearly shows the impact of elitism induced by both the tournament per se and eliminating the worst 50% solutions from the tournament.

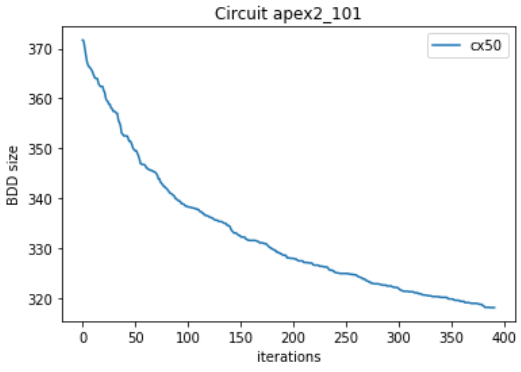


FIGURE 15. The average BDD size versus iteration # for mixing CX with inversion mutation. This plot is for apex2-101 benchmark.

Fig. 15 illustrates the average BDD size (for 100 experiments) versus the iteration # during the GA optimization process for mixing CX with inversion mutation and 3-way tournament-50 selection. Notice that with more iterations, the BDD size decays which indicates that better solutions can be achieved with more optimization iterations.

Fig. 16-a depicts the obtained BDD when GA algorithm is applied on benchmark sym6-63 while the obtained reversible

TABLE 6. Comparison with mixed operators.

Benchmark	Roulette Wheel		Tournament	
	QC	Runtime	QC	Runtime
4mod5-8	24	0	24	0.02
9symml-91	206	0.02	206	0.17
alu-9	29	0.01	29	0.02
apex2-101	1220.12	5.63	2007.4	14.15
apex5-104	9841.52	4.92	8961.83	11.07
bw-116	933.09	0.01	933.98	0.07
cordic-138	311.83	0.32	309.65	1.78
decod24-10	27	0	27	0.01
e64-149	886	2.33	886	4.43
ex5p-154	1843	0.05	1843	0.39
ham7-29	141	0.01	141	0.08
ham15-30	270	0.43	270	1.04
hwb5-13	268	0.01	268	0.04
hwb6-14	503	0.01	503	0.09
hwb7-15	910	0.02	910	0.19
hwb8-64	1452	0.04	1232	0.38
hwb9-65	2239	0.08	2239	0.7
mini-alu-84	60	0	60	0.02
mod5adder-66	300.73	0.01	301.55	0.07
pdc-191	6599	0.88	6599	6.93
plus63mod4096-79	89	0.05	89	0.27
plus63mod8192-80	97	0.1	0.36	0.01
plus127mod8192-78	98	0.1	98	0.36
rd53-68	98	0.01	98	0.02
rd73-69	217	0.01	217	0.08
rd84-70	304	0.02	304	0.14
spla-202	5861.44	0.55	5860.38	4.26
sym6-63	93	0.01	93	0.04
sym9-71	206	0.02	206	0.16
xor5-195	8	0	8	0.02
Average	**1171.19**	**0.52**	**1157.51**	**1.57**

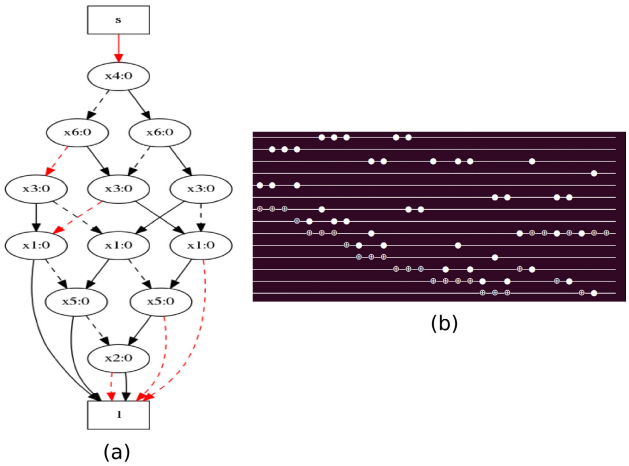


FIGURE 16. (a) The resultant BDD when applying GA algorithm on benchmark sym6-63 where the chromosome representation of the solution is (4,6,3,1,5,2). (b) The resultant reversible circuit when applying the mapping phase on the BDD shown in (a).

circuit after mapping the nodes of that BDD into cascades of reversible gates is shown in Fig.16-b.

