Efficient Construction of QMDDs for Irreversible, Reversible, and Quantum Functions

Philipp Niemann[1], Alwin Zulehner[2], Robert Wille[1,2], and Rolf Drechsler[1,3]

[1] Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
[2] Institute for Integrated Circuits, Johannes Kepler University, Linz, Austria
[3] Department of Computer Science, University of Bremen, Bremen, Germany
Philipp.Niemann@dfki.de, {alwin.zulehner, robert.wille}@jku.at,
drechsle@informatik.uni-bremen.de

Abstract. In reversible as well as quantum computation, unitary matrices (so-called *transformation matrices*) are employed to comprehensively describe the respectively considered functionality. Due to the exponential growth of these matrices, dedicated and efficient means for their representation and manipulation are essential in order to deal with this complexity and handle reversible/quantum systems of considerable size. To this end, *Quantum Multiple-Valued Decision Diagrams* (QMDDs) have shown to provide a compact representation of those matrices and have proven their effectiveness in many areas of reversible and quantum logic design such as embedding, synthesis, or equivalence checking. However, the desired functionality is usually not provided in terms of QMDDs, but relies on alternative representations such as Boolean Algebra, circuit netlists, or quantum algorithms. In order to apply QMDD-based design approaches, the corresponding QMDD has to be constructed first—a gap in many of these approaches. In this paper, we show how QMDD representations can efficiently be obtained for Boolean functions, both reversible and irreversible ones, as well as general quantum functionality.

1 Introduction

Reversible and quantum computation are alternative computational paradigms that have received significant attention in the past decades. In contrast to conventional computation, reversible computations are information loss-less such that the inputs of a computation can always be recovered from the outputs. The absence of information loss helps (at least theoretically) to avoid energy dissipation during computations and is used for certain aspects in low-power design.[1] Moreover, superconducting quantum interference devices [14], nanoelectromechanical systems [6, 5], adiabatic circuits [1], and many further technologies utilize this computation paradigm. Reversibility of the respective operations is also an inherent characteristic of quantum computation [9]. The considered quantum systems are composed of *qubits* which, analogously to conventional bits, can represent a (Boolean) 0 or 1, but also superpositions of the two. This allows for solving many practically relevant problems (e.g. factorization [15] or database search [4])

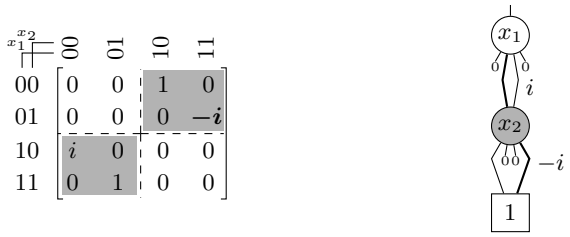
[1] Initial experiments verifying the underlying link between information loss and thermodynamics have been reported in [2].

exponentially faster than in classical computation. In both, reversible as well as quantum computation, unitary matrices (so-called *transformation matrices*) are employed to comprehensively describe the respectively considered functionality. *Quantum Multiple-Valued Decision Diagrams* (QMDDs, [13]) provide a compact, graphical representation of these matrices and allow for applying matrix operations like addition and multiplication directly on the data-structure. To this end, QMDDs have shown their effectiveness with respect to various critical tasks of reversible and quantum logic design. For example:

