A Fast Binary Decision Diagram (BDD)-based Reversible Logic Optimization Engine Driven by Recent Meta-heuristic Reordering Algorithms

Baker Abdalhaq, Ahmed Awad, Amjad Hawash

Department of Information & Computer Science, An-Najah National University, Nablus, Palestine.
{baker,ahmedawad,amjad}@najah.edu

Abstract

Reversible logic has recently gained a remarkable interest due to its information lossless property, which minimizes power dissipation in the circuit. Furthermore, with their natural reversibility, quantum computations can profit from the advances in reversible logic synthesis, as the latter can be easily applied to map practical logic designs to quantum architectures. Although numerous algorithms have been proposed to synthesize reversible circuits with low cost, the increasing demands for scalable synthesis techniques represent a serious barrier in the synthesis process. Furthermore, the enhanced reliability of the synthesized circuits comes at the cost of redundancy in the quantum architecture of the gates composing that circuit, which increases the overall manufacturing cost for fault-tolerant circuits. Binary Decision Diagram (BDD) based synthesis has demonstrated a great evidence in reversible logic synthesis, due to its scalability in synthesizing complex circuits within a reasonable time. However, the cost of the synthesized circuit is roughly correlated to its corresponding BDD size. In this paper, we propose a fast reversible circuit synthesis methodology driven by a BDD-reordering optimization engine implemented by recent meta-heuristic optimization algorithms. Experimental results show that Genetic Algorithm (GA) based reordering supported with Alternating Crossover (AX) and swap mutation outperforms others as it is the least destructive for low-cost BDDs during the optimization recipe.

Keywords: Quantum Cost (QC), Binary Decision Diagram (BDD), Swarm, Genetic Algorithm (GA), Crossover, Mutation.

1. Introduction

With traditional irreversible logic, each bit loss results in energy dissipation regardless to the underlying transistor technology. Since millions of operations are executed in matter of seconds to meet the high performance computing demands, excessive energy dissipation is expected, which badly impacts the reliability of the chip [1].

With the continuous progress in fabrication process, the practical application of quantum computers has become a tangible prospect. With their multiple states representation using qubits, quantum circuits enable enormous speedup in computations. However, traditional synthesis approaches are not feasible to be applied for mapping logic designs into quantum architectures [2].

To keep pace with the advances in the fabrication process towards quantum computing alongside with low power computations, reversible logic proposes a promising direction in this manner. First of all, reversible logic imposes a bijective mapping between input and output combinations. Thus, theoretically there is no energy dissipation with such a one-to-one mapping. Furthermore, quantum circuits are inherently reversible, therefore, employing the advances of reversible computations in the field of quantum computing makes a significant step ahead in the emerging process towards quantum computers [3].

Synthesizing reversible circuits requires re-developing the synthesis approaches of traditional irreversible circuits. This is caused by the limitations imposed to achieve the desired one-to-one mapping in the resultant circuit. Accordingly, the number of input vectors should equal the number of output vectors. This means, additional inputs and/or outputs might be added to preserve the reversibility of the circuit [1][4]. In addition, feedback and fan-out paths are prohibited in reversible circuits. Consequently, a reversible circuit should be composed of gates that satisfy those requirements. Such gates are known as reversible gates whose

proper cascading forms the desired reversible circuitry. Examples of such gates are: NOT, CNOT, Toffoli, and Fredkin gates [5]. Fig. 1 illustrates a reversible circuit that is constructed using one Toffoli gate and 6 CNOT gates.

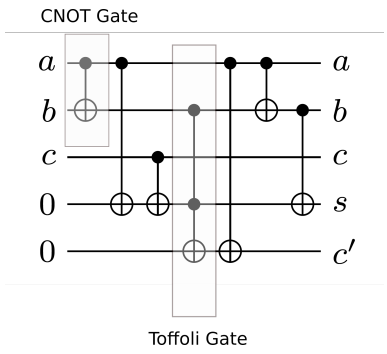

Fig. 1: A reversible circuit that consists of one Toffoli gate and 6 CNOT gates with two additional constant inputs.

A reversible circuit has to be designed with low cost when later mapped to a quantum circuit. The number of elementary quantum gates that compose a quantum circuit is known as the Quantum Cost (QC). This metric should be minimized. In fault-tolerant quantum computing, the enhanced reliability of the circuits to be synthesized is achieved at the cost of redundancy in the quantum architecture of those circuits. In particular, each Toffoli gate is decomposed into many fault-tolerant T gates to increase the circuit reliability. Therefore, in fault tolerant computing, the cost of reversible circuits is evaluated in terms of the number of Toffoli gates whose high values indicate a higher manufacturing cost of the circuit [6][7]. Therefore, the number of those high-cost Toffoli gates in the synthesized circuit should be minimized while preserving the required reliability of the circuit. Synthesis time forms another crucial cost metric that should be minimized as well. The synthesis time becomes a serious concern when repeatedly synthesizing large-scale circuits within the optimization recipe [8][9].

Enormous effort is evident in the literature to develop algorithms for re-

versible logic synthesis for a given Boolean function, such as as Transformation Based Synthesis (TBS), Add-Invert Graphs (AIGs), and Lookup Table (LUT) based algorithms. Both exact and heuristic based approaches have been exploited in this manner to find the proper network of reversible gates that form the required circuitry [10][11][12]. However, to keep pace with the advances in the fabrication process, scalabable and fast synthesis algorithms are required.

As a compact canonical representation of Boolean functions, Binary Decision Diagrams (BDDs) have been widely used in the synthesis, optimization, and equivalence checking for both combinational and sequential circuits. BDDs have been adopted as well in network reliability analysis for large networks through heuristically ordering network components, constructing equivalent BDD for the resultant probabilistic graph, and then evaluating the generated BDD to obtain the network reliability [13][14]. In the field of repairable systems, BDDs have been employed in reliability analysis through representing the system components into a Boolean function, whose companion BDD is then utilized to calculate the total Mean Failures Frequency (MFF) of the system [15]. In addition, when a system state is represented in terms of the states of its components, i.e., Boolean variables, BDDs are exploited to analyze the influence of variable changes on the system reliability, instead of using logical differential calculus whose complexity abruptly increases for large-dimensional systems [16]. Another application of BDDs in reliability analysis is to construct Fault Trees (FTs) for logic circuits with substantial saving of computational complexity in terms of memory and space. This turns out into faster estimation for the entire system reliability [17][18].

BDDs have demonstrated great evidence in the synthesis of reversible circuits [19]. In BDD-based synthesis for reversible circuits, a predefined lookup table is constructed in an early stage. Thereafter, the BDD for the function to be realized is constructed. Each node in the BDD is then mapped into a cascade of Toffoli gates following the pre-constructed lookup table. Therefore, the size of the resultant circuit is bounded by the BDD size which can be effectively reduced using BDD reordering algorithms combined with BDD reduction rules

[20].

Finding the optimal order of variables in a BDD that results in the least size is NP complete problem [21]. Several algorithms have been proposed to solve this problem such as: Satisfiability Solver (SAT) based [22], sifting algorithm [23], exact algorithm [24], and meta-heuristic based algorithms such as evolutionary algorithms [25] and swarm based optimization algorithms [26]. However, the increased complexity in the functions to be synthesized increases the demands for better exploration and exploitation techniques when searching for the near-optimal order of the variables in the BDD. Therefore, exploiting recent swarm and Genetic Algorithm (GA) operators in BDD reordering problem is expected to reduce the resultant BDD size, and thus, reducing the overall QC of the output circuit. Similarly, in fault-tolerant quantum computing, reducing the size of the BDD is expected to reduce the number of Toffoli gates, when each BDD node is mapped into a cascade of Toffoli gates, and thus, reducing the manufacturing cost of such reliable reversible circuits.

In this paper, we propose a fast BDD-based reversible circuit synthesis methodology driven by a BDD-reordering optimization engine implemented using both recent swarm algorithms and GA algorithm supported with effective crossover/mutation operators. To speedup the synthesis time, the proposed algorithm uses the BDD size as an evaluation criterion in the optimization recipe in accordance with a linearized relationship between the QC and the BDD size. Our contributions are summarized as follows:

- We propose a fast BDD-based synthesis algorithm for reversible circuits driven by a BDD-reordering optimization engine under the guidance of BDD size as an evaluation metric in the proposed algorithm.

- We integrate recent swarm optimization algorithms into the BDD-reordering optimizer. This includes: Particle Swarm Optimization (PSO), Cuckoo Search (CS), Firefly Algorithm (FFA), Grey Wolf Optimization (GWO), Moth Flame Optimization (MFO), Bat Algorithm (BAT), Salp Swarm Algorithm (SSA), and Whale Optimization Algorithm (WOA).

- We integrate Genetic Algorithm (GA) supported with recent effective crossover/mutation operators into the BDD-reordering optimization engine. This includes: Ordered Crossover (OX), Partially Mapped Crossover (PMX), Cycle Crossover (CX), Alternating Crossover (AX), shuffle, invert, and swap mutations.