The synthesis of reversible circuits (like the one shown in Fig.16-b) is then mapped into a cascade of quantum gates to realize the required quantum circuit using any available quantum technology. Detailed description of building quantum gates from reversible gates schema can be find in [44]. As the proposed work has successfully obtained a reversible circuit with low Quantum Cost (QC) for that circuit, the algorithms published in [44] can be then applied to obtain a reversible circuit with low cost in terms of the number of elementary quantum elements. Furthermore it is important to mention that this approach needs to be applied on problems that cannot be solved effectively using classical computers. Some NP-hard problems like prime numbers factorization, sophisticated computational models like market behaviour models, drug development, and many other applications, require quantum computers to be solved in a reasonable time. Therefore, the proposed work makes things ready to solve such problems since the resultant reversible circuit can be easily mapped into a quantum circuit with low cost, which in turn, can be used to solve such problems in a reasonable computation time, if compared with classical computers.

VI. CONCLUSION AND FUTURE WORKS

In this work, a comparative study between several BDD reordering approaches and their effect on reversible circuit synthesis is executed. An estimation for each circuit in terms of gate count, line number, transistor and quantum costs is conducted. The outcomes have emerged that exploiting meta heuristic optimization algorithms (specifically genetics and simulated annealing) in reordering of BDDs leads to smaller BDD size which turns out to a lower overall cost. Thereafter, the work proposed a Genetic Algorithm (GA)-based BDD reordering algorithm with supplementary crossover, mutation, and selection operators. The work analyzed the impact of different operators on the generated BDD, and the quantum cost of its corresponding reversible circuit. Experimental results have shown that mixing Cycle Crossover (CX) with inversion mutation and tournament selection outperforms other operators in terms of the quantum cost of the synthesized reversible circuits. Such mixing preserves the optimal locations of some variables in the BDD alongside with exploration of new solutions with preserving the elite ones due to the mutation and selection operators. The synthesis time has been found to be generally correlated with the BDD size, since the more nodes, the longer time required to map those nodes into cascades of Toffoli gates.

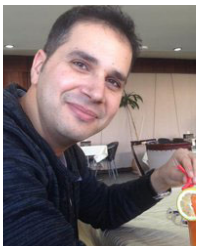
Although the smaller BDD size often the less quantum cost, this might not be always true. Because of this, one of the future planned works is to comprise the synthesis algorithm mapping phase into the GA optimization with fixing minimizing the control lines number as the main objective function.

REFERENCES

[1] R. Wille and R. Drechsler, *Towards a Design Flow for Reversible Logic*. Amsterdam, The Netherlands: Springer, 2010.
[2] N. K. Misra, B. Sen, S. Wairya, and B. Bhoi, "A novel parity preserving reversible binary-to-bcd code converter with testability of building blocks in quantum circuit," in *Proc. 2nd Int. Conf. Comput. Intell. Inform.*, V. Bhateja, J. M. R. Tavares, B. P. Rani, V. K. Prasad, and K. S. Raju, Eds. Singapore: Springer, 2018, pp. 383–393.
[3] M. Soeken, S. Frehse, R. Wille, and R. Drechsler, "Revkit: An open source toolkit for the design of reversible circuits," in *Reversible Computation*, A. De Vos and R. Wille, Eds. Berlin, Germany: Springer, 2012, pp. 64–76.
[4] D. M. Miller and M. A. Thornton, "QMDD: A decision diagram structure for reversible and quantum circuits," in *Proc. 36th Int. Symp. Multiple-Valued Log. (ISMVL)*, May 2006, p. 30.

