Design and synthesis of improved reversible circuits using AIG- and MIG-based graph data structures

ISSN 1751-8601
Received on 10th May 2017
Revised 16th January 2018
Accepted on 23rd January 2018
E-First on 27th September 2018
doi: 10.1049/iet-cdt.2017.0097
www.ietdl.org

Chandan Bandyopadhyay[1] ✉, *Rakesh Das*[1], *Anupam Chattopadhyay*[2], *Hafizur Rahaman*[1]
[1]*Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur, Howrah 711 103, West Bengal, India*
[2]*Computer Science Engineering, Nanyang Technological University, Singapore, 50 Nanyang Avenue, Singapore 639798, Singapore*
✉ *E-mail: chandanb@it.iiests.ac.in*

Abstract: Reversible logic synthesis is one of the best suited ways which act as the intermediate step for synthesising Boolean functions on quantum technologies. For a given Boolean function, there are multiple possible intermediate representations (IRs), based on functional abstraction, e.g. truth table, decision diagrams or circuit abstraction, e.g. binary decision diagram (BDD), and-inverter graph (AIG) and majority inverter graph (MIG). These IRs play an important role in building circuits as the choice of an IR directly impacts on cost parameters of the design. In the authors' work, they are analysing the effects of different graph-based IRs (BDD, AIG and MIG) and their usability in making efficient circuit realisations. Although applications of BDDs as an IR to represent large functions has already been studied, here they are demonstrating a synthesis scheme by taking AIG and MIG as IRs and making a comprehensive comparative analysis over all these three graph-based IRs. In experimental evaluation, it is being observed that for small functions BDD gives more compact circuits than the other two IRs but when the input size increases, then MIG as IR makes substantial improvements in cost parameters as compared with BDD by reducing quantum cost by 39% on an average. Along with the experimental results, a detailed analysis over the different IRs is also included to find their easiness in designing circuits.

1 Introduction

Continuous downscaling of technological features is not alluring anymore due to multiple obstacles such as poor yield, high thermal footprint, low reliability and in general is giving diminishing returns. As a result, the quest for new computing and storage media has been one of the most prominent research drivers in recent times. Reversible logic synthesis has emerged as a major research discipline, which promises asymptotic zero power dissipation [1]. More importantly, quantum technologies enforce physical reversibility via unitary transformations in the Bloch sphere, which, in turn, mandates logical reversibility. Several quantum technologies such as Ion Trap [2], nuclear magnetic resonance [3] and superconducting qubits [4] have demonstrated the practical realisation of reversible logic gates such as Fredkin and Toffoli gates. In a recent breakthrough, two-qubit quantum gates [5] have been realised in silicon [6], thus paving the way for scalable manufacturing of reversible logic gates in quantum technologies.

1.1 Preliminaries

Much akin to the design of conventional logic circuits, where Boolean logic gates are used to design circuits, quantum circuits are built with reversible logic gate libraries. Several such libraries are shown to be universal, i.e. can be composed to create any arbitrary Boolean function. However, the Boolean function itself may not be fully reversible, and therefore requires a pre-processing step from irreversible to reversible conversion. We proceed with a few definitions and examples to familiarise the reader with the reversible logic synthesis flows.

Definition 1: Let $\mathbb{B} = \{0, 1\}$ be a *Boolean set*, then $\beta_{n,m} = \{f: \mathbb{B}^n \to \mathbb{B}^m\}$ is the set of 2^{m2^n} *Boolean multiple-output function* with n inputs and m outputs.

Definition 2: A function $f \in \beta_{n,m}$ can be called *reversible* if there exists a bijective mapping of the inputs (n) to the outputs (m) and also the number of outputs is equal with number of inputs.

Definition 3: A reversible circuit of size n with depth d is a cascade of d reversible gates over the set of input lines $X = \{x_1, x_2, \dots, x_n\}$, i.e. a reversible circuit is denoted as $g_0, g_1, g_2, \dots, g_{d-1}$, where each g_i represents the ith reversible gate of the circuit.

Example 1: In Fig. 1*a*, a two-control reversible gate is shown, where the gate has three input lines x_1, x_2 and x_3 among which the first two input lines act as control line and the last one is the target line. The gate also has three functional outputs y_1, y_2 and y_3 among which the first two functional outputs are called garbage outputs and the intended output is obtained from the third functional output y_3. The first two functional outputs are the same as the first two input lines and this very nature is used to make the gate functionality reversible.

This shown gate is termed as two-controlled CNOT gate or Toffoli gate, which implements the function $y_3 = x_1.x_2 \bigoplus x_3$.

The idea of Toffoli gate can be extended with further control lines, leading to the Toffoli family of reversible gates. Other prominent reversible gates are Fredkin, Peres and NOT gates. There exists a larger spectrum of reversible gates, which are non-classical, i.e. application of these gates results in an output that is not in the classical domain. Examples of such gates are control-V, control-V† and phase rotation gates. A reversible circuit is designed by a cascade of reversible gates operating on the input qubits, as shown in Fig. 1*b*. The quality of a reversible circuit is determined by cost parameters such as T-count, T-depth, quantum cost (QC) and number of ancilla lines.

Definition 4: The QC (Q_C) of a gate (g) is the number of elementary quantum operations performed to design that specified gate (g).

For large and complex Boolean functions, to determine the most efficient reversible circuit, as per the above parameters, is a non-trivial task. To that end, researchers developed a wide range of reversible logic synthesis techniques. These techniques can be categorised into several classes based on the type of algorithm deployed (e.g. optimal and heuristic) or the type of intermediate representation (IR) (e.g. truth table and decision diagram).

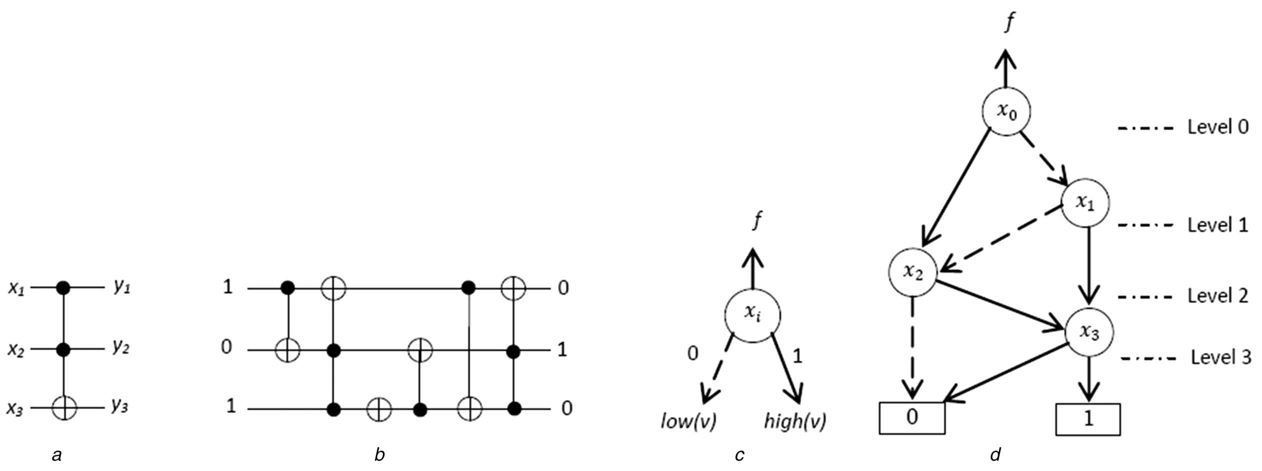


Fig. 1 *Example of reversible gate and circuit*
(a) Three-qubit reversible gate, ***(b)*** Reversible circuit over three qubit lines, ***(c)*** BDD node, ***(d)*** BDD representation corresponding to input function $f = \overline{x_0}\, x_1 x_3 + \overline{x_0}\,\overline{x_1}\, x_2 x_3 + x_0 x_2 x_3$