- We evaluate the different algorithms when integrated with our proposed optimization engine in terms of the size of the resultant BDD. This evaluation has been done on the public benchmarks.

It is worth to mention that the proposed methodology for BDD reordering can be easily extended for other applications in reliability analysis. For example, when BDD is employed for network reliability analysis as published in [13], the smaller the BDD size, the faster evaluation model to calculate the network reliability. Similarly, when BDD is constructed for computing a system availability as in [15][16], the smaller BDD size, the less computations to be performed, and thus, the faster reliability estimation.

The rest of this paper is organized as follows: Related work is presented in Section II. Section III presents problem formulation and evaluation metrics. Reversible circuit synthesis flow and the proposed optimization methodology are presented in Section IV. Experimental results are presented in Section V and Section VI concludes this paper.

2. Previous Work

The literature contains a set of algorithms that have been proposed to optimize synthesis of reversible circuits including but not limited to: exact synthesis, Transformation Based Synthesis (TBS) [11], and Boolean Satisfiability Solver (SAT) based synthesis. However, these algorithms suffer from the slow convergence especially when complex and big size circuits have to be synthesized.

Heuristic algorithms have been well exploited to converge for a near-optimal reversible circuits synthesis. As an example, Particle Swarm Optimization and

Genetic Algorithm have been found to output satisfactory reversible circuits synthesis as published in [27, 28]. However, these algorithms might slowly converge for small-scale circuits and they need to be integrated with other algorithms that work as local search in order to output low cost circuits within a reasonable time. [22].

Binary Decision Diagrams (BDDs) have been exploited as well in the construction of reversible circuits. The function to be synthesized is represented using BDD whose nodes are then converted to their equivalent reversible gates. This approach has provided an opportunity to synthesize low cost reversible circuits especially after applying the BDD reduction rules in order to minimize the number of reversible gates and hence decrease the total Quantum Cost (QC) for those output reversible circuits [1][29]. In this manner, the algorithm presented in [20] proposes a well modified version of mapping based synthesis, while the paths of BDD were used to synthesis the reversible circuits instead of using the corresponding truth table. The work presented in [30] is related to synthesizing huge circuits that are composed of more than a hundred of variables for their Boolean functions. The authors of the work presented a technique that utilizes BDDs to synthesis reversible circuits. It has been found that there is a relation between BDD size and the number of reversible gates of the synthesized circuit. This allows to transfer theoretical results known from BDDs to reversible circuits.

The cost of the synthesized reversible circuity is influenced by the BDD size that is related directly to the order of variables in the BDD. Several greedy algorithms have been proposed to reorder the BDD nodes as a try to minimize its size and hence minimize the resultant reversible circuit. Sifting and Windowing algorithms are two examples for such greedy algorithms [23]. Other algorithms were invented based on dynamic programming [24], while others are heuristic-based ones such as Genetic Algorithm (GA) and Simulated Annealing (SA) algorithms. However, heuristic algorithms have been considered good choices to deal with such an NP-complete problem. [31][25].

Different techniques related to GA are also used in reordering of BDD nodes.

The work presented in [32] examined genetic algorithm with three crossover operators: order, cycle and partially mapped. The algorithm has been proposed for shared ordered BDDs minimization. The results of the work have been compared taking into account the three mentioned operators and tested for Multi-input Adder Benchmark circuits. However, integrating other operators is required to keep pace with the increased complexity in the Boolean functions to be synthesized.

Ordered Binary Decision Diagrams (OBDDs) were the base for the work presented in [33]. Authors discussed an approach for the optimization of OBDDs based on GA. They considered completely specified Boolean Functions (BFs). Their method is based on executing a specific reordering heuristics and combine them with principles of genetic algorithms as a try to determine the good reordering of variables. However, the population has been considered to be small in this algorithm while advanced complex functions might include hundreds of variables to be reordered during BDD construction. In addition, the synthesis of reversible circuits might require dealing with partially specified Boolean functions that should be embedded with a reversible circuit.

The work presented in [34] is related to GA application to synthesizing large-scale reversible circuits based on Constraint Satisfaction Problems (CSP). Authors of the work were concerned in finding a set of optimal solutions considering minimizing the costs of synthesized circuits within a huge solutions space. The work was related to investigate and examine 3 searching algorithms: Random, Tabu and Genetic algorithms. The investigation process was based on a criterion to compare the three algorithms depending on the QCs that were represented by the number of reversible gates for the synthesised reversible circuits.

Authors of the work presented in [35] studied a set of optimization algorithms that are based on swarm intelligence. Authors discovered that there have been some distinguishable advantages over traditional approaches when optimization is used. Their analysis of the used algorithms focused on the way the algorithms achieve exploration and exploitation as well as the basic components of evolutionary operators like crossover, mutation and the selection of

the most suitable genes. However, despite the achievement gained, these recent algorithms have not been exploited in the process of BDD reordering of nodes.

As an extension of BDDs, Quantum Multiple-Valued Decision Diagrams (QMDDs) have been proposed to synthesize quantum circuits in Multi-Valued Logic (MVL) with exploiting the quantum phonemena, such as quantum superposition and and quantum entanglement [36][37]. Therefore, QMDDs have r^2 transition edges instead of two possible edges in the classical BDDs, where r represents the radix of the multi-valued logic being represented. This aims to tackle the limitation of the binary data amount that can be transferred in the circuit. In this manner, sifting algorithm with some other heuristics have been employed to reorder the nodes of the QMDD such that its companion size is minimized, which turns out into reducing the cost of the synthesized circuit [38]. However, modern meta-heuristic algorithms have not been integrated and only small-scale benchmarks have been tested.

In the work published in [9] a comparative study is conducted between different BDD reordering algorithms in terms of their impact on the cost of the synthesized reversible circuits. This includes comparing: sifting algorithm, windowing, exact algorithm, simulated annealing, and GA algorithm with exploiting several crossover, mutation, and selection operators. However, this work does not consider recent operators in GA including Alternating Crossover (AX), swap mutation, and shuffle mutation. Moreover, the proposed methodology deals with the problem in the discrete domain while recent swarm optimization algorithms (working in the continuous domain) have not been considered. Therefore, in this paper, a methodology is proposed to allow employing continuous domain operators through proper discretization of a vector of real numbers to be applied onto BDD reordering problem. This includes recent swarm optimization algorithms.

This work is an extension for the work published in [26]. While the work published in [26] considers only swarm optimizers, this work effectively conducts a comparison between GA (supported with recent mutation/crossover operators) and swarm optimizers in terms of the resultant BDD size. In addition, large

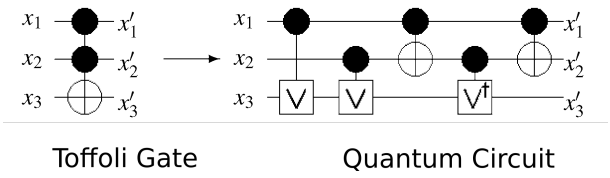

Fig. 2: A 3×3 Toffloli gate quantum circuit with a QC equals to 5.

size benchmarks are considered in the evaluation in this work.

3. Preliminary Terminologies

In the following, we briefly describe the basic terminologies and concepts with respect to synthesis of reversible circuits as well as the cost metric formulations.

3.1. Terminology of Reversible Logic

A reversible gate g with $n \times n$ inputs/outputs links can be used to realize a reversible mapping (function) $f : B^n \rightarrow B^n$ such that $B \in \{0, 1\}$, so that the mapping done by the funciton f is bijective between the n-bit input and output vectors.

Depending on the previous definition for a reversible gate, a reversible circuit $G = (g_0, g_1, \cdots, g_{m-1})$ can be defined as a sequence of reversible gates that reflects a reversible Boolean function.

3.2. Cost Metrics of Reversible Logic

The elementary quantum operations needed to realize a reversible gate can be used to compute the reversible gate Quantum Cost (QC). Eq.(1) is used to compute the QC for Toffoli reversible gate g with C control lines [39]. Fig. 2 represents the quantum circuit realizations for a 3×3 Toffloli gate (equals to 5). The summation for the QC for all reversible gates that compose the reversible circuit is the total QC for that reversible circuit[1].

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

For reversible circuit reliability, many T gates are required in the form of Toffoli gates to maximize the reliability the circuit, which in turn, increases the cost for each Toffoli gate. Therefore, the more Toffoli gates cascaded together to form the circuit, the higher manufacturing cost of the circuit. Consequently, in fault-tolerant quantum computing, the number of Toffoli gates, denoted by $TOF\#(G)$, of a reversible circuit G defines its cost.