- Embedding: Due to the inherent reversibility of quantum and reversible logic, irreversible objective functions have to be embedded into reversible ones. For this purpose, a certain number of additional inputs (ancillary inputs) and outputs (garbage outputs) needs to be added and corresponding functionality is to be assigned in order to obtain reversibility. While it has been shown to be coNP-hard to determine an appropriate/minimal number of additional in- and outputs [17], the probably even larger problem is how to assign the additional mappings. QMDDs have been shown to be very efficient in this regard [20].
- Synthesis: Once a reversible function description is available, the synthesis problem of quantum and reversible logic is to determine an equivalent circuit representation in terms of a quantum or reversible gate library (e.g. Toffoli gates, the NCV library, or the Clifford+T library). QMDDs have successfully been employed for synthesis purposes in the past – particularly in order to realize larger reversible and quantum functionality with a minimum number of circuit lines and qubits, respectively (see e.g. [16, 11]).
- Equivalence Checking: Frequently, designers are facing different functional descriptions, e.g. before and after a technology-mapping, employing different gate libraries, unoptimized and optimized versions, etc. In these cases, it is often helpful to prove whether different descriptions indeed realize the same functionality. Since QMDDs are canonic, they are very suited to conduct corresponding equivalence checks (see e.g. [12]).

However, in most cases the desired functionality is originally not provided in terms of QMDDs, but using alternative representations such as Boolean Algebra, circuit netlists, or quantum algorithms. In order to apply the corresponding approaches, the QMDD representing the considered functionality has to be constructed first. So far, it has not been considered in the literature yet how to do that efficiently.

In fact, for the Boolean domain, there is a large body of research on the construction of various description means for Boolean functions, e.g. Boolean algebra, circuit descriptions, or graphical representations. However, the resulting representations are far from the function matrix description that is required to build the corresponding QMDD. In fact, most compact representations (algebraic or graphical) require an evaluation/traversal for each primary output in order to determine a particular input-output-mapping, i.e. a single entry of the function matrix. In the quantum domain, the desired functionality is usually given in terms of quantum algorithms or quantum circuits which are composed of modules or gates that realize a computational step (e.g. modular exponentiation) or quantum operations (e.g. rotations, controlled operations), respectively. The overall transformation matrix is computed by multiplying the matrices of the individual modules/gates, but those need to be constructed somehow first.



(a) Matrix (b) QMDD

Fig. 1. Matrix and QMDD representation of a 2-qubit quantum operation.

In this paper, we close these gaps and present detailed approaches for an efficient construction of QMDDs for Boolean as well as general quantum functionality. The paper is organized as follows: In Section 2, we provide a brief review of QMDDs. Afterwards, in Section 3 and 4, we present detailed approaches for an efficient construction of QMDDs for Boolean and quantum functionality, respectively. The results of a feasibility study to confirm the applicability of the proposed methodologies are provided in Section 5 before the paper is concluded in Section 6.

2 Quantum Multiple-Valued Decision Diagrams

In the following, we briefly introduce basic concepts and ideas of *Quantum Multiple-Valued Decision Diagrams* (QMDDs). For a more thorough introduction, we refer to [13]. QMDDs have been introduced as a data-structure for the efficient representation and manipulation of unitary, complex-valued matrices that are frequently considered in reversible and quantum computation.

Example 1. Figure 1a shows a transformation matrix of a 2-qubit quantum operation. Columns and rows (representing the inputs and outputs of the operation, respectively) are indexed by the same set of variables $\{x_1, x_2\}$.

The main idea of QMDDs is a recursive partitioning of the (square) transformation matrices and the use of edge weights to represent various complex-valued matrix entries. More precisely, a matrix of dimension $2^n \times 2^n$ is partitioned into four sub-matrices of dimension $2^{n-1} \times 2^{n-1}$ as follows:

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_{00} & \mathbf{M}_{01} \\ \mathbf{M}_{10} & \mathbf{M}_{11} \end{bmatrix}$$

This partitioning is relative to the most significant row and column variable.

Example 2. Consider again the matrix shown in Fig. 1a. This matrix is partitioned with respect to variable x_1. The sub-matrices are identified by subscripts giving the row (output) and column (input) value for that variable identifying the position of the sub-matrix within the matrix. Using this partition, a matrix can be represented as a graph with vertices as shown in Fig. 2a. The vertex is labeled by the variable associated with the partition and has directional edges pointing to vertices corresponding to the sub-matrices. More precisely, the first, second, third, and fourth outgoing edge of the vertex (from left to right) points to a vertex representing $\mathbf{M}_{00}$, $\mathbf{M}_{01}$, $\mathbf{M}_{10}$, and $\mathbf{M}_{11}$, respectively.