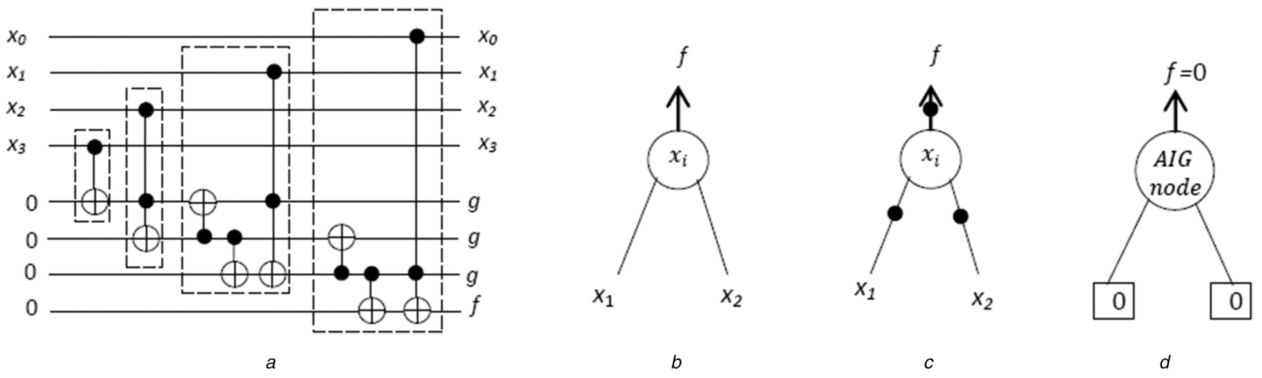


Fig. 2 *Designing reversible circuit from BDD*
(a) Obtained reversible circuit from BDD graph of Fig. 1*d*, ***(b)*** AIG for function $f = x_1.x_2$, ***(c)*** AIG for function $f = x_1 + x_2$, ***(d)*** $f = 0$ When constant inputs $x_1 = 0$ and $x_2 = 0$

Optimal methods: Scalability of such methods [7, 8] is restricted up to six-variable functions as and when they are scaled beyond the said limit, time and space complexities [9] increase rapidly.

Heuristic methods: Methods such as genetic algorithms, *A** [10], ant colony optimisation [11], simulated annealing [12], exclusive sum-of-product (ESOP) [13] are the examples of heuristic approaches.

An alternative classification of the synthesis techniques is possible by checking the intermediate data structure that is used for manipulation and optimisation of the reversible circuit. In [14], two classes of IRs are suggested, namely functional representation and structural representation. In a functional representation, the truth values of the Boolean functions are known at any phase, whereas in a structural representation the truth values are implicit.

It is observed in [14] that structural representations are more scalable, which is also the reason why such representations are the de-facto standard in classical logic synthesis. And-inverter graph (AIG) and majority inverter graph (MIG) are examples of such structural representations. On the other hand, the binary decision diagram (BDD) is a functional representation, which does not scale but is imperative for efficient conversion of irreversible to reversible Boolean function. Therefore, Soeken and Chattopadhyay *et al.* [14] proposed a hybrid logic synthesis flow that combines both the structural and functional representations.

In the following, we briefly outline these IRs. Before stating on a different type of IRs, here we are giving some preliminaries and notations that we have used in our later sections and also in describing the different IRs.

Definition 5: A directed graph with *M* number of primary inputs (PIs) over *V* number of nodes and *E* number of edges with *N* outputs can be expressed in the form *G (M, N, V, E)*, where the number of edges that is incident on a node v_i is called the in-degree $d_{\text{in}}(v_i)$, whereas the number of outgoing edges from that node v_i is known as out-degree $d_{\text{out}}(v_i)$ of that node v_i.

The relation between graph nodes to reversible gates can be found as follows.

Definition 6: A node v_i in a graph *G* (*M, N, V, E*) having in-degree M_i $\left(\text{where } M = \sum_{i=1}^{N} M_i\right)$ can be mapped to a Toffoli gate $\text{TOF}_g\{m[\text{input}(\text{in}_i \in \boldsymbol{M})],\ \perp\}$ with constant input $\perp \in \{\mathbf{0,1}\}$ if it satisfies the function $\mathbb{B}^{d_{\text{in}}(v_i)} \to \mathbb{B}^1$, where the function $m[\text{input}(\text{in}_i \in \boldsymbol{M})]$ denotes the number of control lines present in that Toffoli gate.

Now, we briefly state the different types of the graph-based IRs.

BDDs: Each BDD is expressed using a set of non-terminal vertices *N*, some terminal vertices $T \in \{0, 1\}$ and directed edges *E* which collectively implements a function *f*. Each non-terminal vertex $v \in N$ (labelled as x_i in the graph) has exactly two children which are termed as low*(v)* and high*(v)* and the directed edges to these children having dashed line are called low edge and edges with a solid line are called high edge. The structure of a BDD node is shown in Fig. 1*c*. Using the Shannon decomposition [15], a relation between non-terminal vertex *v* and the function *f(x)* can be obtained as $f(x) = \overline{x_i}\,.\,\text{low}(v) + x_i\,.\ \text{high}(v)$.

Example 2: Here, we see how BDD is used to build the reversible circuit. Let a function be $f = \overline{x_0}\ x_1\ x_3 + x_2 x_3$ and after transforming the function into a decision-based tree form, the equivalent BDD graph is obtained in Fig. 1*d*. Now, taking this form as IR and after invoking an appropriate mapping scheme a reversible circuit is obtained in Fig. 2*a*.

BDDs can be optimised further to form more efficient designs and such a form is named as *reduce ordered BDD*, which can give highest level of optimisations. Again, in spite of considering single

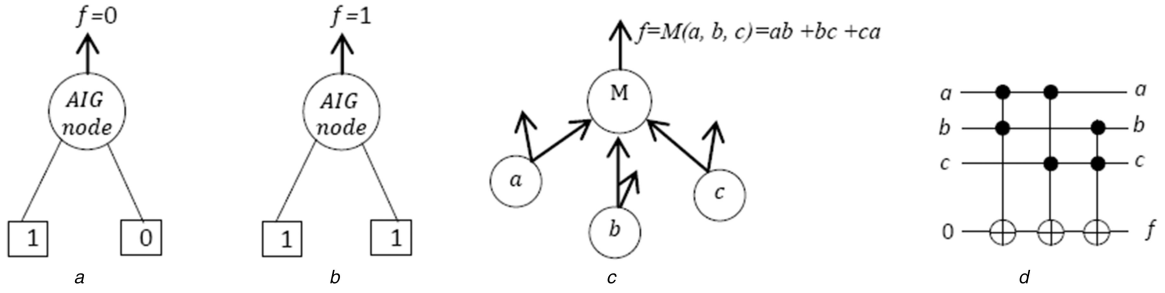


Fig. 3 *Different structural forms of AIG node*
(a) $f = 0$ When constant inputs $x_1 = 1$ and $x_2 = 0$, ***(b)*** $f = 1$ Constant inputs when $x_1 = 1$ and $x_2 = 1$, ***(c)*** Majority node, ***(d)*** Equivalent Toffoli cascades for MIG node

BDDs for each of outputs, if multiple BDDs are combined then a shared BDD can be formed.

AIG: AIG [16] is a type of directed acyclic graph which can be used as an IR in designing circuits from functions. In an AIG, each node may have either 0 or 2 incoming edges and such edges may be in the complemented form or in uncomplemented form. If two uncomplemented edges incident on a node, then AND operation is performed (implements function $f = x_1.x_2$) at the outgoing edge of that node (see Fig. 2*b*). However, if both the incoming edges, as well as the outgoing edge, are complemented (signals are inverted), then logical OR (implements function $f = x_1 + x_2$) operation is performed (see Fig. 2*c*).

The basic AIG node configurations with respect to constant nodes and PIs are also given in Fig. 2*d* and Figs. 3*a* and *b*.

Some nodes in AIG may not contain incoming edges and such nodes are called PIs. Again, the nodes whose output edges never enter into any future nodes are termed as primary outputs; e.g. root node.

MIG: MIG is a superset of AIG since a majority operator can be rendered as an AND (or an OR) operator. A structural diagram of an MIG node is given in Fig. 3*c*, where it can be seen that the MIG contains three majority nodes (a, b, c) which collectively implement the function $f = ab + bc + ca$. Such as operations of AIG, in an MIG node if an input let c is set to 0 (no signal is applied), then logical AND operation is performed at output edge. Again if $c = 1$, then logical OR operation is performed.

1.2 Related works

In the reversible logic synthesis, there have been numerous studies using BDD and AIG [17] as the IR of choice. Although MIG-based reversible logic synthesis has not been reported, a preliminary mapping of MIG nodes to reversible logic gates is recently proposed [18].

Use of BDD as IRs in synthesising reversible circuit is stated in [19], where in the initial stage, the BDD graphs corresponding to input functions are generated and then a template library comprising with reversible circuits is formed to perform technology mapping by replacing the nodes of the graphs with suitable reversible templates. However, this approach results in a high number of garbage lines. To alleviate this problem, a new design strategy is being incorporated in [20], where ancilla-free reversible logic synthesis is done using BDDs. In a way to make the BDD representation more compact, an approach with bi-conditional BDDs is introduced in [21], where better performances for certain Boolean functions are reported.