3.3. Terminology of Binary Decision Diagram (BDD)

A BDD is defined as a directed acyclic graph $G(V, E)$ where a given node $v \in V$ is either a non-terminal (internal) or terminal (leaf) node. Each terminal node has the value of either 0 or 1 with no outgoing edges. Whereas, a non-terminal node is augmented by one of the input variables of the Boolean function being represented, where Shannon decomposition is applied according to Eq. (2), where x_i represents an input variable, f represents the Boolean function being represented by the given BDD, f_{x_i} represents the function f when $x_i = 1$ (positive cofactor), and $f_{\bar{x_i}}$ represents the function f when $x_i = 0$ (negative cofactor).

$$f = \bar{x_i} f_{\bar{x_i}} + x_i f_{x_i} \tag{2}$$

To clarify the idea of Shannon decomposition, Fig. 3 below illustrates the idea by a sample function $f = x_1 \oplus x_2 \cdot x_3$, where the XOR operation is denoted by $\oplus$.

Each BDD has its own size that is represented by the number of internal nodes (non-terminal ones) it contains. A sample BDD in Fig. 3 has a size of 5. However, a set of well known reduction rules (shared nodes, complemented edges, and nodes reordering) can be applied on a BDD to reduce its size without impacting its reliability for the Boolean function it represents. Changing the order of variables for a Boolean function leads to a change in the construction of the corresponding BDD and hence a change in its size. Due to this, several researches were conducted with a common interest in reordering the variables of the Boolean functions. To illustrate the idea, take the function $f = x_1 \cdot x_2 + x_3 \cdot$

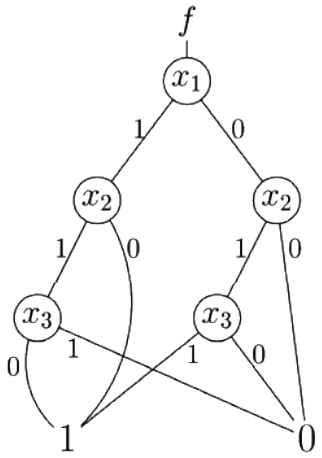

Fig. 3: BDD representation of the Boolean function $f = x_1 \oplus x_2 \cdot x_3$ [1].

$x_4 + \cdots + x_{n-1} \cdot x_n$ as an example with its companion BDD represented in Fig. 4-a that depicts the BDD representation of the function f with a size complexity of $O(n)$. However, Fig. 4-b represents another BDD for the same function f with a different nodes order, in which the size complexity is $O(2^n)$ [40]. This example assures the fact that changing the order of a Boolean function variables that is reflected on changing the position BDD nodes, leads to different realizations of the result reversible circuits with different costs. This makes it worthy to work around searching for the minimal BDD sizes in order to have minimal cost for the synthesized reversible circuits.

3.4. Mathematical Formulation for Reversible Circuit Synthesis Problem

Starting from a Boolean function f, the main goal is to find correct realization for the Boolean function represented by a reversible circuit G with a low cost and within an acceptable synthesis time. Eq.(3) formulates this optimization problem with the existence of the two constraints represented by both the Cost $\zeta(G)$ and synthesis (realization) time τ that is limited with a predefined

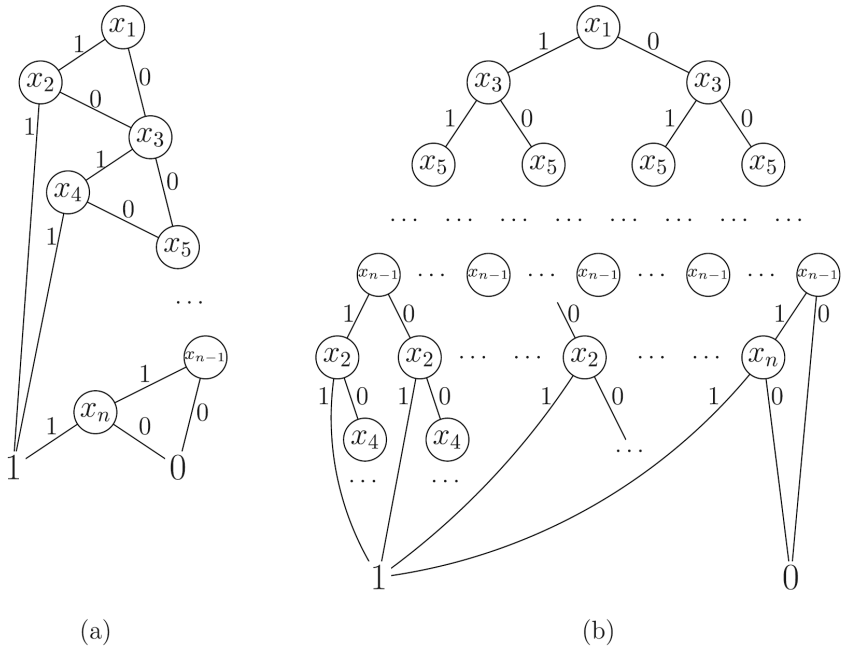

Fig. 4: The boolean function $f = x_1 \cdot x_2 + x_3 \cdot x_4 + \cdots + x_{n-1}.x_n$ is represented by two BDDs with two different node orders: (a) Size complexity of $O(n)$. (b) Size complexity of $O(2^n)$ [40]

upper bound τ_{U} for the Boolean function $f(X)$ where $X = x_1, x_2, \cdots, x_n$.

$$\begin{aligned} &\underset{G}{\text{minimize}}\ \zeta(G) \\ &\textbf{subject to} \\ &G \text{ realizes } f(X) \\ &\tau \leq \tau_{\mathrm{U}} \end{aligned} \tag{3}$$

The cost of a reversible circuit G, denoted by $\zeta(G)$, equals to $QC(G)$ for the circuit in general. However, for reliability analysis and fault-tolerant quantum circuits, $\zeta(G) = TOF\#(G)$, where $TOF\#(G)$ represents the number of Toffoli gates composing circuit G.

Starting from the BDD (that represents a given Boolean function), the realization process of the corresponding reversible circuit is the sequential substi-

tution of each node in the BDD to its mapped reversible Toffoli gates. Due to the direct 1-1 mapping between each node in the BDD and its corresponding cascade of reversible gates, it is possible to evaluate the cost of a circuit in terms of the BDD size as a consequence of: (1) The linear relationship between the BDD size and the number of Toffoli gates (2) A potential for a relationship between the BDD size and the QC of the circuit. While the first reason is intuitive, the second requires investigation. To show this relation, we used a set of benchmarks (Boolean functions) to compute the QC of the synthesized reversible circuits as well as the sizes of the BDDs for these benchmarks. Results of this test are depicted in Fig. 5 which discloses a strong linear and correlated relationship between QC and BDD size with a correlation coefficient equals to 0.985. Due to this, one can depend on the BDD size as an evaluation criterion for the problem of synthesis optimization.

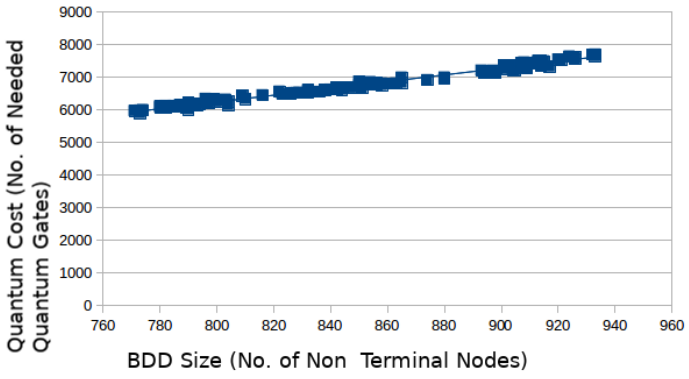

Fig. 5: The linear relation between BDD size and QC for different benchmarks (Revkit simulator [41] is used). Notice that the X-axis represents the number of non-terminal nodes in each BDD while the Y-axis represents the number of elementary quantum gates required to implement each Boolean function represented by the given BDD.

Hence, we can reformulate the problem of reversible synthesis to have a simpler objective function that relies on finding the best order of the input variables $\pi(X)$ of a given Boolean function $f(X)$ that leads to the minimal BDD size after applying the suitable reduction rules instead of computing the

cost $\zeta(G)$. The formulation of the new problem is given by Eq.(4) where the order (permutation) of input variables of the BDD is represented by π and the BDD size related to that permutation is given by $\psi(\pi)$ after suitable reduction rules are applied. Notice the constraint applied to the BDD size that is limited by $\alpha|X|$ where α symbolizes a pre-defined constant. Fig. 4-a depicts a BDD whose order is represented by $\pi(X) = (x_1, x_2, x_3, \ldots, x_n)$ where $\psi(\pi) = n$.

$$\begin{aligned} &\underset{\pi(X)}{\text{minimize}}\ \psi(\pi(X)) \\ &\textbf{subject to} \\ &\psi(\pi(X)) \leq \alpha * |X| \\ &\tau \leq \tau_{\mathrm{U}} \end{aligned} \tag{4}$$

4. Proposed BDD Reordering Optimization Engine

According to the algorithm proposed in [30], each node in the BDD is substituted with a cascade of Toffoli gates following a pre-defined lookup table. The proper substitution for a node is mainly determined by its location and its successor nodes in the graph.