(a) Relation to matrix partitioning (b) Normalization of weights

Fig. 2. QMDD vertices

The partitioning process can recursively be applied to each of the sub-matrices and to each of the subsequent levels of sub-matrices until one reaches the terminal case where each sub-matrix is a single value. The result is that the initial matrix is represented by a directed, acyclic graph (DAG)—the QMDD. By traversing the tree, one can access the successively partitioned sub-matrices of the original matrix down to the individual elements.

Example 3. Figure 1b shows the QMDD for the transformation matrix from Fig. 1a. Here, the single *root vertex* (labeled x_1) represents the whole matrix and has four outgoing edges to vertices representing the top-left, top-right, bottom-left, and bottom-right sub-matrix (from left to right). This decomposition is repeated at each partitioning level until the terminal vertex (representing a single matrix entry) is reached. To obtain the value of a particular matrix entry, one has to follow the corresponding path from the root vertex at the top to the terminal vertex while multiplying all edge weights on this path. For example, the matrix entry $-i$ from the top-right sub-matrix of Fig. 1a (highlighted bold) can be determined as the product of the weights on the highlighted path of the QMDD in Fig. 1b. For simplicity, we omit edge weights equal to 1 and indicate edges with a weight of 0 by stubs.

The performed decompositions unveil redundancies in the description for which representations can be shared—eventually yielding a rather compact representation of the matrix. More precisely, the edge weights in a QMDD are normalized in order to extract common multipliers and represent sub-matrices that only differ by a scalar factor by a shared vertex.

Example 4. The top-right and bottom-left sub-matrices of the matrix in Fig. 1a (highlighted in gray) differ by a scalar factor only (namely, i) and, thus, can be represented by a single, shared QMDD vertex as shown in Fig. 1b. In order to obtain shared vertices when constructing the QMDD, the following normalization scheme is performed: for each non-terminal vertex the weights of all outgoing edges are divided by the weight of the first non-zero edge. In other words, a vertex is normalized if, and only if, the first non-zero edge has weight 1. The extracted factor is then propagated to all incoming edges as shown in Fig. 2b.

Fortunately, the simple normalization scheme from Example 4 is sufficient to obtain the maximum shared vertex compression. No improvement is possible with more sophisticated normalization schemes [10]. However, by applying different variable orders, the QMDD size can often be reduced significantly. If, in contrast, a particular variable order is fixed, QMDDs are indeed canonical representations. This means that for a given matrix the corresponding QMDD representation is unique (for a fixed normalization scheme). Moreover, efficient

algorithms have been presented for applying operations like matrix addition or multiplication directly on the QMDD data-structure.

Overall, QMDDs allow for applying matrix-based approaches in reversible and quantum logic design directly on this compact data-structure and, thus, make them applicable to systems of considerable size. However, the desired functionality needs to be on hand in terms of its QMDD representation first. As QMDDs are usually not the original description means, in the following we present a methodology for deriving QMDD representations from commonly used function representations for Boolean as well as quantum functionality.

3 Constructing QMDDs for Boolean Functionality

In this section, we describe how to obtain a QMDD representation for multi-output Boolean functions—both, irreversible and reversible ones.