[5] T. N. Sasamal, A. K. Singh, and A. Mohan, "Reversible logic circuit synthesis and optimization using adaptive genetic algorithm," *Procedia Comput. Sci.*, vol. 70, pp. 407–413, Jan. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915032184
[6] A. Kole and K. Datta, "Improved ncv gate realization of arbitrary size toffoli gates," in *Proc. 30th Int. Conf. VLSI Design 16th Int. Conf. Embedded Syst. (VLSID)*, Jan. 2017, pp. 289–294.
[7] P. Kerntopf, "A new heuristic algorithm for reversible logic synthesis," in *Proc. 41st Annu. Conf. Design Autom. DAC*, Jul. 2004, pp. 834–837.
[8] M. Fujita, *Logic Synthesis for Reversible Circuits*. Singapore: Springer, 2020, pp. 65–91, doi: 10.1007/978-981-13-8821-7_5.
[9] J. Hu, Y. Lin, and X. Zhang, "Reversible logic synthesis using Gröbner base," in *Proc. IEEE 2nd Int. Conf. Electron. Technol. (ICET)*, May 2019, pp. 229–232.
[10] D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis," in *Proc. 40th Conf. Design Autom. DAC*, Jun. 2003, pp. 318–323.
[11] M. Saeedi and I. L. Markov, "Synthesis and optimization of reversible circuit-a survey," *ACM Comput. Surv.*, vol. 45, no. 2, pp. 21:1–21:34, Mar. 2013, doi: 10.1145/2431211.2431220.
[12] A. Zulehner and R. Wille, "Exploiting coding techniques for logic synthesis of reversible circuits," in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2018, pp. 670–675.
[13] R. Ray, A. Deka, and K. Datta, "Exact synthesis of reversible logic circuits using model checking," 2017, *arXiv:1702.07470*. [Online]. Available: http://arxiv.org/abs/1702.07470
[14] A. Chattopadhyay, A. Littarru, L. Amaru, P.-E. Gaillardon, and G. D. Micheli, "Reversible logic synthesis via biconditional binary decision diagrams," in *Proc. IEEE Int. Symp. Multiple-Valued Log.*, May 2015, pp. 2–7.
[15] M. Mohammadi and M. Eshghi, "Heuristic methods to use don't cares in automated design of reversible and quantum logic circuits," *Quantum Inf. Process.*, vol. 7, no. 4, pp. 175–192, Aug. 2008, doi: 10.1007/s11128-008-0081-x.
[16] R. Wille and R. Drechsler, "BDD-based synthesis of reversible logic for large functions," in *Proc. 46th Annu. Design Autom. Conf. ZZZ DAC*, 2009, pp. 270–275, doi: 10.1145/1629911.1629984.
[17] K. Podlaski, "Reversible circuit synthesis using binary decision diagrams," in *Proc. MIXDES 23rd Int. Conf. Mixed Design Integr. Circuits Syst.*, Jun. 2016, pp. 235–238.
[18] A. Zulehner and R. Wille, "One-pass design of reversible circuits: Combining embedding and synthesis for reversible logic," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 5, pp. 996–1008, May 2018.
[19] D. Große, X. Chen, G. W. Dueck, and R. Drechsler, "Exact sat-based toffoli network synthesis," in *Proc. 17th great lakes Symp. Great lakes Symp. VLSI - GLSVLSI*, 2007, pp. 96–101, doi: 10.1145/1228784.1228812.
[20] B. Bollig and I. Wegener, "Improving the variable ordering of OBDDs is NP-complete," *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 993–1002, Sep. 1996.
[21] B. Bollig, M. Löbbing, and I. Wegener, "Simulated annealing to improve variable orderings for OBDDs," in *Proc. Int. Workshop Log. Synth*, 1995, p. 5.
[22] B. Abdalhaq, A. Awad, and A. Hawash, "A swarm based binary decision diagram (BDD) reordering optimizer for reversible circuit synthesis," in *Proc. 15th Design Technol. Integr. Syst. Nanosc. Era (DTIS)*, Apr. 2020, pp. 1–6.
[23] W. Lenders and C. Baier, "Genetic algorithms for the variable ordering problem of binary decision diagrams," in *Foundations of Genetic Algorithms*, A. H. Wright, M. D. Vose, K. A. De Jong, and L. M. Schmitt, Eds. Berlin, Germany: Springer, 2005, pp. 1–20.
[24] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Proc. Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 1993, pp. 42–47.
[25] S. J. Friedman and K. J. Supowit, "Finding the optimal variable ordering for binary decision diagrams," in *Proc. 24th ACM/IEEE Conf. Proc. Design Autom. Conf. DAC*, Jun. 1987, pp. 348–356.