Applications of higher-order functional programmes in realising reversible circuit have been stated in [17], where regular Boolean programmes have been transformed into a circuit-generating code. In later time, a novel synthesis scheme in realising arithmetic circuits using a variety of classical design automation flows has been investigated in [22]. Extending the work further, a more advanced data structure exclusive OR-majority graph (XMG) is shown in [23], where look-up table (LUT) concept has been introduced to get the best possible results. To make a fine trade of in-between scalability and performance, a hybrid synthesis scheme has been introduced in [14] that runs a Decision Diagram (DD)-based synthesis process under an AIG-based wrapper and generates compact functional representations.

1.3 Contribution

In a way to find out the effectiveness of different graph-based representations in the realisation of reversible circuits, in the present work we undertake a comprehensive analysis and make a detailed benchmarking on three prominent IRs: BDD, AIG and MIG and compare their respective performances. The main objective of our work is to help a designer to select an appropriate IR depending on the need. Getting the idea from [17], here we have extended the AIG work and applied it for the multi-node platform. In our AIG mapping process, we have shown simple AIG templates and have mapped them to functions for getting equivalent reversible realisations. Furthermore, we also have made the technology mapping between MIG to the reversible circuit and have tested it over a wide spectrum of benchmarks. A behavioural analysis of the mentioned IRs over their scalability has also been undertaken.

The rest of this paper is organised as follows. The proposed methodology is discussed in Section 2. Illustration on the presented technique with examples is given in Section 3. Experimental results followed by theoretical analysis over the synthesis process are stated in Sections 4 and 5, respectively. The final concluding remarks appear in Section 6.

2 Proposed technique

Still, so far, we have learned that different IRs lead to different representations of circuits which finally result in variation in cost parameters in the design. How BDDs can be used as IR to get efficient reversible circuits is already an existing work [19] and now we are studying the usefulness of the rest two IRs (MIG and AIG) to build reversible circuits. Along with that we are also making a comparative analysis of the three data structures BDD, AIG and MIG to find out which among these three IRs gives more efficient representation in designing reversible circuits.

This section is divided into two parts: in the first part, we state about the AIG template library and related mapping schemes to obtain reversible designs from AIGs. In the second part, we show how MIGs also can be used as IR in building efficient reversible circuits. Illustrations on the stated IRs and their corresponding mapping processes with examples are presented thereafter.

2.1 Synthesis approach using AIG technique

The process of designing reversible circuits from AIGs involves two phases. In the first phase, AIG graphs are generated from benchmark specifications using some predefined tools. Then, a mapping scheme runs and transforms the graph into an equivalent circuital form. While performing this technological mapping, template libraries are formed that helps to replace graph nodes with sub-circuits, and in a way to design better circuits the formation of optimised template libraries and development of efficient graph traversing algorithms also play a very important role here. In our developed scheme, we also have taken care of these two issues. Now, we explain the entire procedure in detail.

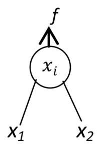
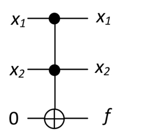
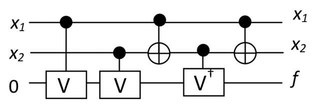
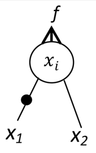
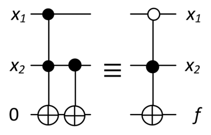
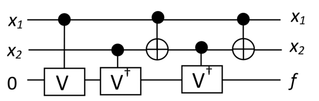
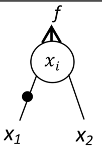
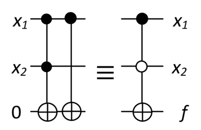
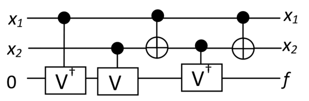
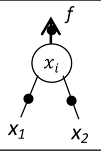
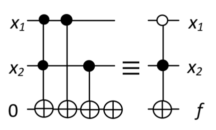
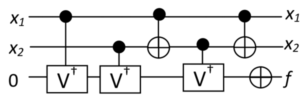
Template IDs	AIG node structures	Toffoli Templates	Optimized NCV realization	T_d	T_c	Q_c
T-1				7	3	5
T-2				7	3	5
T-3				7	3	5
T-4				9	3	6

Fig. 4 *AIG template library*

Generation of AIG: After taking a benchmark file as input, it goes through a set of predefined AIG node structures. Then, the matched sub-functions are replaced with AIG nodes and this process continues until the entire function converts into a graphical form. Finally, AIG is formed by cascading all these nodes in a single graph.

Mapping of AIGs to reversible circuits: This process involves two stages: first the formation of template library and then the construction of AIG equivalent design from the obtained graph by the help of template library.

Stage 1: We have seen that there may be four variations in the node configurations of AIG, and depending on these four different structures initially we form the template library comprising with Toffoli templates. After that, to further optimise the templates, we find its NOT-CNOT-V/V+ (NCV) realisations and make changes in the designs to keep the cost metrics (Q_c, T_d and T_c) at a minimum. Different AIG node structures and their corresponding templates with incurred cost values are shown in Fig. 4.

Stage 2: Now, traverse the AIG in a bottom-up manner while scanning each level from left to right. The scanning algorithm reads each of the nodes and fetches a corresponding Toffoli circuit from template library by finding a match with the selected graph node to the template library node.

If a node is shared by more than one predecessor nodes (this event mostly happens in multiple-output functions), then our algorithm finds such structures and shares the sub-circuits with its predecessors. Once we reach the root node, the algorithm terminates and after cascading all the Toffoli templates a final circuit is formed.

2.2 Synthesis approach using MIG technique

As with AIG mapping technique, this synthesis process also has two stages: (i) generation of MIG graph and then (ii) mapping of MIG graph to the reversible circuit.

Generation of MIG: Initially, an input function is parsed through a mapping scheme, where each sub-module of that input specification file is mapped to equivalent MIG node functions. Once this mapping completes, all the nodes are connected in a systematic way to form a final MIG.

Formation of the reversible circuit from MIG: As we have stated earlier that MIG is the super class of AIG, so, all the templates of AIG are included in MIG template library. In addition to that one more template (T-5) is also included in that library. The design of the new template (T-5) and its internal quantum domain realisation are shown in Fig. 3*d* and 5*a*, respectively.

The important observation related to this template (T-5) is that if any of the edges have an inverter or signal inversions take place in that edge, then corresponding control line in the MIG template changes accordingly, i.e. all the control nodes existing on that line will be changed to negative controls.

As the mapping processes of both AIGs and MIGs to reversible circuits are same, here we are skipping the details and have given a generic algorithm (Algorithm 1) that shows the formation of reversible circuits from either of these IRs.

For an ease of understanding, here we are giving few examples and the detailing related observations developed in the generated circuits.

3 Illustrations on different IRs using examples

Still, so far, we have seen the mapping processes for AIGs and MIGs. However, there remain several design issues related to the building of reversible circuits from these different IRs, and especially here in our examples we have tried to include those areas and also have stated related observations found from the instances. The given examples mainly cover the following three goals:

(i) It states how our mapping algorithms make efficient representations when shared nodes are found in AIGs and MIGs.
(ii) The way of performing '*Bennett embedding*' [1] in our IR generated circuits.
(iii) The decision regarding invoking of which of the IRs in what type of functions can be allowed for designing most cost-effective circuits is also included in the examples and their respective findings.

Algorithm 1: Mapping from IRs to Reversible Circuit

```
Input: A Graph G (M,N,T,⊥) containing MIG/AIG nodes
Output: Circuit composed of reversible gates
1 function: IR_to_Circuit (G)
2 set C ← empty circuit;
3 set P ← 1;
4 initialize stack A;
```