3.1 General Idea and Methodology

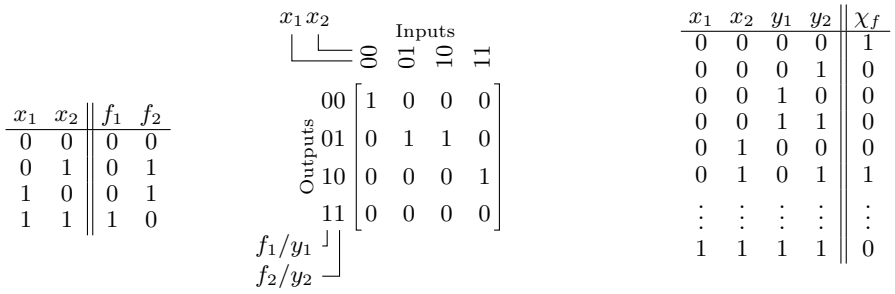
A multi-output Boolean function $f\colon \mathbb{B}^n \to \mathbb{B}^m$ is commonly given in terms of descriptions of its *primary outputs* $f_1, \ldots, f_m$ (also termed *component functions*). These single-output Boolean functions $\mathbb{B}^n \to \mathbb{B}$ are commonly described in terms of Boolean Algebra, i.e. as *Sums of Products* (SOP), *Products of Sums* (POS), or the like. In the following, we focus on SOP representations, but any other description means can be treated similarly.

Matrices, however, as required for the construction of corresponding QMDDs, are usually not employed to describe these functions—with one exception: reversible Boolean functions can be interpreted as permutations of the set $\mathbb{B}^n$ and are frequently represented as *permutation matrices*. In these $2^n \times 2^n$ matrices $\mathbf{P}_f = [p_{i,j}]_{2^n \times 2^n}$, each column (row) denotes a possible input (output) pattern. Moreover, $p_{i,j} = 1$ if, and only if, f maps the input pattern corresponding to column j to the output pattern corresponding to row i. Otherwise $p_{i,j} = 0$.

In order to have a baseline for the QMDD construction, these matrices can be generalized in a straightforward fashion to functions with different numbers of inputs and outputs. In fact, the *function matrix* of a Boolean function $f\colon \mathbb{B}^n \to \mathbb{B}^m$ needs to have the dimension $2^m \times 2^n$ in order to allow for the same correspondence of input (output) patterns and columns (rows).

Example 5. A *half adder* can be described by the multi-output Boolean function $f\colon \mathbb{B}^2 \to \mathbb{B}^2$ with component functions $f_1(x_1, x_2) = x_1 \wedge x_2$ (*carry*) and $f_2(x_1, x_2) = x_1 \oplus x_2 = x_1\overline{x_2} \vee \overline{x_1}x_2$ (*sum*). The corresponding truth-table and function matrix representations are shown in Figs. 3a and 3b, respectively. Each line of the truth-table is represented by a single 1 entry in the function matrix. For instance, the third line stating that $(1, 0)$ is mapped to $(0, 1)$ is represented by the 1 in the third column (10), second row (01).

In order to bridge the gap between the initial representation (which is essentially a more compact representation of the truth-table of f) and the targeted QMDD representation (which is essentially a more compact representation of the function matrix of f), the main idea is to employ the so-called *characteristic function* χ_f of f. This is a Boolean function $\mathbb{B}^n \times \mathbb{B}^m \to \mathbb{B}$ with n inputs labeled $x = x_1, \ldots, x_n$ and m inputs labeled $y = y_1, \ldots, y_m$, where $\chi_f(x, y) = 1$



x_1	x_2	f_1	f_2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(a) Truth-table

(b) Function matrix

x_1	x_2	y_1	y_2	χ_f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
⋮	⋮	⋮	⋮	⋮
1	1	1	1	0

(c) Char. function

Fig. 3. Representations of a half adder.

if, and only if, $f(x) = y$. In other words, χ_f evaluates to true if, and only if, the backmost m inputs represent the correct output pattern that is generated when applying f to the input pattern specified by the first n inputs. Thus, the entries of the function matrix can be interpreted as the outcomes of χ_f.

Example 6. The characteristic function of the half adder from Example 5 is shown in Fig. 3c in terms of its truth-table. Each line corresponds to one entry of the function matrix. More precisely, writing all columns of the function matrix on top of each other would yield the χ_f column of the truth-table.