[26] M. Raseen, P. W. Chandana Prasad, and A. Assi, "An efficient estimation of the ROBDD's complexity," *Integration*, vol. 39, no. 3, pp. 211–228, Jun. 2006.
[27] P. Manna, D. K. Kole, H. Rahaman, D. K. Das, and B. B. Bhattacharya, "Reversible logic circuit synthesis using genetic algorithm and particle swarm optimization," in *Proc. Int. Symp. Electron. Syst. Design (ISED)*, Dec. 2012, pp. 246–250.
[28] K. Podlaski, "Reversible circuit synthesis with particle swarm optimization using crossover operator," in *Proc. 22nd Int. Conf. Mixed Design Integr. Circuits Syst. (MIXDES)*, Jun. 2015, pp. 375–379.
[29] D. Maslov, G. W. Dueck, and D. M. Miller, "Synthesis of fredkin-toffoli reversible networks," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 6, pp. 765–769, Jun. 2005.
[30] D. Maslov, G. W. Dueck, and D. M. Miller, "Toffoli network synthesis with templates," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 6, pp. 807–817, Jun. 2005.
[31] M. Hawash, M. Perkowski, S. Bleiler, J. Caughman, and A. Hawash, "Reversible function synthesis of large reversible functions with no ancillary bits using covering set partitions," in *Proc. 8th Int. Conf. Inf. Technol., New Generat.*, Apr. 2011, pp. 1008–1013.
[32] M. Hawash, B. Abdalhaq, A. Hawash, and M. Perkowski, "Application of genetic algorithm for synthesis of large reversible circuits using covered set partitions," in *Proc. 4th IEEE Int. Symp. Innov. Inf. Commun. Technol. (ISIICT)*, vol. 1, Nov. 2011, pp. 1–7.
[33] L. Biswal, R. Das, C. Bandyopadhyay, A. Chattopadhyay, and H. Rahaman, "A template-based technique for efficient Clifford+T-based quantum circuit implementation," *Microelectron. J.*, vol. 81, pp. 58–68, Nov. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026269218300430
[34] R. Wille and R. Drechsler, "Effect of BDD optimization on synthesis of reversible and quantum logic," *Electron. Notes Theor. Comput. Sci.*, vol. 253, no. 6, pp. 57–70, Mar. 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1571066110000198
[35] J. Lee, Y.-H. Ye, X. Huang, and R.-L. Yang, "Binary-decision-diagram-based decomposition of Boolean functions into reversible logic elements," *Theor. Comput. Sci.*, vol. 814, pp. 120–134, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397520300463
[36] M. Sarvaghad-Moghaddam, A. A. Orouji, Z. Ramezani, I. Sadegh Amiri, and A. Mahdavi Nejad, "Reversible gates in emerging quantum-dot cellular automata technology: An innovative approach to design and simulation," 2018, *arXiv:1803.11017*. [Online]. Available: http://arxiv.org/abs/1803.11017
[37] A. Al-Rabadi, *Reversible Logic Synthesis: From Fundamentals to Quantum Computing*. Berlin, Germany: Springe, 2004.
[38] S. M. Saeed, X. Cui, R. Wille, A. Zulehner, K. Wu, R. Drechsler, and R. Karri, "Towards reverse engineering reversible logic," 2017, *arXiv:1704.08397*. [Online]. Available: http://arxiv.org/abs/1704.08397
[39] S. H. Gerez, *Algorithms for VLSI Design Automation*, 1st ed. New York, NY, USA: Wiley, 1999.
[40] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Proc. Conf. Proc. 27th ACM/IEEE design Autom. Conf. DAC*, 1990, pp. 40–45, doi: 10.1145/123186.123222.
[41] R. Drechsler, N. Drechsler, and W. Günther, "Fast exact minimization of BDDs," in *Proc. 35th Annu. Conf. Design Autom. Conf. - DAC*, 1998, pp. 200–205, doi: 10.1145/277044.277099.
[42] O. Abdoun and J. Abouchabaka, "A comparative study of adaptive crossover operators for genetic algorithms to resolve the traveling salesman problem," 2012, *arXiv:1203.3097*. [Online]. Available: http://arxiv.org/abs/1203.3097
[43] R. K. Naveen kumar, Karambir, "A comparative analysis of pmx, cx and ox crossover operators for solving travelling salesman problem," *Int. J. Latest Res. Sci. Technol.*, vol. 1, pp. 98–101, 2012.
[44] D. Grosse, R. Wille, G. W. Dueck, and R. Drechsler, "Exact synthesis of elementary quantum gate circuits for reversible functions with don't cares," in *Proc. 38th Int. Symp. Multiple Valued Log. (ismvl)*, May 2008, pp. 214–219.