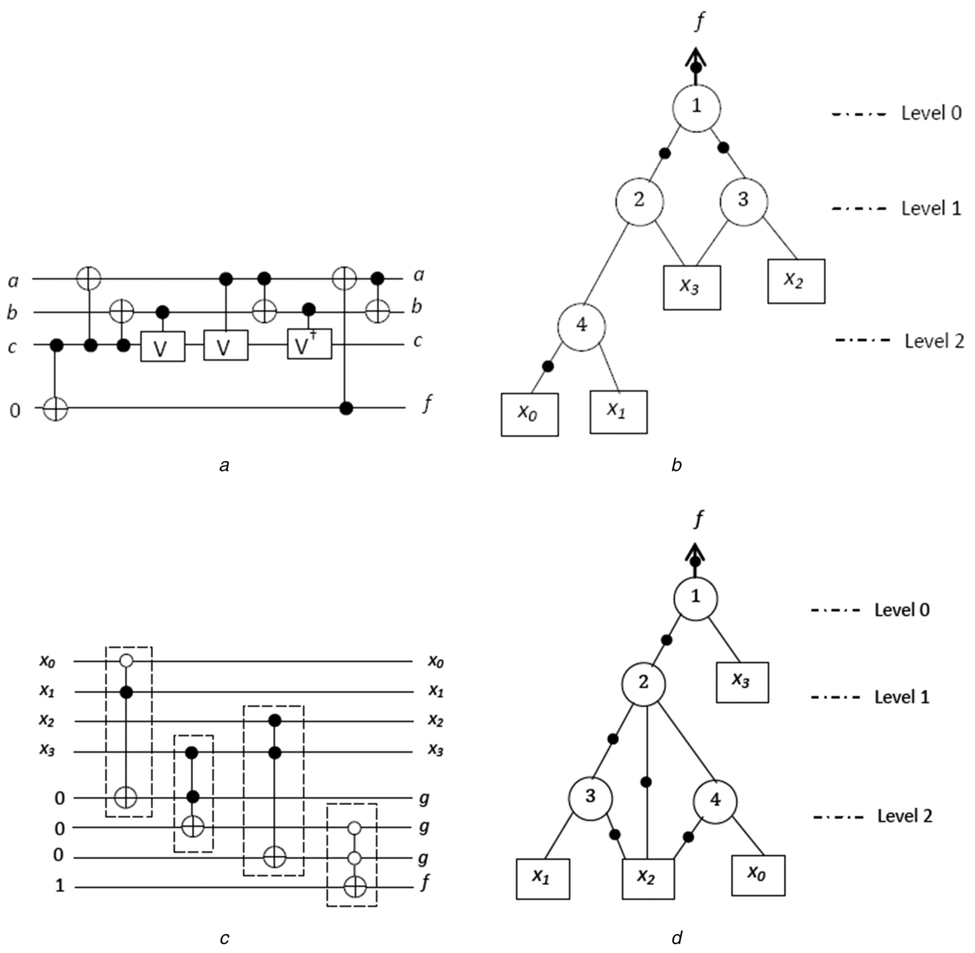


Fig. 5 *Toffoli template (T-5), corresponding to MIG node of Fig. 3c*
(a) Optimised NCV realisation of Fig. 3*d* incurring total $Q_C = 9$, $T_d = 3$ and $T_c = 7$, ***(b)*** AIG for function $f = \overline{x_0}\, x_1\, x_3 + x_2\, x_3$. *Designing reversible circuit from AIG* ***(c)*** AIG equivalent circuit of Fig. 5*b*, ***(d)*** MIG for function $f = \overline{x_0}\, x_1\, x_3 + x_2\, x_3$

```
5 initialize hash_map m;
6 set S ← ∅;
7 set F ← {output(o_i) | o_i ∈ N};

8    for each node v_i ∈ G do
9        –add input line with name v_i to C;
10       perform template matching for that node
11       set m(o_i) ← ⊥ [∀ output(o_i) : some constant ⊥, ∃ ⊥ ∈ {0, 1}];
12     set P ← P + 1;
13 end

14 for g ∈ reverse_order() do
15 set ⊥← get_constant();
16 append TOF_g(m(input(in_i ∈ M)), ⊥) to C;
17 set m(g) ← t;
18       if r(g) = 0 then
19           set S ← ∅;
20           uncompute_intermediate_result(g);
21       end
22 end

23 for each output o_i ∈ N do
24       rename output of line m(output(o_i)) in C to o_i;
25 end
26 return C;

27 function get_constant()
28       if A is not empty then
29           return A.pop();
30       else
31           set P ← P + 1;
32       return P;
33 end
39 function uncompute_intermediate_result(g)
40 if g ∈ S then return;
41 if g ∉ F then
42 set t ← m(g);
43 append TOF(g)(m(input(in_i ∈ M)), ⊥) to C;
44 C.push(t);
45 set m(t) ← 0;
46 set S ← S ∪ {g};
47 end
```

Example 3: Let, the input function be $f = \overline{x_0}x_1x_3 + x_2x_3$ and we will find its AIG equivalent circuital representation.

First, we obtain the AIG form (in Fig. 5*b*) for the function f. Now, evoke the function IR_to_Circuit() in Algorithm 1. By taking the AIG graph as input, the said process starts traversing the tree in a bottom-up manner while scanning each level of the graph from left to right direction. As this algorithm contains the AIG template library, while reading each node structure from that graph, the algorithm tries to find the nodes that have the structure match with its own template structures. As a result, it finds that the node 4 in Fig. 5*b* has a match with template T-2, whereas nodes 3, 2 and 1 find a structure resemblance with templates T-1, T-1 and T-4, respectively.

As the AIG graph in Fig. 5*b* does not pose any shared nodes in its structure, we simply replace each node in that graph G with a template, and finally after cascading all the templates over the

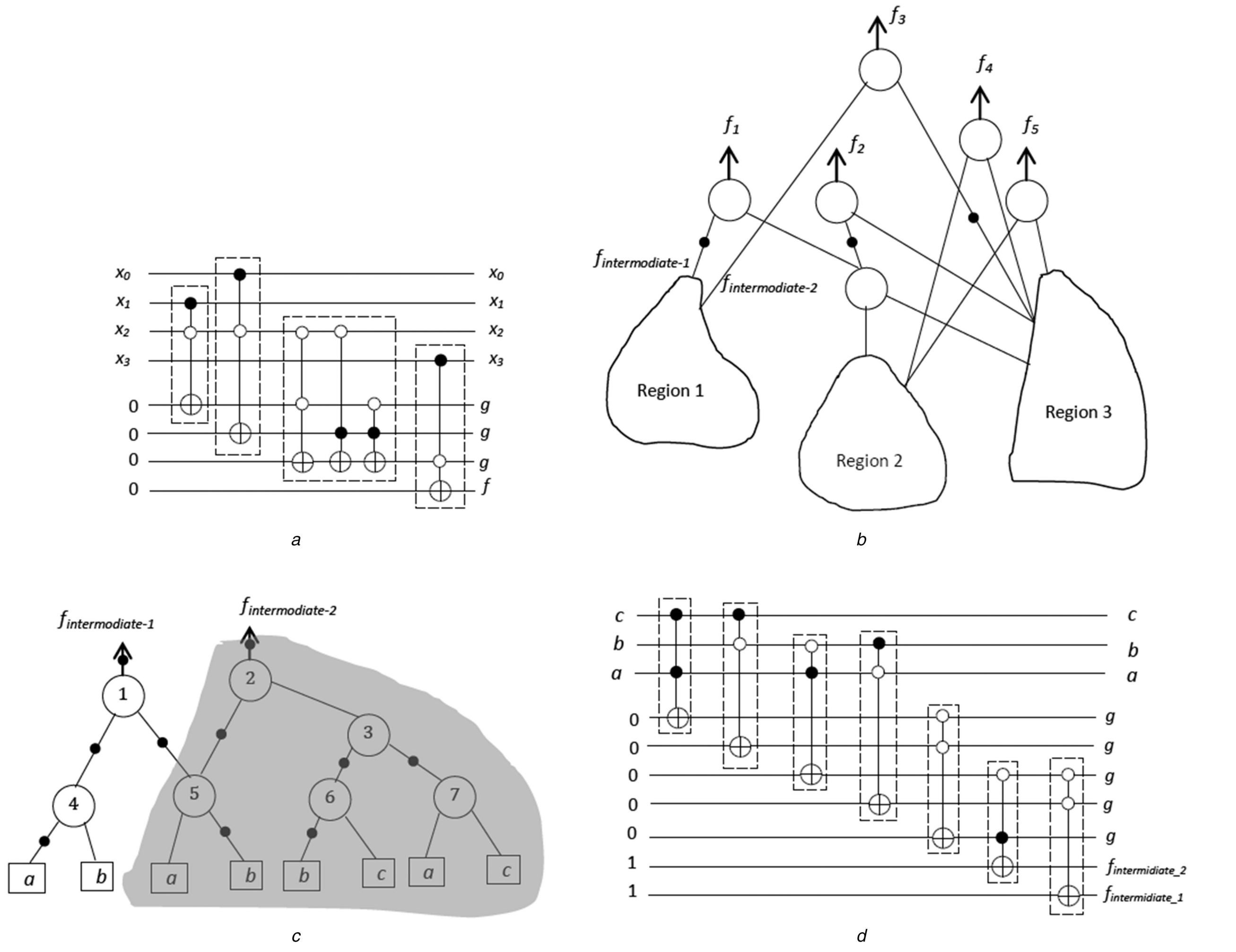


Fig. 6 *Designing reversible circuit from MIG*
(a) MIG equivalent circuit of Fig. 5*d*, ***(b)*** AIG for a five-variable function. Finding reversible circuit by taking AIG as IR, ***(c)*** AIG graph present in region 1 of Fig. 6*b*, ***(d)*** Equivalent Toffoli circuit for Fig. 6*c*

desired control lines an AIG equivalent design is obtained in Fig. 5*c*.