As it is infeasible to construct and store the whole function matrix at once due to its exponential complexity, we rather employ compact, graphical representations of Boolean functions (especially of the characteristic functions) from which the desired QMDD representation can then be derived directly without explicitly considering the function matrix. To this end, we make use of *Binary Decision Diagrams* (BDDs, [3]). These are similar to QMDDs, but each non-terminal vertex has only two instead of four outgoing edges (termed *high* and *low edge*) and represents a (single-output) Boolean function rather than a matrix. More precisely, the function f_v of a vertex v labeled by x_i is recursively defined as

$$f_v = \left(x_i \wedge f_{high(v)}\right) \vee \left(\overline{x_i} \wedge f_{low(v)}\right),$$

where $f_{high(v)}$ and $f_{low(v)}$ denote the functions represented by the high and low child, respectively. This equation has a strong analogy to the *Shannon decomposition* of f (wrt. a primary input x_i) which is given as

$$f = (x_i \wedge f_{x_i=1}) \vee \left(\overline{x_i} \wedge f_{x_i=0}\right).$$

Here, $f_{x_i=1}$ and $f_{x_i=0}$ are the so-called *co-factors* of f which are obtained by setting the primary input x_i to 1 and 0, respectively. The analogy between the two equations, on the one hand, justifies the claim that the BDD vertices represent the Shannon decomposition of f with respect to its primary inputs and, on the other hand, yields a blueprint for how to construct the BDD representation of a given function. Alternatively, as logical operations like AND, OR, etc. can be conducted directly and efficiently on BDDs, the BDD representation of an SOP can also be constructed by first building the BDDs for the individual products

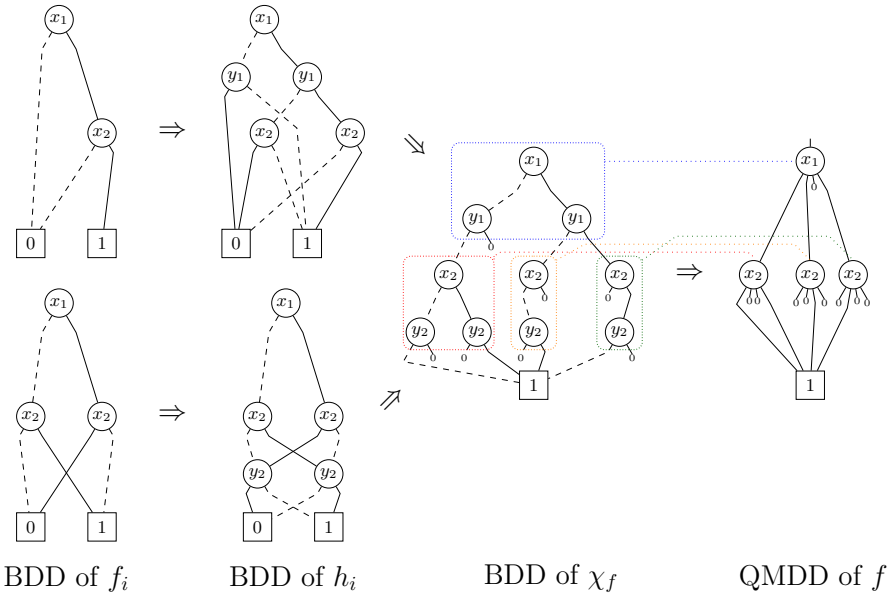


Fig. 4. Construction of the QMDD for the half adder.

and then using the BDD equivalent of the logical OR operation to "sum up" the products.[2]

Example 7. The BDDs for the component functions of the half adder reviewed in Examples 5 and 6 are shown on the left-hand side of Fig. 4.

Overall, there is a well-developed methodology for constructing the BDD representation of the component functions of f. These BDDs have then to be composed in a second step to obtain the BDD of the characteristic function χ_f. Since the outcomes of χ_f essentially describe the entries of the desired function matrix, the resulting BDD can eventually be transformed to a QMDD. In the following, these steps are described in more detail.