BAKER K. ABDALHAQ received the Ph.D. degree in computer science from the Autonomous University of Barcelona. His Ph.D. research was on data assimilation of wildfire simulators. He is currently an Assistant Professor of computer science with An-Najah National University. He used meta-heuristic and evolutionary algorithms, the same techniques used in artificial intelligence (AI) and machine learning (ML). Simulators, in general, are based on assumptions and parameters that are hard or impossible to measure accurately in real life. Wildfire simulators depend on wind speed and direction that is extremely hard to predict and is highly variable. The vegetation (fuel) conditions and characteristics are another important input to the simulator that affects its accuracy and is difficult to collect at sufficient resolution. The conclusion of his work was that a simulator can adapt and learn from data.



AHMED AWAD (Member, IEEE) received the bachelor's degree in computer systems engineering and the master's degree (Hons.) in computing from Birzeit University, in 2009 and 2011, respectively, and the Ph.D. degree from the Department of Communications and Computer Engineering, Tokyo Institute of Technology, Japan, in 2016. His Ph.D. thesis focuses on developing novel algorithms for mask optimization problem in optical lithography for large scale integrated digital circuits. He published over 14 conference proceedings and journal articles. He worked as a Quality Assurance Engineer for CISCO Systems, from 2009 to 2013. He is currently an Assistant Professor with the Department of Computer Sciences, An-Najah National University. His research interests include network-related optimization problems, the Internet-of-Things (IoT) security, reversible logic synthesis and optimization, optical lithography and OPC, and advanced memory design problems. He is a member of IEICE, IPSJ, and the Jordanian Engineering Association.

AMJAD HAWASH received the bachelor's degree in computer science from An-Najah National University, in 1996, the master's degree in computer information systems from Yarmouk University, in 2006, and the Ph.D. degree from the Sapienza University of Rome, in 2015. His master's thesis was related to automating the configuration of Cisco routers (set of features) by moving from the command prompt to GUI. His Ph.D. thesis was related to introducing groups to an annotation system. His Ph.D. thesis contained a study of involving ontologies to execute a matching between users (represented by their annotation tags) with the set of system groups (represented by ontologies) in the process of suggesting groups of annotators to users and vice versa. During his Ph.D. degree, he succeeded in publishing ten publications. His current research interest includes the set of computer science relate researches like Arabic ontologies, graph analysis, and reversible circuits synthesis in which a set of publications are done and ongoing research is going on.