In a way to design equivalent circuit by taking MIG as IR, such as AIG, first, we find an MIG representation (in Fig. 5*d*) for the same function f. Then similar to AIG mapping process, we traverse the circuit level by level in a bottom-up manner and map each node of that tree with appropriate templates. We find that both the nodes 4 and 1 of the MIG graph in Fig. 5*d* have a match with template T-2, whereas node 3 and node 2 can be replaced with templates T-3 and T-5. Now, we combine all the four templates and finally get the MIG equivalent design in Fig. 6*a*.

Observation 1: We have used the same function $f = \overline{x_0}x_1x_3 + x_2x_3$ while describing the formation of circuits from IRs in examples 2 and 3. However, before the formation of circuits, if we closely look into the design of different IRs for the said function, then an important observation can be found. The equivalent BDD of the function contains four levels, whereas MIG and AIG take three levels in their respective graphs.

For small functions, (such as the stated one) the improvements in cost parameters due to level compaction are rarely seen but for very-large benchmarks this difference (number of levels) widens and simultaneously the number of nodes in the tree get reduced, which finally impacts on circuit's cost parameters resulting in many efficient designs from AIGs and MIGs.

To give more insight view on this observation and to find out which among these two IRs (AIG and MIG) provide better optimised designs for higher-order benchmarks, here we are stating the findings with instances.

Example 4: Let us consider the design shown in Fig. 6*b* representing an AIG graph for a five variable function. This design contains three regions and each region implements certain functionalities over AIG nodes. Now, consider region 1 in Fig. 6*b*, where AIG nodes have formed shared configuration (node 5 is shared by nodes 1 and 2) in-between themselves. The circuital representation of region 1 using AIG as IR is given in Fig. 6*d*.

Now, consider the shaded region of Fig. 6*c* in which there exist five AIG nodes. Very interestingly, all the five AIG nodes in that shaded part can be replaced by only two majority nodes (see Fig. 7*a*) leading to a much optimised circuit in Fig. 7*b* and this optimisation bears much significance which is reported in our next observation.

Observation 2: Two important developments can be noted here. First, we find that MIG as IR poses higher capacity in terms of graph optimisations than AIG.

Second, due to this node-level optimisation, there is a reduction in number of ancillary lines (the circuit from AIG in Fig. 6*c* requires seven ancillary lines, where Fig. 7*b* takes only four additional lines to implement the same function for region 1).

Again, when the benchmark size increases further such optimisations become wider leading to many compact representations for reversible circuits from MIG than either of AIG or BDD.

However, for small functions, MIGs and AIGs give lower level of optimisations than BDDs and this finding also can be seen in examples 2 and 3. To make this fact more vivid, now we are producing one more circuit here.

Example 5: Let us consider a function $f = x_1 \oplus x_2$. Similar to the previous designs, here also first we find the different IRs (in Figs. 7*c* and 8*a*, *c*) for the input function *f*. Then, after invoking respective mapping algorithms over the graphs, we find the final

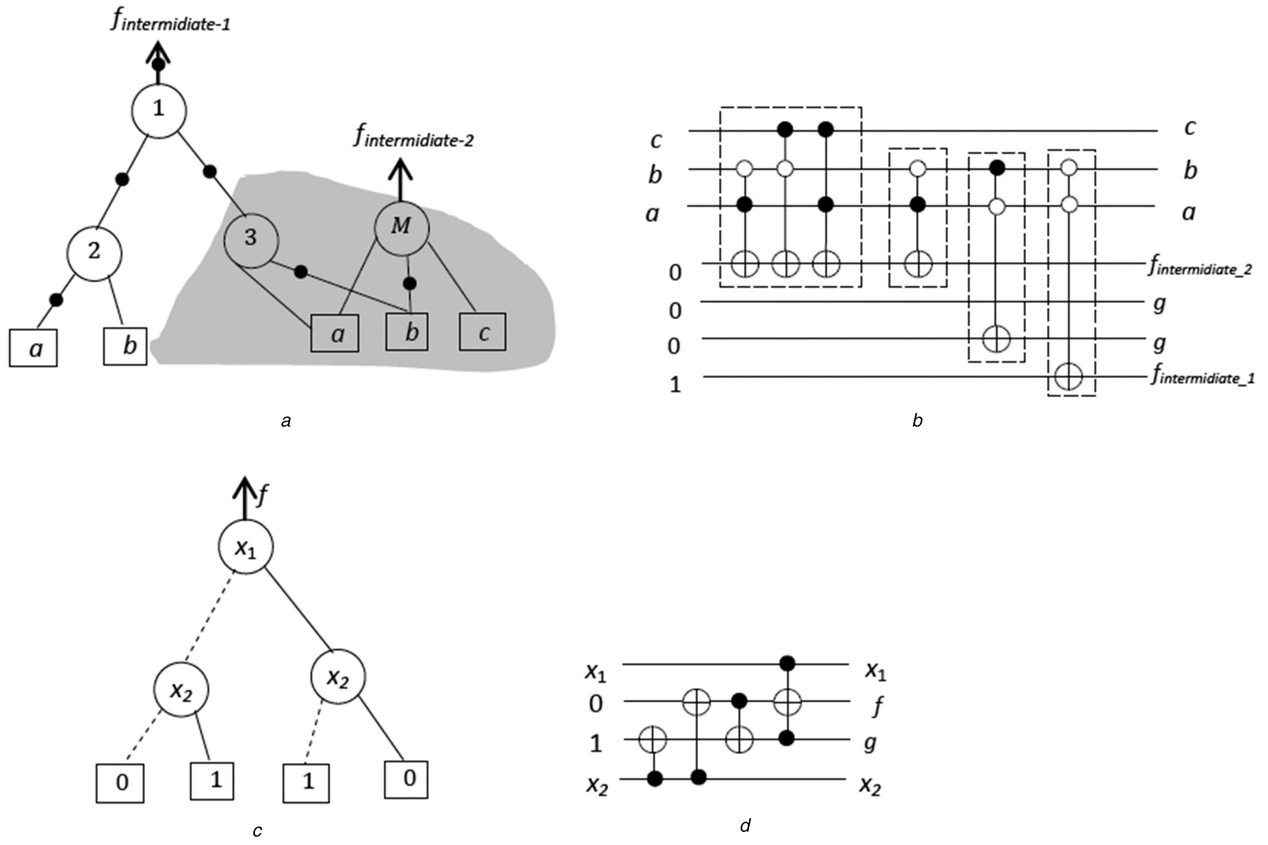


Fig. 7 *Finding reversible circuit by taking MIG as IR*
(a) Equivalent MIG for region 1, ***(b)*** Equivalent circuit of Fig. 7*a*. Constructing circuit for function $f = x_1 \oplus x_2$ by choosing BDD as IR ***(c)*** BDD for function $f = x_1 \oplus x_2$, ***(d)*** Resulting circuit for Fig. 7*c*

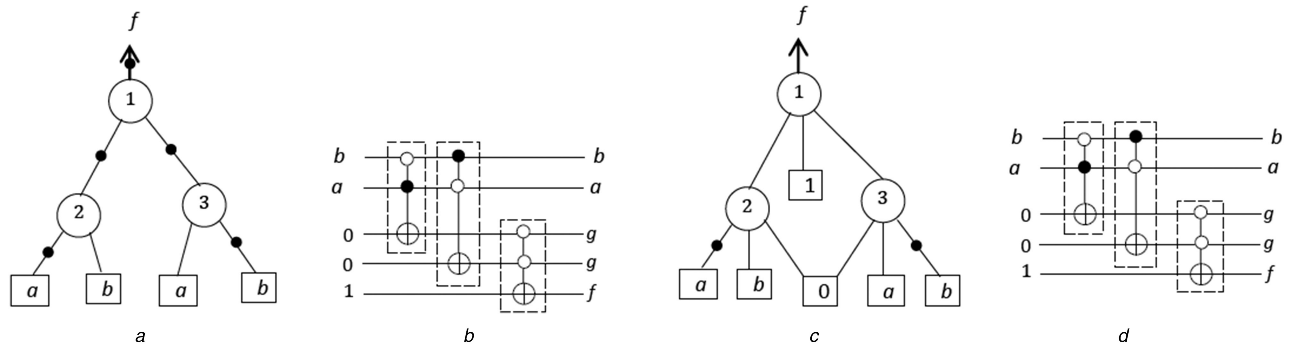


Fig. 8 *Constructing circuit for function $f = x1 \oplus x2$ by choosing AIG as IR*
(a) AIG for function $f = \bar{a}\, b + a\, \bar{b} = a \oplus b$, ***(b)*** Equivalent reversible circuit of Fig. 8*a*. Constructing circuit for function $f = x1 \oplus x2$ by choosing MIG as IR. Constructing circuit for function $f = x1 \oplus x2$ by choosing MIG as IR, ***(c)*** MIG for function $f = \bar{a}\, b + a\, \bar{b} = a \oplus b$, ***(d)*** Equivalent reversible circuit of Fig. 8*c*

circuits in Figs. 7*d* and 8*b*, *d*. Although all these circuits in the above-mentioned figures are equivalent but their designs are different which lead to different cost metrics as BDD as IR incurs eight QCs but selecting AIG and MIG as IR incurs a higher cost (QC 16) than BDD and this finding leads to the next observation.