3.2 Generating the BDD of the Characteristic Function

In order to derive the BDD representing the characteristic function χ_f of a multi-output function $f\colon \mathbb{B}^n \to \mathbb{B}^m$, we first introduce new variables y_i for the primary outputs of f (referred to as *output variables* in the following). While the original (input) variables are used to encode the column index of the function matrix, the output variables encode rows. Then, we construct the characteristic function for each output. More precisely, we construct the helper functions h_i given by

$$h_i(x_1, \ldots, x_n, y_i) = f_i(x_1, \ldots, x_n) \odot y_i,$$

where $\odot$ denotes the XNOR-operation. This logical operation—and, thus, the entire function h_i—evaluates to true if, and only if, both operands are equal,

[2] Actually, there is a large body of research on how to derive BDD representations from various other, algebraic or netlist-based, representations of Boolean functions.

i.e. $f_i(x_1, \ldots, x_n) = y_i$. Consequently, the h_i-function can be interpreted as characteristic functions of the primary outputs of f.

Afterwards, the BDD of χ_f can be constructed by AND-ing the BDDs representing the h_i-functions as the following calculation shows:

$$\begin{aligned}
&h_1 \wedge h_2 \wedge \ldots \wedge h_m = 1 \\
\Leftrightarrow &\forall i \in \{1, \ldots, n\} : h_i = 1 \\
\Leftrightarrow &\forall i \in \{1, \ldots, n\}, (x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathbb{B}^{n+m} : f_i(x_1, \ldots, x_n) = y_i \\
\Leftrightarrow &f(x_1, \ldots, x_n) = (y_1, \ldots, y_m) \\
\Leftrightarrow &\chi_f(x_1, \ldots, x_n, y_1, \ldots, y_m) = 1
\end{aligned}$$

Remark 1. If $n > m$, i.e. if f has more primary inputs than outputs, we pad the function with zeros in order to obtain a Boolean function with the same number of inputs and outputs, such that the resulting function matrix is square. More precisely, we add $n - m$ additional constant outputs/component functions $f_j \equiv 0$. While these can, in principle, be added at any position, we add them in front of the original outputs/component functions. If, in contrast, $m > n$, we add $m - n$ additional inputs that have no impact on the functionality of f. Again, these inputs can, in principle, be added at any position, but we add them in front of the original inputs. Overall, this ensures that the original functionality is represented by the sub-matrix of dimension $2^m \times 2^n$ in the top-left corner of the square function matrix. Moreover, this allows us to assume in the following that $n = m$ without restriction.

As the BDD representing χ_f is guaranteed to be exponential in size for the variable order $x_1 \succ \ldots \succ x_n \succ y_1 \succ \ldots \succ y_m$ (at least for reversible functions), we enforce an interleaved variable order $x_1 \succ y_1 \succ x_2 \succ y_2 \succ \ldots \succ x_n \succ y_n$ when constructing the BDD for χ_f.

Example 8. Consider again the half adder example. The BDDs representing the helper functions $h_1 = f_1 \odot y_1$ and $h_2 = f_2 \odot y_2$ are computed using the BDD equivalent of the logical XNOR operation and are shown in Fig. 4 (next to the BDDs representing f_1 and f_2). By AND-ing these BDDs, we obtain the BDD representing χ_f which is shown in the center of Fig. 4. In this BDD, all edges pointing to the zero-terminal are indicated by stubs for the sake of a better readability and to emphasize the similarity to the targeted QMDD.