4.1. BDD-based Reversible Logic Synthesis Algorithm

Fig.6 depicts the basic BDD-based synthesis algorithm for reversible circuits. The input for this algorithm is a Boolean function f and the output is a reversible circuit G that realizes function f using a cascade of Toffoli gates. The algorithm consists of the following phases:

1. Initial BDD Construction phase: The input function f is represented by an initial BDD for which shared nodes and complemented edges reduction rules are applied.

2. BDD Reordering Optimization Phase: This phase is the core of our work, wherein, the BDD reordering optimization engine manipulates the BDD iteratively under the guidance of BDD size as an evaluation criterion. This phase will be explained in details subsequently.

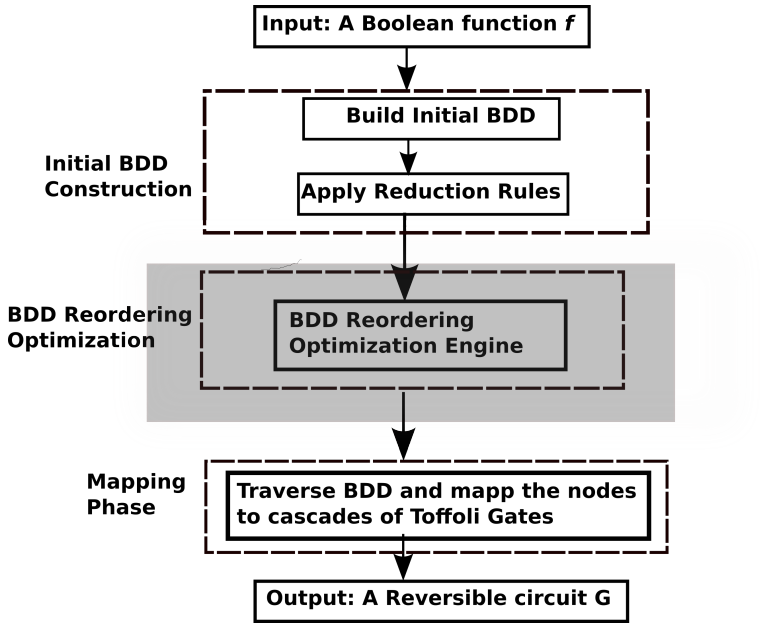

Fig. 6: BDD-based Basic Reversible Circuit Synthesis Algorithm.

3. Mapping phase: Once a small-sized BDD is outputted from the previous phase, it is traversed in Depth-First-Search (DFS) fashion so that each node in the BDD is substituted with the proper cascade of Toffoli gates following the lookup table published in [30].

4.2. Proposed BDD-Reordering Optimization Engine

Our proposed BDD-reordering optimization engine exploits recent swarm optimization algorithms and Genetic Algorithm (GA) supported with recent crossover/mutation operators. However, there are two key differences between both approaches: (1) The swarm algorithms deal with real numbers while the GA deals directly with the permutation as a solution representation. (2) The GA applies crossover/mutation operators on the population and replaces the worst solution in the population iteratively to preserve elitism, whereas, the swarm algorithm guides the next iteration in accordance with the phenomenon it simulates. Because of those differences, we propose two versions of the opti-

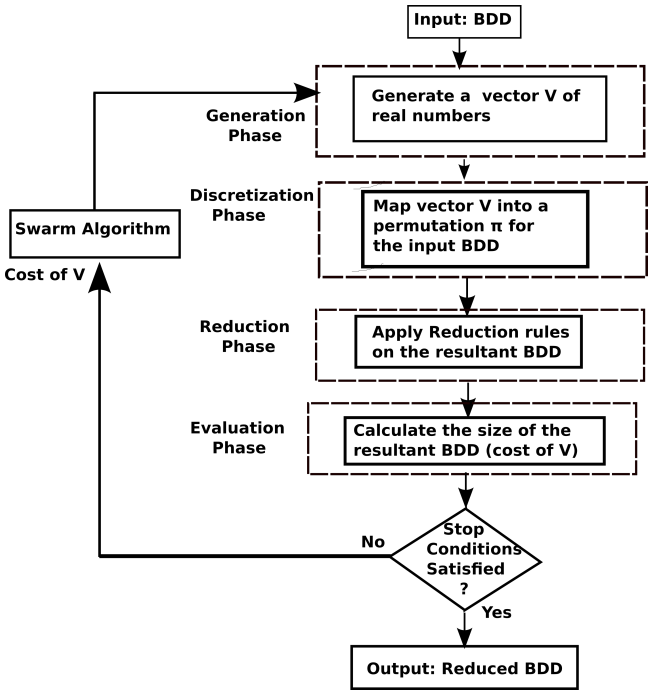

Fig. 7: BDD Size Swarm-based Optimization Engine.

mization engine: swarm-based, and GA-based.

Our proposed Swarm-based BDD reordering optimization engine is illustrated in Fig.7 below, where its input is a BDD and undergoes the following phases:

1. Generation phase: A vector V of real numbers within the interval $[\alpha, \beta]$ is randomly generated.

2. Discretization phase: The generated vector V of real numbers and the positions of those numbers in their ascending order are mapped to generate the order π of the BDD variables. That is, $h : V \rightarrow \pi$ where h represents the mapping function.

3. Reduction phase: BDD reduction rules are applied in this phase (includes shared nodes and complemented edges).

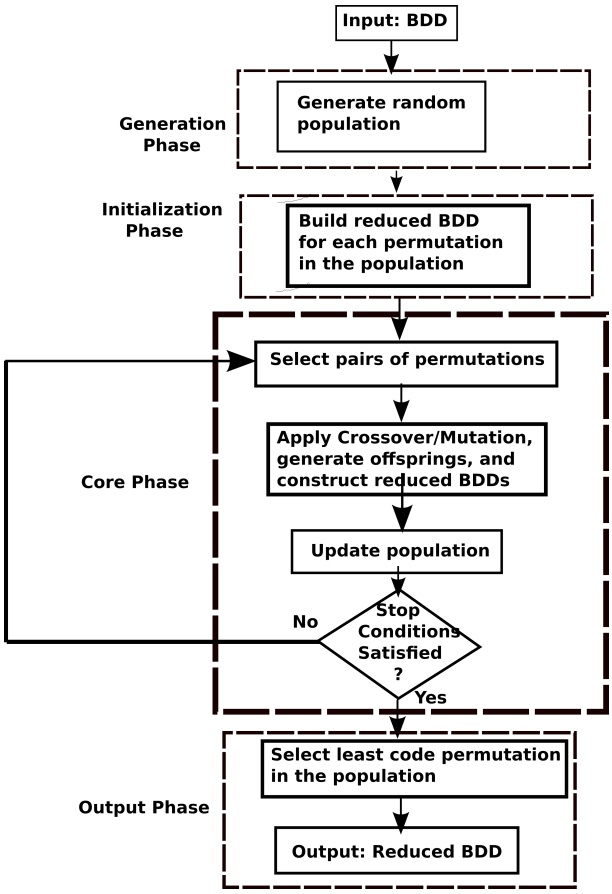

Fig. 8: BDD Size GA-based Optimization Engine.

4. Evaluation phase: After the reduction of the resultant BDD is done, it is evaluated in terms of its size in accordance with the observed linearity between the BDD size and the QC. This size represents the cost of the vector V. If the stop conditions are satisfied, the reduced BDD is outputted. Otherwise, the cost of the vector drives the used swarm optimization algorithm to generate the next vector of real numbers.

Fig.8 illustrates our proposed GA-based BDD-reordering optimization engine which consists of the following phases:

1. Generation phase: An initial random population is generated where each member in the population represents a permutation of the variables in the BDD.

2. Initialization phase: For each solution representation in the population, the BDD is constructed and an initial reordering is applied using sifting algorithm [23].

3. Core phase: In this phase, selection is applied to choose the solutions on which crossover/mutation is applied. Thereafter, each pair of the selected solutions undergoes crossover followed by either another crossover or a mutation. The choice of the second operation is done randomly. The corresponding BDD for each offspring is then constructed, sifted, and reduced using reduction rules. If the cost of this offspring (in terms of its corresponding BDD size) is less than the worst cost in the population, the solution that results in the worst cost in the population is replaced with the recently generated offspring.

4. Output phase: Once the maximum number of iterations has been reached, the permutation that results in the least cost in the population is outputted. Notice that, the maximum number of iterations has been set to be multiples of the number of the variables in the BDD. This aims to avoid being stuck into large number of iterations for small BDDs, whose proper reordering is expected to be obtained in the first few iterations.

4.3. Recent Swarm Optimization Algorithms

The nature is considered the first teacher to the human. It is the inspiring of a lot of ideas that let to important inventions. One of these lessons is to mimic its creatures and their behaviors in their daily lives. One of these inspiration fields is the ones that are related to algorithms development in computer science fields. Several searching and optimization algorithms are named according to their mimic for some related creatures.