Observation 3: Although there may exist some exceptions but the behaviour that we have seen so far indicates that for very-small functions BDD as IR performs better than both the AIG and MIG. To verify this finding, we also experimented (the obtained results are given in five tables at the end of the work) with a large spectrum of benchmarks and found that the results from those circuits also confront our claim.

Performing 'Bennett embedding' over our circuits: Still, so far, the areas that we have discussed are mainly related to cost metrics and efficient design strategies. However, there exist some design constraints in practical realisations of such circuits and one such restriction is that a reversible circuit should be designed in such a way that it should not expose intermediate results to output lines. A well known technique to meet this constraint is by performing '*Bennett embedding*' [24] in the design. Here, we are showing how the logic of '*Bennett embedding*' can be fitted over our IRs generated circuits to meet this constraint.

Example 6: Let us consider the circuit as shown in Fig. 9*a*, which operates over two qubits and has two functional outputs. First, we add two more ancillary lines (number of additional lines that are to be added in the design actually depends on the number of functional outputs the circuit has) in the existing design of Fig. 9*a*. Then, we transfer both the functional outputs to newly added ancillary lines and start uncomputing effects of gates over existing lines by reapplying the whole circuit in reverse direction. After nullifying all the effects of the qubits and constant lines, finally, the desired design is obtained in Fig. 9*b*.

Although we have successfully performed '*Bennett embedding*' in our designs but this process adds some overhead in the design as it doubles the gate count (T-count) value and increases the number of lines in the circuit.

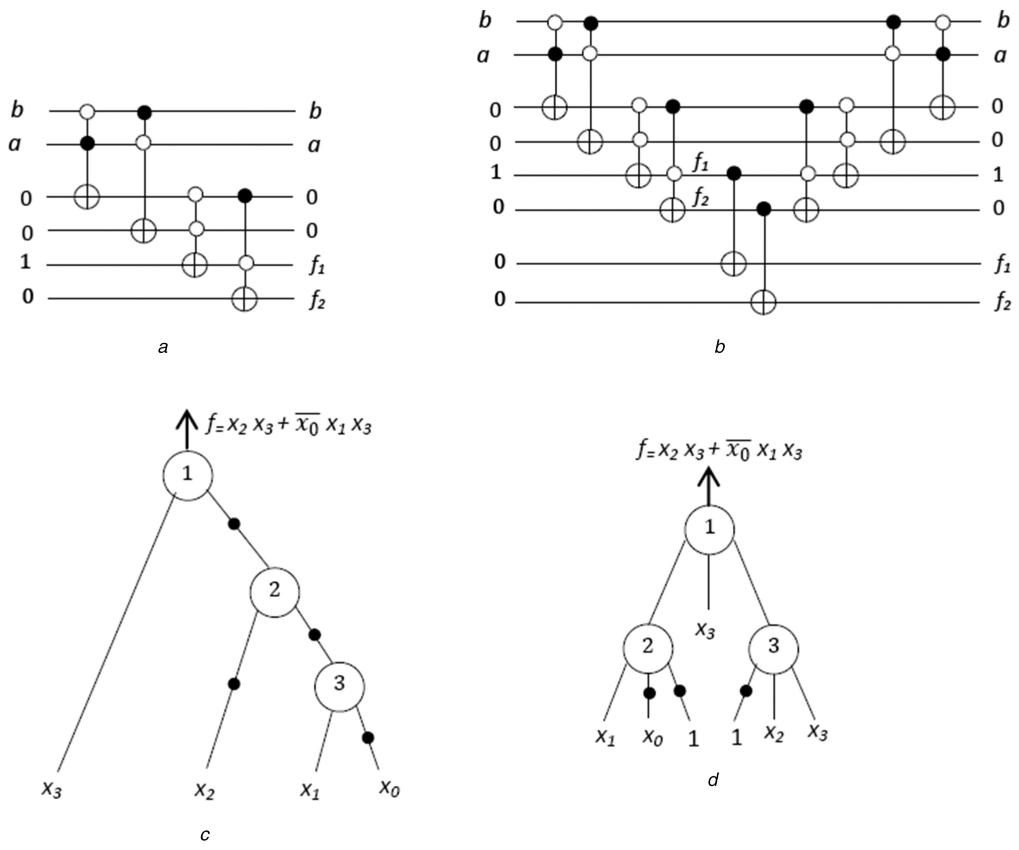


Fig. 9 *Performing 'Bennett embedding' over our IR generated circuits*
(a) Input circuit formed using AIG as IR, ***(b)*** '*Bennett embedded'* design of Fig. 9*a*. An instance when circuit cost for MIG is higher than AIG ***(c)*** AIG for function $f = x_2 x_3 + \overline{x_0} x_1 x_3$, ***(d)*** MIG for function $f = x_2 x_3 + \overline{x_0} x_1 x_3$

Table 1 BDD versus ESOP comparison on small- and medium-sized benchmarks

	First result set				
Very-small benchmarks	Size [input/output (I/O)]	BDD results [19]		ESOP results [27]	
		QC	Total lines	Q_c	Total lines
xor2	2/2	8	4	1	2
decod24_10	2/4	23	6	21	6
rd32_19	3/2	19	6	18	5
peres_4	3/3	11	7	9	6
miller_5	3/3	39	8	35	6
3_17_6	3/3	29	7	23	6
one-two-three_27	3/3	35	9	29	6
4gt11_23	4/1	5	5	5	5
4mod5_8	4/1	24	7	20	5
4gt13_25	4/1	15	7	13	5
4gt5_21	4/1	12	6	14	6
total		220	72	188	58

4 Experimental results

The stated synthesis scheme has been developed using C++ and has taken the support of MLP package [25] and ABC tool [26] to generate MIG and AIG graphs. The experiments have been performed over a large spectrum of benchmarks taken from REVLIB [27] LGSynth and École Polytechnique Fédérale De Lausanne (EPFL) [28] benchmarks suites.

We have given five different result tables containing different types of benchmarks tested over our developed algorithms, and in those tables five cost parameters [QC (Q_C), T-depth (T_d), T-count (T_c) and execution time (ET)] have been evaluated to compare each of the approaches.

A cost comparison on ESOP and BDD over small and very-small-sized benchmarks is given in Table 1.

The third and fourth tables contain a comparative study of BDD, AIG and MIG using medium- and large-sized benchmarks.

Table 2 shows the results from very-large benchmark circuits and makes a performance comparison in-between AIG and MIG generated circuits.

A comparative study in-between AIG, MIG and XMG over *Bennett embedded* logic has been summarised in Table 3.