3.3 Transforming the BDD into a QMDD

With a BDD in interleaved variable order representing χ_f, the matrix partitioning employed by QMDDs is already laid out implicitly. In fact, corresponding bits of the column and row indices are represented by different, but adjacent variables (x_i and y_i), while QMDDs combine these in a single variable. Consequently, the BDD of χ_f can be transformed into the QMDD for f using the general transformation rule shown in Fig. 5. However, there are two special cases that have to be treated separately:

- If an input variable x_i is skipped (more precisely: a vertex labeled by y_i is the child of a vertex not labeled by x_i), this implies the x_i vertex would be redundant, i.e. high and low edge point to the same vertex. This case can

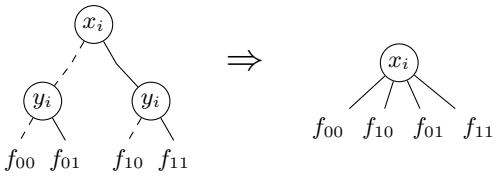
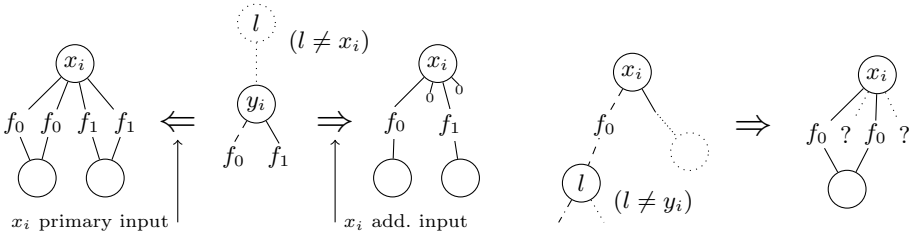


Fig. 5. General transformation rule from characteristic BDDs to QMDDs.



(a) Skipped input variables (b) Skipped output variables

Fig. 6. Handling skipped variables.

easily be handled by setting $f_{00} = f_{10} = f_0$ or $f_{01} = f_{11} = f_1$, respectively, as illustrated on the left-hand side of Fig. 6a. If, however, x_i is not an original input of the function, but has been introduced later in order obtain the same number of in- and outputs, we set $f_{10} = f_{11} = 0$ instead to ensure that the original functionality occurs only once in the final function matrix (as illustrated on the right-hand side of Fig. 6a).

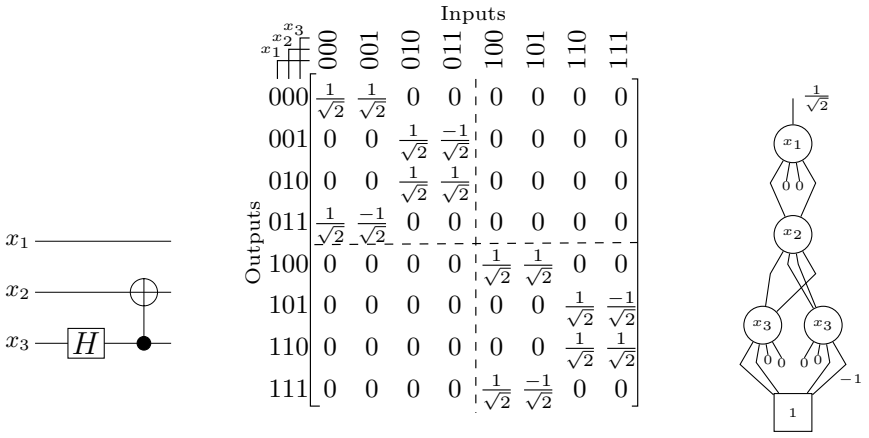
– If an output variable level y_i is skipped (more precisely: the high or low edge of a vertex labeled by x_i point to a vertex labeled by $l \neq y_i$), this implies the skipped y_i vertex would be redundant (both children would be the same). This case can easily be handled by setting $f_{00} = f_{01} = f_0$ or $f_{10} = f_{11} = f_1$, respectively, before applying the general transformation rule. For instance, the case of a skipped variable on the low edge is illustrated in Fig. 6b.