Cuckoo Search (CS) is considered one of the most interesting swarm algorithms and their variations. The core of this algorithm is to mimic the Cuckoo bird breeding behavior. A random selection of nests is executed by this algorithm with the evaluation of the worst ones by objective function in order to remove them from the population. To explore the search space, the algorithm uses levy's walk alongside with local walks in order to explore the possible solutions [42]. However, another algorithm works in a similar way to particle swarm algorithm is called Firefly Algorithm (FFA). This algorithm assumes that the less brighter fireflies are attracted to the brighter ones. The objective function in this algorithm is the brightness property itself. A function of distance is dropped exponentially upon the seen of the brightness of one individual firefly [43]. Another promising swarm based algorithm is the Grey Wolf Optimization (GWO) algorithm. In nature, the Grey Wolfs act in a very complex social and hunting behaviors. The algorithm simulates the Grey Wolfs behavior in selecting the best three leader wolfs and makes the others as followers (called omegas). In this context, the omegas always moves to the center of the leaders, so that the leaders always surround omegas. Upon this nature of wolves, the algorithm keeps searching for solutions near to the three main solutions represented by the leaders. In other words, the exploitation part of the algorithm is conducted by divergence of leaders (the current best solutions) to simulate the search for preys (other next better solutions) [44].

From the behavior of moths and how they move in spiral move around the source of light (or fire flames) comes the Moth-Flame Optimization (MFO) algorithm. In this algorithm, the flames are considered the best solution found

and moths are considered the other problem solutions. In order to update the location of the solutions (the moths), the algorithm exploits the spiral movement in order to move to the next found better solution [45].

Bat Algorithm (BAT), Slap Swarm Algorithm (SSA), Whale optimization Algorithm (WOA), and Multiverse Optimization Algorithm)MVO) are all examples of interesting searching algorithms [46] [47] [48] [49]. Each of which mimics the nature of creatures where the algorithm takes the name from.

4.4. Genetic Algorithm

We formulated in Eq.(3) the problem of minimizing BDD size as finding the best order (or permutation) of the variables that construct the BDD. Therefore, the natural way to define chromosome is using permutation. First variables are given an arbitrary order. Each variable is associated with a number from 0 to number of variables minus one. From this setting, the order $\pi_0 = (0, 1, 2, 3, 4, ...n - 1)$ represents the original order of the variables. Any other permutation means changing the order of the variables. For example $\pi_1 = (4, 1, 2, 3, 0, ...n - 1)$ means swapping the first variable with the fifth one. In other words the fifth variable in the original order will be the root of the BDD and the root is occupying the fifth level. Both crossover and mutation operators help in increasing the exploration and exploitation of the search space in the GA-based optimization. However, such operators that preserve the order of some variables in their near-optimal locations during the optimization process, are expected to generate BDDs with low size.

4.4.1. Crossover Operators

Crossover operator might result in a newborn chromosome (permutation of BDD variables) which is better than the parents during the evolution process. The classical crossover GA operator that cuts one part of the chromosome then concatenates it with a part from the other parent chromosome will lead to corrupted not viable offspring. The literature suggests some crossover operators that preserve a permutation characteristics. That means all the numbers from

0 to $n-1$ present once and only once. In this paper, we have chosen four crossover operators to be applied in BDD reordering optimization. This includes: Partially Mapped Crossover (PMX), Ordered Crossover (OX), Cycle Crossover (CX), and Alternating Crossover (AX) [50][51]. The choice for those crossovers is justified by their ability to increase the exploration/exploitation with relatively low destruction in the optimal order of some variables during the optimization process.

Ordered Crossover (OX): Given two permutations for the variables in a BDD, two random cutting points are chosen in those permutations. Those points will partition each permutation into left, middle, and right portions. Each child is then constructed through copying 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 [50]. Fig. 9 illustrates an example of OX.

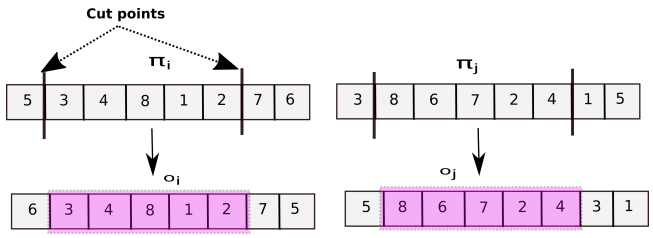

Fig. 9: Ordered crossover where π_i, π_j represent two parent permutations and o_i, o_j represent their corresponding offsprings. Notice that the swash of consecutive variables between the cut points remains the same for each offspring as its corresponding parent.

Cycle Crossover (CX): This crossover identifies a cycle between the two parent permutations. The formed cycle starting from the first parent is used to fill the first offspring while the remaining portions 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 portions [52]. Fig. 10 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.

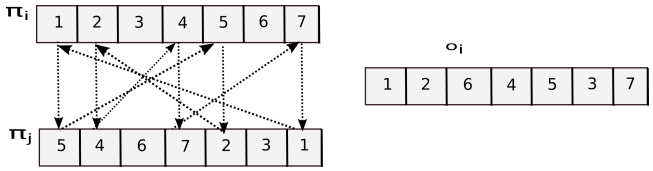

Fig. 10: Cycle crossover where π_i, π_j represent two parent permutations and o_i represents the offspring resultant from a cycle starting with the first part of parent π_i. Notice that the dashed arrows represent how a cycle is constructed.

Partially-Mapped Crossover (PMX): This crossover defines two cut points in the two parents. The portion of variable positions between the two cut points will be swapped between both parents to simultaneously generate portions of the two offsprings. The remaining portions of the offsprings will be determined using the created mapping relation between both parents [52]. Fig. 11 illustrates PMX example with the following mapping relations resultant from the swapping process: $6 \leftrightarrow 3, 7 \leftrightarrow 4, 2 \leftrightarrow 5, 1 \leftrightarrow 6$.

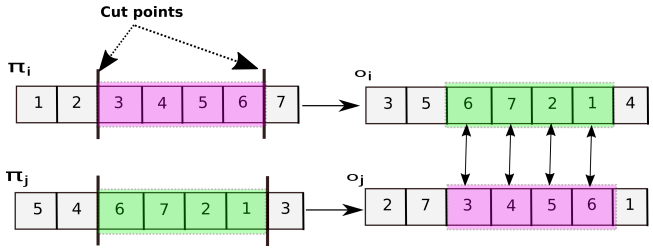

Fig. 11: Partially mapped crossover where π_i, π_j represent two parent permutations and o_i, o_j represents the offsprings.

Alternating Crossover (AX): This crossover simply generates the offspring by alternatively taking a variable position from each parent. That is, a position is taken from one parent followed by a position taken from the other. If a position has already been taken, then it will be ignored and the first not taken position in the other parent is checked [51]. Fig. 12 illustrates an example of

AX.

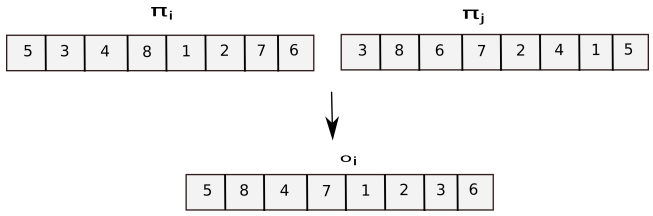

Fig. 12: Alternating crossover where π_i, π_j represent two parent permutations and o_i represents the offspring. Notice that this crossover just alternates between both permutations when generating the offspring with skipping the already recorded positions.

4.4.2. Mutation Operator

The purpose of mutation is to increase the exploration of the GA. This operator is applied to a single permutation to get a new similar one whose BDD might have a smaller size. In this paper, three mutation operators are exploited as follows:

Inversion Mutation: This mutation is employed due 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. 13 illustrates inversion mutation.

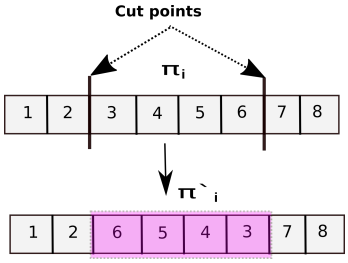

Fig. 13: Inversion mutation where π_i represents a permutation and π'_i represents the new permutation after inversion.

Swap Mutation: In this mutation, two positions are selected randomly in the chromosome representing a solution. Then, simply the selected positions are swapped as illustrated in Fig. 14 [53].

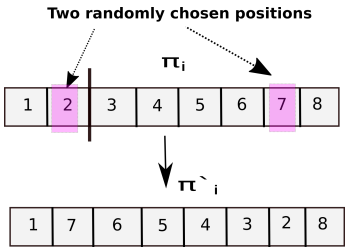

Fig. 14: Swap mutation where π_i represents a permutation and π_i' represents the new permutation after swapping.

Shuffle Mutation: In this mutation, two cut points are chosen randomly in the chromosome representing a solution. Thereafter, the portion of the positions bounded by those cut points are randomly shuffled [54]. Fig.15 illustrates an example of this mutation.

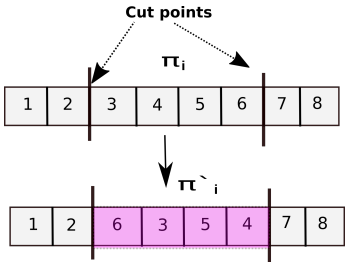

Fig. 15: Shuffle mutation where π_i represents a permutation and π_i' represents the new permutation after randomly shuffling the portion of positions bounded by the two cut points.