5 Analysis and discussion over experimental results

As we have stated in our previous section (in observation 3) that for small- and medium-sized functions, BDD outperforms AIG and MIG and this finding also can be verified here from Table 4 results. However, if the function sizes are very-small (up to four variables), then it is seen that for most of the cases ESOP technique produces better designs than BDD generated circuits. To validate this statement, we have shown two sets of results in Table 1, where the

	Second result set				
Small/medium benchmarks	Size (I/O)	QC	Total lines	Q_c	Total lines
ex1_150	5/1	7	6	7	6
ex3_152	5/1	61	11	73	6
majority	5/1	41	10	106	6
C17_117	5/2	49	10	99	7
rd53	5/3	98	13	136	8
sqr6	6/12	237	49	583	10
sqn	7/3	426	40	1183	10
z5xp1	7/10	421	30	786	17
rd84	8/4	304	38	2749	12
9sym	9/1	206	27	1895	10
clip	9/5	704	57	3824	14
parity	16/1	31	31	32	17
total		2585	322	11,473	123

Table 2 Comparison of AIG versus MIG over very-large-sized benchmarks

Name	I/O	AIG				MIG			
very-large benchmarks		QC	T_d	T_c	ET	QC	T_d	T_c	ET
log2v	32/32	250,026	111,114	259,266	0.10	234,222	116,436	271,684	0.18
log_td	32/32	323,849	147,861	345,009	0.09	211,284	144,630	337,470	0.12
MUL32	64/64	80,107	35,541	82,929	0.12	49,478	32,988	76,972	0.15
sqrt_td	128/64	475,077	220,995	515,655	0.11	296,555	206,463	481,747	0.22
sqrt_tfd	128/64	420,333	205,359	479,171	0.19	287,999	201,333	469,777	0.28
adder_t	128/128	48,088	24,687	57,603	0.07	23,890	20,580	48,020	0.09
div_td	128/128	796,151	383,661	895,209	0.39	455,214	339,084	791,196	0.53
mult_t	128/128	491,016	239,544	558,936	0.37	270,382	205,980	480,620	0.42
sqrt_t	128/128	479,928	222,534	519,246	0.35	314,899	226,653	528,857	0.37
mult_tfd	128/128	420,333	205,359	479,171	0.29	243,908	179,112	417,928	0.34
i2c	147/142	7831	3345	7805	0.02	6928	3456	8064	0.05
max_t	512/128	491,016	239,544	558,936	0.62	45,657	35,055	81,795	0.65
max_td	512/130	74,068	36,516	85,204	0.52	43,334	31,998	74,662	0.57
mult_bf	512/130	396,263	193,815	452,235	0.79	215,949	162,387	378,903	0.81
total		4,754,086	2,269,875	5,296,375	4.03	2,699,699	1,906,155	4,447,695	4.78

Table 3 After performing '*Bennett embedding*' over AIG, MIG and XMG generated circuits

Name	I/O	AIG			MIG			XMG [22]			XMG [23] using LUT of size 5		
Very-large benchmarks		T_d	T_c	ET, s	T_d	T_c	ET, s	T_d	T_c	ET, s	T_d	T_c	ET, s
majority	5/1	60	140	0.03	48	112	0.08	78	182	0.01	36	84	<0.01
rd53	5/3	330	770	0.03	378	882	0.14	12,480	14,936	0.06	282	378	<0.01
squar5	5/8	384	896	0.02	450	1050	0.15	1710	2490	0.01	348	728	<0.01
sqr6	6/12	990	2310	0.02	834	1946	0.13	40,398	44,872	0.37	1116	1852	0.01
sqn	7/3	834	1946	0.02	816	1904	0.14	93,582	103,890	0.55	4968	6968	0.01
z5xp1	7/10	1212	2828	0.02	768	1792	0.12	14,238	16,886	0.10	5754	6986	0.01
rd84	8/4	1380	3220	0.03	1704	3976	0.17	111,558	120,610	0.01	3432	8008	0.01
9sym	9/1	1428	3332	0.16	1356	3164	0.03	8064	10,752	0.03	8064	10,752	0.03
clip	9/5	1050	2450	0.03	702	1638	0.18	120,420	130,300	0.08	6378	8206	3.55
cordic	23/2	498	1162	0.04	348	812	0.15	30,726	33,834	<0.01	2268	3852	0.14
log2v	32/32	222,228	518,532	0.10	232,872	543,368	0.18	10,021,758	11,417,550	54.03	17,909,160	19,851,348	225.12
apex5	117/88	12,906	30,114	0.24	12,144	28,336	0.35	47,400	63,996	0.74	37,644	53,408	0.43
adder_t	128/128	49,374	115,206	0.07	41,160	96,040	0.09	1,120,581	1,235,641	64.17	583,878	664,670	10.45
div_td	128/128	767,322	1,790,418	0.39	678,168	1,582,392	0.53	71,193,714	77,957,758	1427	126,376,530	137,866,862	1457.73

comparisons between BDD and ESOP are given. In that table, it can also be noted (in the second set) that when the benchmark size increases, then BDD performs better than ESOP.

We stated in our observation 1 and observation 2 that when the benchmark size increases, MIG gives more compact graph representations (as MIG reduces, number of levels in large functions by making the design more compact, and once the levels are reduced automatically number of nodes in the tree get lesser) than either of BDD or AIG and as the results suggest in Tables 4 and 5, for large-/very-large-sized benchmarks, MIG as IR supersedes AIG and BDD in terms of cost parameters and even AIG performs better than BDD for those benchmarks as well.

However, there are some instances in which it can be seen that AIGs have performed better than MIGs. For example, consider the designs of Figs. 9*c* and *d*. To represent the function $f = x_2 x_3 + \overline{x_0} x_1 x_3$ by taking AIG (see Fig. 9*c*) as IR will cost 15, whereas if an

Table 4 Comparison of BDD-AIG-MIG over small- and medium-sized benchmarks

Name	I/O	BDD [19]				AIG				MIG			
Small- and medium-sized benchmarks		QC	T_d	T_c	ET	QC	T_d	T_c	ET	QC	T_d	T_c	ET
majority	5/1	41	21	49	<0.01	50	30	70	0.03	40	24	56	0.08
rd53	5/3	98	48	112	<0.01	327	165	385	0.03	286	189	441	0.14
squar5	5/8	267	129	301	<0.01	320	192	448	0.02	339	225	525	0.15
sqr6	6/12	237	249	581	<0.01	825	495	1155	0.02	593	417	973	0.13
rd73	7/3	217	108	252	<0.01	900	456	1064	0.02	898	618	1442	0.16
sqn	7/3	426	219	511	<0.01	695	417	973	0.02	584	408	952	0.14
z5xp1	7/10	421	123	287	<0.01	1010	606	1414	0.02	568	384	896	0.12
rd84	8/4	304	150	350	<0.01	1356	690	1610	0.03	1265	852	1988	0.17
9sym	9/1	206	108	252	<0.01	1380	714	1666	0.16	1049	678	1582	0.03
clip	9/5	704	357	833	<0.01	1029	525	1225	0.03	520	351	819	0.18
parity	16/1	31	0	0	<0.01	253	135	315	0.03	360	216	504	0.16
spla	16/46	5925	2229	5201	0.10	13,685	8211	19,159	0.03	10,996	8790	20,510	0.19
total		8877	3741	8729	0.21	21,830	12,636	29,484	0.46	17,498	13,152	30,688	1.65

Table 5 Comparison of BDD-AIG-MIG over large-sized benchmarks

Name	I/O	BDD [19]				AIG				MIG			
Large-sized benchmarks		QC	T_d	T_c	ET	QC	T_d	T_c	ET	QC	T_d	T_c	ET
cordic	23/2	325	168	392	0.03	485	249	581	0.04	272	174	406	0.15
misex2	25/18	588	276	644	0.01	631	357	833	0.03	476	300	700	0.18
apex2	39/3	5922	3132	7308	0.24	3750	2250	5250	0.14	2540	1329	3101	0.21
seq	41/35	19,362	10,029	23,401	1.14	18,307	9750	22,750	1.01	13,724	9750	22,750	1.13
apex5	117/88	11,292	5583	13,027	0.14	11,972	6453	15,057	0.24	7888	6072	14,168	0.35
ex4p	128/28	4009	1833	4277	0.03	4603	2421	5649	0.03	2873	1809	4221	0.28
i5	133/66	1738	396	924	0.09	6401	3813	8897	0.03	772	564	1316	0.45
i8	133/81	11,478	9378	21,882	0.25	3310	18,129	42,301	0.52	7340	6852	15,988	0.67
frg2	143/139	14,944	6558	15,302	0.69	12,012	6342	14,798	0.48	8137	6243	14,567	0.18
i4	192/6	6827	891	2079	0.43	4132	2436	5684	0.34	1240	996	2324	0.52
total		76,485	38,244	89,236	3.05	65,603	52,200	121,800	2.86	45,262	34,089	79,541	4.12

equivalent circuit is constructed for the same function from MIG (in Fig. 9*d*), then it would cost 19.

Although level compaction is performed in the MIG of Fig. 9*d* but due to node configurations in the MIG graph, it costs more. Nodes 2 and 3 of design Fig. 9*d* are same as the two AIG nodes cost $5+5=10$ but node 1 costs 9 and cumulatively the entire design incurs a total cost of 19. So, it can never be guaranteed that MIG will always perform better than AIG but in most of the cases and for very higher-order circuits, where MIG makes a very-large difference in number of levels with AIG, the performance of MIG is found better than AIG.

Such as the previous instance, there may exist functions where some exceptions (though it is rarely seen) can be seen and these differences purely depend on the nature of input function types and respective intermediate graph representations.