5. Experimental Tests

To understand the behavior of the algorithms, we compared several swarm algorithms using six benchmarks of different sizes and complexities side by side with genetic algorithm. It is important to mention that the results we obtained can not be directly compared with meta-heuristic based previous works for several reasons; First, different works use different number of iterations and population sizes. Second, different works use different stopping criteria. For

example, some algorithms use a fixed number of iterations while others keep the progress until reaching a known minimum. Therefore, to have a fair comparison, we have implemented the commonly used algorithms in our own framework and compared our approach to them on the public benchmarks. The classical greedy reordering algorithm, namely sifting (which is implemented in the CUDD library) has been considered as a baseline for our comparison.

As mentioned before, several crossover/mutation combinations were experimented alongside with swarm optimizers. The experiment was performed using python implementation of optimization techniques. We also used python binding to CUDD library that is written in C. CUDD was used to build BDDs [1]. To find the average case of the stochastic algorithms we repeated the same experiments ten times. The following pairs of values represent a function name and its corresponding number of variables where all benmarks are available in RevLib website [2]: (Urf3, 10), (Hwb6, 6), (41084, 14), (Ham15, 15), (Apex4, 19), and (Seq, 41). Those functions have been used in our experiment. The maximum number of iterations is 3 times the number of variables of the synthesized function. Population size (swarm size) was fixed to 20 for all functions.

Table 1 shows the results of steady state GA using crossover and mutations operators. Obviously the BDDs resultant from GA are around 11 times smaller than those obtained from the classical sifting algorithm. Mutation operators dominate the effect on the performance except with AX. The difference in performance between mutation operators is greater than the difference of performance between crossover operators. Figure 16 depicts the box-plot for the "seq" benchmark for different operators. It clearly shows that swap operator outperforms others in the terms of minimum, maximum, and average results. Second important observation is that the less changes in the chromosome (permutation), the better the performance obtained by an operator. Swap performs the best because it is a minimalist operator that only changes the order of two

[1] https://davidkebo.com/cudd
[2] http://www.revlib.org/

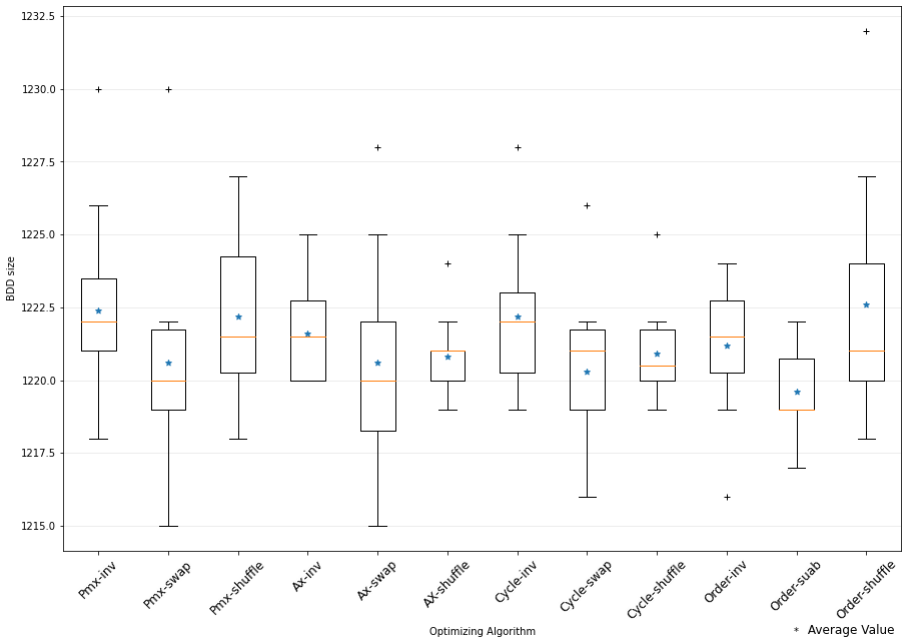

Fig. 16: Boxplot of "seq" benchmark. The distribution of BDD size for different crossover/-mutation operators.

variables. The results show that shuffle destroys part of the chromosome which increments the degree of randomness in the algorithm.

Even though Crossover operators affect the performance less than mutation operators in this experiment they still have an effect. The same conclusion can be observed. Cycle and order operators have a tendency to keep the same variables order as the parents more than PMX. This observation can be seen only with swap mutation. The operator that is less destructive seems to be AX even when used with shuffle mutation it still shows excellence. AX preserves the level of each variable as it appears in one of its parents. Although the relative order of variables affects how the BDD will be reduced also the level of the variable in the graph is very influential on the size of the graph.

As shown in Table 2 and Fig.17, it is obvious that MFO, SSA, WOA and PSO exhibited similar convergence speed. The differences between the four algorithms are minor. They all converge to almost the same solution, although

Table 1: GA performance with different operators in terms of average BDD size

Crossover	**Mutation**	**urf3**	**hwb6**	**410184**	**ham15**	**apex4**	**seq**	**sum**
Sifting		1212.8	77.5	270.3	108.4	1036.0	14247.0	16952.0
Without reordering		21158.0	571.0	5250.0	17283.0	12527.0	142291.0	199080.0
AX	swap	172.4	26.2	38.7	23.0	48.3	1220.6	1529.2
order	Swap	176.7	26.1	39.0	23.0	48.1	1219.6	1532.5
AX	shuffle	174.0	26.1	40.9	23.0	48.2	1220.8	1533.0
cycle	Swap	176.3	26.2	38.8	23.0	48.4	1220.3	1533.0
PMX	Swap	176.7	26.0	39.8	23.0	48.0	1220.6	1534.1
order	Invert	177.5	26.1	40.6	23.0	47.6	1221.2	1536.0
PMX	Invert	176.9	26.4	39.5	23.0	47.9	1222.4	1536.1
AX	invert	174.7	26.2	41.7	23.0	48.9	1221.6	1536.1
cycle	Invert	177.4	26.2	39.4	23.0	48.9	1222.2	1537.1
order	Shuffle	177.4	26.1	40.8	23.0	48.4	1222.6	1538.3
PMX	Shuffle	177.3	26.2	41.9	23.0	48.1	1222.2	1538.7
Cycle	Shuffle	177.7	26.1	42.7	23.0	48.6	1220.9	1539.0

Table 2: Algorithms performance in terms of average BDD size

Algorithm	**urf3**	**hwb6**	**0410184**	**ham15**	**apex4**	**seq**	**Sum**
sifting	1212.8	77.5	270.3	108.4	1036.0	14247.0	16952.0
no reordering	21158.0	571.0	5250.0	17283.0	12527.0	142291.0	199080.0
MFO	178 .1	26.0	36.0	23.0	47.7	1223.6	1534.4
SSA	178 .3	26.0	36.4	23.0	47.7	1223.2	1534.6
WOA	179 .1	26 .2	36.0	23.0	49.1	1227.9	1541.3
PSO	178.6	26.3	37.3	23.0	48.3	1223.6	1537.1
CS	178.3	26.0	45.9	23.0	48.5	1223.6	1545.3
GWO	178.3	26.3	45.5	23.0	48.9	1222.8	1544.8
MVO	179.3	26.2	47.3	23.0	48.4	1222.0	1546.2
FFA	178.8	26.3	50.1	23.0	50.3	1222.3	1550.8
BAT	181.1	26.4	61.2	23.0	51.8	1238.0	1581.5

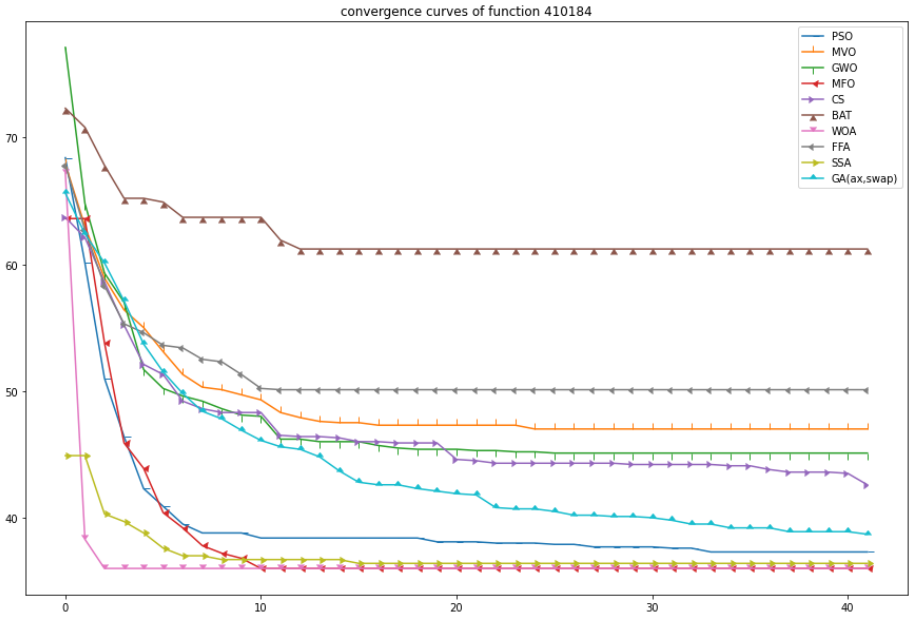

Fig. 17: Convergence Curves for function 0410184 where the X-axis represents iteration# and Y-axis represents the BDD size.

MFO is the fastest and gives the best solution. MFO and WOA include explicit spiral movement while MFO, SSA and WOA include adaptive parameters that are inspired by SA. The parameters control exploration and exploitation throughout the course of search process.The achieved reduction by using MFO over the classical CUDD sifting algorithm is about 11 times smaller in the resultant BDD size as shown in Table 2.

However GA with AX and swap operators outperform all other experimented optimization techniques. This result differs from results in the literature that usually uses PMX operator. For example Stojkovic [51] concluded that mixing PMX with mutation gives best results within the number of iterations and the benchmark functions they used in terms of BDD size (that ranges between 4 and 25). However, the main difference between this work and Stojkovic's work is that we use sifting as local search combined with GA while the work published in [51] does not use any local search.

MFO outperforms all other variations of GA. Comparing PSO and GA variations side by side we can conclude that they are almost the same. Some of GA variations outperform some PSO variations and vice versa. Therefore the main observation is that GA and PSO are comparable. The difference comes from how the algorithm makes the changes to the solutions.

Our experiments consideration in this work goes to the BDD size mainly because of the 1-1 mapping inherent between the BDD nodes and their corresponding reversible circuits. If another mapping scheme is applied, the synthesis time is expected to be dramatically affected. This has been considered in our work published in [55] and [56]. Another metric to be considered is the T-depth of the resultant circuitry. However, this metric is considered in a special set of quantum circuits. Therefore, the BDD size is the metric considered in the comparison conducted in this work.

6. Conclusion and Future Works

In this work we presented swarm based and GA based frameworks to optimize BDD size in order to minimize the cost for reversible circuits. We have compared nine swarm algorithms and six different GA operators to be used in the frameworks. Swarm algorithms are PSO, CS, MFO, SSA, WOA, GWA, FFA, BAT, MVO. GA operators are PMX, CX, AX, OX, swap, shuffle and invert. Experimental results have shown that GA, MFO, SSA, WOA and PSO are potential algorithms to solve this particular problem. GA using a mix of AX and swap operators showed the best performance and fastest convergence behavior. However MFO still can be considered as competitive.

The practical relevance of the obtained results is tangible. To synthesize a reversible circuit for an input Boolean function, the proposed BDD-based synthesis under the guidance of either GA or MFO-swarm optimizer is recommended to be used. GA-based optimizer reinforced with AX and swap operators guarantees outputting reliable reversible circuits with low cost in terms of either the QC or the number of reversible gates composing those circuits. Furthermore, for other applications, wherein BDD is exploited for reliability analysis, the proposed methodology assures reducing the companion BDD size for such applications, and thus, speeding up the reliability estimation. ⇐ 2-2

Although smaller BDD size often results into less quantum cost, this might not be always the case. Therefore, one of the possible future works is to include the mapping phase of the synthesis algorithm into the optimization per se with setting minimizing the number of control lines as the objective function. Another potential extension of this work is to apply the proposed approach with recent meta-heuristic based algorithms on Quantum Multiple-Valued Decision Diagram (QMDD) for Multi-Valued Logic (MVL) to support the existing sifting algorithm with smart exploration of the search space to reduce the total cost of the synthesized MVL circuits.

References

[1] R. Wille, R. Drechsler, Towards a Design Flow for Reversible Logic., Springer, 2010.

[2] M. Soeken, M. Roetteler, N. Wiebe, G. De Micheli, Hierarchical reversible logic synthesis using luts, in: 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), 2017, pp. 1–6.

[3] T. N. Sasamal, A. K. Singh, A. Mohan, Reversible logic circuit synthesis and optimization using adaptive genetic algorithm, Procedia Computer Science 70 (2015) 407 – 413, proceedings of the 4th International Conference on Eco-friendly Computing and Communication Systems. doi:https://doi.org/10.1016/j.procs.2015.10.054.
URL http://www.sciencedirect.com/science/article/pii/S1877050915032184

[4] M. Handique, J. K. Deka, S. Biswas, A fault detection scheme for reversible circuits using -ve control k-cnot based circuit, in: 2020 IEEE REGION 10 CONFERENCE (TENCON), 2020, pp. 1–6. doi:10.1109/TENCON50793.2020.9293770.

[5] M. Perkowski, L. Jozwiak, P. Kerntopf, A. Mishchenko, A. Al-Rabadi, A. Coppola, A. Buller, X. Song, M. Khan, S. Yanushkevich, V. Smerko, M. Chrzanowska-jeske, A general decomposition for reversible logic, Proc. RM (01 2001).

[6] C. Jones, Low-overhead constructions for the fault-tolerant toffoli gate, Physical Review A 87 (2) (Feb 2013). doi:10.1103/physreva.87.022328.
URL http://dx.doi.org/10.1103/PhysRevA.87.022328

[7] H. M. Gaur, A. K. Singh, A. Mohan, M. Fujita, D. K. Pradhan, Design of single bit fault tolerant reversible circuits, IEEE Design Test (2020) 1–1doi:10.1109/MDAT.2020.3006808.

[8] P. Kerntopf, A new heuristic algorithm for reversible logic synthesis, in: Proceedings. 41st Design Automation Conference, 2004., 2004, pp. 834–837. doi:10.1145/996566.996789.

[9] B. K. Abdalhaq, A. Awad, A. Hawash, Reversible logic synthesis using binary decision diagrams with exploiting efficient reordering operators, IEEE Access 8 (2020) 156001–156016.

[10] R. Ray, A. Deka, K. Datta, Exact synthesis of reversible logic circuits using model checking, CoRR abs/1702.07470 (2017). arXiv:1702.07470.
URL http://arxiv.org/abs/1702.07470

[11] D. M. Miller, D. Maslov, G. W. Dueck, A transformation based algorithm for reversible logic synthesis, in: Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451), 2003, pp. 318–323. doi:10.1145/775832.775915.

[12] L. Biswal, R. Das, C. Bandyopadhyay, A. Chattopadhyay, H. Rahaman, A template-based technique for efficient clifford+t-based quantum circuit implementation, Microelectronics Journal 81 (2018) 58 – 68. doi:https://doi.org/10.1016/j.mejo.2018.08.011.
URL http://www.sciencedirect.com/science/article/pii/S0026269218300430

[13] Z. Pan, L. Xing, Y. Mo, A new reliability evaluation method for networks with imperfect vertices using bdd, Quality and Reliability Engineering International 33 (8) (2017) 1957–1967. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/qre.2159, doi:https://doi.org/10.1002/qre.2159.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/qre.2159

[14] V. Sharma, R. Mishra, Reliability analysis of multi-state networks using multi-state binary decision diagrams, in: 2020 IEEE Students Conference on Engineering Systems (SCES), 2020, pp. 1–6. doi:10.1109/SCES50439.2020.9236735.

[15] W. G. Schneeweiss, A short boolean derivation of mean failure frequency for any (also non-coherent) system, Reliability Engineering & System Safety 94 (8) (2009) 1363–1367. doi:https://doi.org/10.1016/j.ress.2008.12.001.
URL https://www.sciencedirect.com/science/article/pii/S0951832008002779

[16] E. Zaitseva, V. Levashenko, J. Kostolny, Importance analysis based on logical differential calculus and binary decision diagram, Reliability Engineering & System Safety 138 (2015) 135–144. doi:https://doi.org/10.1016/j.ress.2015.01.009.
URL https://www.sciencedirect.com/science/article/pii/S0951832015000198

[17] W. G. Schneeweiss, Better graphs for dependability modeling, Microelectronics Reliability 38 (5) (1998) 807–820. doi:https://doi.org/10.1016/S0026-2714(98)00013-4.
URL https://www.sciencedirect.com/science/article/pii/S0026271498000134

[18] F. P. García Márquez, I. Segovia Ramírez, B. Mohammadi-Ivatloo, A. P. Marugán, Reliability dynamic analysis by fault trees and binary decision diagrams, Information 11 (6) (2020). doi:10.3390/info11060324.
URL https://www.mdpi.com/2078-2489/11/6/324

[19] A. Chattopadhyay, A. Littarru, L. Amarú, P. Gaillardon, G. D. Micheli, Reversible logic synthesis via biconditional binary decision diagrams, in: 2015 IEEE International Symposium on Multiple-Valued Logic, 2015, pp. 2–7. doi:10.1109/ISMVL.2015.21.

[20] K. Podlaski, Reversible circuit synthesis using binary decision diagrams, in: 2016 MIXDES - 23rd International Conference Mixed Design of Integrated Circuits and Systems, 2016, pp. 235–238. doi:10.1109/MIXDES.2016.7529738.