Another set of results have been summarised in Table 3. In those result sets, it can be observed that for small benchmarks XMG with k-LUT (result obtained by taking $k=5$) sometimes produces better results than either of MIG or AIG but with the increase of function sizes XMG-based solutions cost more and simultaneously consume a higher amount of time in performing the simulation.

A further important consideration to distinguish the efficiency of the different IRs is the number of additional circuit lines that are appended during the synthesis. The approach of mapping a node to a template naturally incurs additional circuit lines (called ancillary lines in reversible circuit's context). Although ancilla-free reversible logic synthesis with BDD has been proposed in [20], for a fair evaluation between the different IRs, we restricted our experimental studies to the approaches, where ancilla is added. So far, to the best of our knowledge, there is no ancilla-free reversible logic synthesis method using structural IRs, e.g. MIG or AIG.

After analysing all the IR structures, we have seen that BDDs have fixed boundary limits, whereas AIG and MIG never give fixed boundary values. As BDD is purely decisional based data structure, where each node can have at most two child and due to that in BDD the upper bound for ancilla can be 2^n but the same cannot be said either for AIG or MIG as these IRs give structural representations in which a node can have more than two child that vary on function to function and due to that a direct relation between number input variables with number of nodes cannot be established for these two IRs.

On the other side, in the best case, the lower-bound limit for ancilla in BDD can be n if the BDD contains n number of unique nodes specifying all the n different literals in implementing the function.

To be mentioned that BDDs can have an exception for constant functions as well, where the upper- and lower-bound values for the required number of ancilla lines can be 0 as constant functions require only a single BDD node to implement the logic that does not require any additional lines (e.g. BDD for NOT gate).

Similarly, the upper-bound gate count limit for BDD can be at most 3×2^n if there exist such functions whose equivalent BDDs contain 2^n number of nodes, where all the nodes are substituted with the largest Toffoli template that contains three gates. However, the same cannot be computed either for AIG and MIG as it is not possible to specify how many nodes are there in the corresponding AIG or MIG graph.

6 Conclusion

In this work, we have reviewed the roles of different IRs in efficient synthesis of reversible circuits and have made a thorough benchmarking over the different representations. Extending the existing AIG synthesis work, we have incorporated multi-node concept in AIG synthesis and also have simplified the AIG templates as well. Technology mapping in-between MIG to reversible circuit also has been developed here. In way to studying the behaviour of different IRs, we have experimented over a large spectrum of benchmarks (arithmetic, non-arithmetic and highly scalable functions) and related findings are reported in result tables. Along with that we have made a detailed analysis where we talked about the appropriateness of IRs in finding the best designs.

7 References

[1] Soeken, M., Roetteler, M., Wiebe, N., *et al.*: 'Logic synthesis for quantum computing', arXiv preprint arXiv:1706.02721, 2017

[2] Haffner, H., Hansel, W., Roos, C.F., *et al.*: 'Scalable multi-particle entanglement of trapped ions', *Nature*, 2005, **438**, (7068), pp. 643–646

[3] Laforest, M., Simon, D., Boileau, J.-C., *et al.*: 'Using error correction to determine the noise model', *Phys. Rev. A*, 2007, **75**, (1), p. 012331

[4] Ghosh, J., Galiautdinov, A., Zhou, Z., *et al.*: 'High-fidelity controlled-*z* gate for resonator-based superconducting quantum computers', *Phys. Rev. A*, 2013, **87**, (2), p. 022309

[5] Nielsen, M.A., Chuang, I.L.: '*Quantum computation and quantum information*' (Cambridge University Press, Cambridge, 2010)

[6] Veldhorst, M., Yang, C.H., Hwang, J.C.C., *et al.*: 'A two-qubit logic gate in silicon', *Nature*, 2015

[7] Golubitsky, O., Falconer, S.M., Maslov, D.: 'Synthesis of the optimal 4 bit reversible circuits'. Proc. 47th Design Automation Conf., Anaheim, CA, USA, 2010, pp. 653–656

[8] Grobe, D., Wille, R., Dueck, G.W., *et al.*: 'Exact multiple-control Toffoli network synthesis with SAT techniques', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2009, **28**, (5), pp. 703–715

[9] Chattopadhyay, A., Chandak, C., Chakraborty, K.: 'Complexity analysis of reversible logic synthesis', arXiv preprint arXiv:1402.0491, 2014

[10] Datta, K., Rathi, G., Sengupta, I., *et al.*: 'Synthesis of reversible circuits using heuristic search method'. 2012 25th Int. Conf. VLSI Design (VLSID), 2012, pp. 328–333

[11] Li, M., Zheng, Y., Hsiao, M.S., *et al.*: 'Reversible logic synthesis through ant colony optimization'. Design, Automation & Test in Europe Conference & Exhibition (DATE) 2010, 2010, pp. 307–310

[12] Abdessaied, N., Soeken, M., Dueck, G.W., *et al.*: 'Reversible circuit rewriting with simulated annealing'. 2015 IFIP/IEEE Int. Conf. Very Large Scale Integration (VLSI-SoC), Daejeon, South Korea, 2015, pp. 286–291

[13] Fazel, K., Thornton, M., Rice, J.E.: 'ESOP-based Toffoli gate cascade generation'. IEEE Pacific Rim Conf. Communications, Computers and Signal Processing, Victoria, BC, Canada, 2007, pp. 206–209

[14] Soeken, M., Chattopadhyay, A.: 'Unlocking efficiency and scalability of reversible logic synthesis using conventional logic synthesis'. Proc. 53rd Design Automation Conf. (DAC) number EPFL-CONF-216837, Austin, TX, USA, 2016

[15] Shannon, C.E.: 'A symbolic analysis of relay and switching circuits', *Trans. Am. Inst. Electr. Eng.*, 1938, **57**, (12), pp. 713–723

[16] Mishchenko, A., Chatterjee, S., Brayton, R.: 'DAG-aware AIG rewriting a fresh look at combinational logic synthesis'. Proc. 43rd Annual Design Automation Conf., San Francisco, CA, USA, 2006, pp. 532–535

[17] Valiron, B.: 'Generating reversible circuits from higher-order functional programs'. Int. Conf. Reversible Computation, Bologna, Italy, 2016, pp. 289–306

[18] Chattopadhyay, A., Amar, L., Soeken, M., *et al.*: 'Notes on majority Boolean algebra'. Proc. IEEE Int. Symp. Multi-valued Logic (ISMVL), Sapporo, Japan, 2016, pp. 50–55, doi:10.1109/ISMVL.2016.21

[19] Wille, R., Drechsler, R.: 'BDD-based synthesis of reversible logic for large functions'. Proc. 46th Annual Design Automation Conf., San Francisco, CA, USA, 2009, pp. 270–275

[20] Soeken, M., Tague, L., Dueck, G.W., *et al.*: 'Ancilla-free synthesis of large reversible functions using binary decision diagrams', *J. Symb. Comput.*, 2016, **73**, pp. 1–26

[21] Chattopadhyay, A., Littarru, A., Amaru, L., *et al.*: 'Reversible logic synthesis via bi-conditional binary decision diagrams'. 2015 IEEE Int. Symp. Multiple-valued Logic (ISMVL), Waterloo, ON, Canada, 2015, pp. 2–7

[22] Soeken, M., Roetteler, M., Wiebe, N., *et al.*: 'Design automation and design space exploration for quantum computers'. 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 2017, pp. 470–475

[23] Soeken, M., Roetteler, M., Wiebe, N., *et al.*: 'Hierarchical reversible logic synthesis using LUTs'. Proc. 54th Annual Design Automation Conf. 2017, ACM., Austin, TX, USA, 2017, p. 78

[24] Bennett, C.H.: 'Logical reversibility of computation', *IBM J. Res. Dev.*, 1973, **17**, pp. 525–532

[25] Amaru, L., Gaillardon, P.-E., Micheli, G.D.: 'Majority-inverter graph: a novel data-structure and algorithms for efficient logic optimization'. Proc. 51st Annual Design Automation Conf., San Francisco, CA, USA, 2014, pp. 1–6

[26] Brayton, R., Mishchenko, A. (Eds.): 'ABC: an academic industrial-strength verification tool'. In (Eds.): '*Computer aided verification*' (Springer, Berlin, Heidelberg, 2010), pp. 24–40

[27] 'REVLIB': 'An online resource for reversible functions and reversible circuits'. Available at http://www.revlib.org/, accessed date 12/06/2017

[28] 'The EPFL combinational benchmark suite'. Available at http://lsisrv5.epfl.ch/benchmarks/EPFLfull.zip, last accessed 22 April 2017