[21] B. Bollig, I. Wegener, Improving the variable ordering of obdds is np-complete, IEEE Transactions on Computers 45 (9) (1996) 993–1002. doi:10.1109/12.537122.

[22] D. Grosse, X. Chen, G. W. Dueck, R. Drechsler, Exact sat-based toffoli network synthesis, in: Proceedings of the 17th ACM Great Lakes Symposium on VLSI, GLSVLSI '07, ACM, New York, NY, USA, 2007, pp. 96–101. doi:10.1145/1228784.1228812.
URL http://doi.acm.org/10.1145/1228784.1228812

[23] R. Rudell, Dynamic variable ordering for ordered binary decision diagrams, in: Proceedings of 1993 International Conference on Computer Aided Design (ICCAD), 1993, pp. 42–47. doi:10.1109/ICCAD.1993.580029.

[24] S. J. Friedman, K. J. Supowit, Finding the optimal variable ordering for binary decision diagrams, in: Proceedings of the 24th ACM/IEEE Design Automation Conference, DAC '87, ACM, New York, NY, USA, 1987, pp. 348–356. doi:10.1145/37888.37941.
URL http://doi.acm.org/10.1145/37888.37941

[25] W. Lenders, C. Baier, Genetic algorithms for the variable ordering problem of binary decision diagrams, in: A. H. Wright, M. D. Vose, K. A. De Jong, L. M. Schmitt (Eds.), Foundations of Genetic Algorithms, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 1–20.

[26] B. Abdalhaq, A. Awad, A. Hawash, A swarm based binary decision diagram (bdd) reordering optimizer for reversible circuit synthesis, in: 2020 15th Design Technology of Integrated Systems in Nanoscale Era (DTIS), 2020, pp. 1–6.

[27] P. Manna, D. K. Kole, H. Rahaman, D. K. Das, B. B. Bhattacharya, Reversible logic circuit synthesis using genetic algorithm and particle swarm optimization, in: 2012 International Symposium on Electronic System Design (ISED), 2012, pp. 246–250. doi:10.1109/ISED.2012.71.

[28] K. Podlaski, Reversible circuit synthesis with particle swarm optimization using crossover operator, in: 2015 22nd International Conference Mixed Design of Integrated Circuits Systems (MIXDES), 2015, pp. 375–379. doi:10.1109/MIXDES.2015.7208546.

[29] R. Wille, R. Drechsler, Effect of bdd optimization on synthesis of reversible and quantum logic, Electronic Notes in Theoretical Computer Science 253 (6) (2010) 57 – 70, proceedings of the Workshop on Reversible Computation (RC 2009). doi:https://doi.org/10.1016/j.entcs.2010.02.006.
URL http://www.sciencedirect.com/science/article/pii/S1571066110000198

[30] R. Wille, R. Drechsler, Bdd-based synthesis of reversible logic for large functions, in: Proceedings of the 46th Annual Design Automation Conference, DAC '09, ACM, New

York, NY, USA, 2009, pp. 270–275. doi:10.1145/1629911.1629984.
URL http://doi.acm.org/10.1145/1629911.1629984

[31] B. Bollig, M. Löbbing, I. Wegener, Simulated annealing to improve variable orderings for obdds, in: IN INT'L WORKSHOP ON LOGIC SYNTH, 1995, pp. 5–5.

[32] R. Kaur, M. Bansal, Bdd ordering and minimization using various crossover operators in genetic algorithm, 2014.

[33] I. Furdu, B. Patrut, Genetic algorithm for ordered decision diagrams optimization 16 (2006) 437–444.

[34] M. Hawash, B. Abdalhaq, , A. Hawash, M. Perkowski, Application of genetic algorithm for synthesis of large reversible circuits using covered set partitions, in: The Fourth IEEE International Symposium on Innovation in Information Communication Technology (ISIICT 2011), Vol. 1, 2011.

[35] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95-International Conference on Neural Networks, Vol. 4, IEEE, 1995, pp. 1942–1948.

[36] E. Zaitseva, V. Levashenko, I. Lukyanchuk, J. Rabcan, M. Kvassay, P. Rusnak, Application of generalized reed–muller expression for development of non-binary circuits, Electronics 9 (1) (2020). doi:10.3390/electronics9010012.
URL https://www.mdpi.com/2079-9292/9/1/12

[37] F. Zahoor, F. A. Hussin, F. A. Khanday, M. R. Ahmad, I. Mohd Nawi, C. Y. Ooi, F. Z. Rokhani, Carbon nanotube field effect transistor (cntfet) and resistive random access memory (rram) based ternary combinational logic circuits, Electronics 10 (1) (2021). doi:10.3390/electronics10010079.
URL https://www.mdpi.com/2079-9292/10/1/79

[38] D. Y. Feinstein, M. A. Thornton, D. M. Miller, Minimization of quantum multiple-valued decision diagrams using data structure metrics, J. Multiple Valued Log. Soft Comput. 15 (4) (2009) 361–377.

[39] S. M. Saeed, X. Cui, R. Wille, A. Zulehner, K. Wu, R. Drechsler, R. Karri, Towards reverse engineering reversible logic, CoRR abs/1704.08397 (2017). arXiv:1704.08397.
URL http://arxiv.org/abs/1704.08397

[40] R. Drechsler, N. Drechsler, W. Günther, Fast exact minimization of bdds, in: Proceedings of the 35th Annual Design Automation Conference, DAC '98, ACM, New York, NY, USA, 1998, pp. 200–205. doi:10.1145/277044.277099.
URL http://doi.acm.org/10.1145/277044.277099

[41] M. Soeken, S. Frehse, R. Wille, R. Drechsler, Revkit: An open source toolkit for the design of reversible circuits, in: A. De Vos, R. Wille (Eds.), Reversible Computation, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 64–76.

[42] X. Yang, Suash Deb, Cuckoo search via lévy flights, in: 2009 World Congress on Nature Biologically Inspired Computing (NaBIC), 2009, pp. 210–214. doi:10.1109/NABIC.2009.5393690.

[43] X.-S. Yang, Firefly algorithm, stochastic test functions and design optimisation, IJBIC 2 (2010) 78–84.

[44] S. Mirjalili, S. M. Mirjalili, A. Lewis, Grey wolf optimizer, Advances in engineering software 69 (2014) 46–61.

[45] S. Mirjalili, Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm, Knowledge-based systems 89 (2015) 228–249.

[46] X.-S. Yang, A new metaheuristic bat-inspired algorithm, in: Nature inspired cooperative strategies for optimization (NICSO 2010), Springer, 2010, pp. 65–74.

[47] S. Mirjalili, A. H. Gandomi, S. Z. Mirjalili, S. Saremi, H. Faris, S. M. Mirjalili, Salp swarm algorithm: A bio-inspired optimizer for engineering design problems, Advances in Engineering Software 114 (2017) 163 – 191. doi:https://doi.org/10.1016/j.advengsoft.2017.07.002.
URL http://www.sciencedirect.com/science/article/pii/S0965997816307736

[48] S. Mirjalili, A. Lewis, The whale optimization algorithm, Advances in Engineering Software 95 (2016) 51–67. doi:10.1016/j.advengsoft.2016.01.008.

[49] D. E. Goldberg, J. H. Holland, Genetic algorithms and machine learning (1988).

[50] O. Abdoun, J. Abouchabaka, A comparative study of adaptive crossover operators for genetic algorithms to resolve the traveling salesman problem, CoRR abs/1203.3097 (2012). arXiv:1203.3097.
URL http://arxiv.org/abs/1203.3097

[51] S. Stojković, D. Veličković, D. Veličković, C. Moraga, Genetic algorithm for binary and functional decision diagrams optimization, Facta universitatis - series: Electronics and Energetics 31 (9) (2018) 169–187. doi:10.2298/FUEE1802169S.

[52] N. Kumar, Karambir, R. Kumar, A comparative analysis of pmx, cx, ox crossover operators for solving traveling salesman problem, International Journal of Latest Research in Science and Technology (2012) 98–101.

[53] M.-W. Tsai, T.-P. Hong, W.-T. Lin, A two-dimensional genetic algorithm and its application to aircraft scheduling problem, Mathematical Problems in Engineering 2015 (2015) 1–12. doi:10.1155/2015/906305.

[54] A. Otman, J. Abouchabaka, C. Tajani, Analyzing the performance of mutation operators to solve the travelling salesman problem, Int. J. Emerg. Sci. 2 (03 2012).

[55] A. Hawash, A. Awad, B. Abdalhaq, Towards reducing reversible circuit synthesis time, in: 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), 2019, pp. 538–543. doi:10.1109/JEEIT.2019.8717492.

[56] A. Hawash, A. Awad, B. Abdalhaq, Reversible circuit synthesis time reduction based on subtree-circuit mapping, Applied Sciences 10 (12) (2020). doi:10.3390/app10124147. URL https://www.mdpi.com/2076-3417/10/12/4147