Example 9. Consider again the characteristic BDD shown in the center of Fig. 4. Here, the single x_1 vertex and the leftmost x_2 vertex can be transformed to their QMDD equivalent by applying the general transformation rule. For the remaining x_2 vertices, the methodology for skipped y_2 output variables is to be applied. Overall, this yields the QMDD shown on the right-hand side of Fig. 4.

Overall, following this procedure yields a QMDD representing the function matrix (in case of a reversible function, a permutation matrix) of any Boolean function f originally provided in terms of an SOP.

4 Constructing QMDDs for Quantum Functionality

In this section, we describe how to efficiently construct a QMDD representing desired quantum functionality. General quantum functionality is usually either given (a) in terms of an abstract *quantum algorithm* which describes a series of computational steps or complex quantum operations (modules) to be conducted



(a) Quantum circuit (b) Transformation matrix (c) QMDD

Fig. 7. Different representations of quantum functionality.

or (b) in terms of a *quantum circuit* consisting of a cascade of elementary quantum operations (so-called *quantum gates*) that form a more complex operation.

Example 10. Consider the 3-qubit quantum circuit shown in Fig. 7a. Horizontal lines represent qubits. Quantum gates, i.e. $\boxed{H}$ (a Hadamard operation) and $\bullet\!\!-\!\!\oplus$ (a controlled NOT, $CNOT$), are applied successively from left to right. The corresponding transformation matrix is depicted in Fig. 7b. As for any matrix of a linear transformation, columns denote input basis vectors and rows denote output basis vectors. In the quantum domain, the basis vectors are called *basis states* and are commonly denoted as $|x_1x_2x_3\rangle$ using the so-called *ket-notation*. For instance, the basis state $|001\rangle$ is mapped to the linear combination (*superposition*) $\frac{1}{\sqrt{2}}|000\rangle - \frac{1}{\sqrt{2}}|011\rangle$. Note that there is a strong relationship between the partitioning of the matrix with respect to a variable x_j and the input/output mapping of the corresponding qubit. More precisely, the top-left sub-matrix of a partitioning represents the mapping $|0\rangle \mapsto |0\rangle$, the top-right sub-matrix represents the mapping $|1\rangle \mapsto |0\rangle$, and so on. This transfers to the corresponding QMDD vertices such that the outgoing edges represent the mappings $|0\rangle \mapsto |0\rangle$, $|1\rangle \mapsto |0\rangle$, $|0\rangle \mapsto |1\rangle$, and $|1\rangle \mapsto |1\rangle$ from left to right and are denoted by $e_{00}, e_{10}, e_{01}, e_{11}$ in the following. Finally, the corresponding QMDD is depicted in Fig. 7c.

For quantum algorithms as well as circuits, the representation/description of the overall functionality is successively built from functional descriptions/representations of the individual parts (modules or gates). More precisely, for a cascade of modules/gates $g_1g_2 \dots g_l$ where the transformation for module/gate g_i is defined by matrix $\mathbf{M}_i$, the transformation for the complete algorithm/circuit is given by the direct matrix product $\mathbf{M}_l \cdot \mathbf{M}_{l-1} \cdot \ldots \cdot \mathbf{M}_1$. Note that the order of the matrices has to be reversed to achieve the correct order of applying the modules/gates (first g_1, then g_2, etc.). To construct this matrix product, the QMDDs for the single modules/gates simply have to be multiplied using the

QMDD-based algorithm for matrix multiplication. Consequently, for the remainder of this section we focus on how the QMDD representations for elementary quantum gates can be constructed efficiently.

A gate g is specified by the 2×2 base transition matrix $\mathbf{B}$, the target qubit x_t and a possible empty set of control qubits $C \subset \{x_1, \ldots, x_n\}$ (with $x_t \notin C$) together with a map $\alpha \colon C \to \{|0\rangle, |1\rangle\}$ which describes the activating values, i.e. qubit basis states, of each control qubit.

Example 11. The base transition matrix of the first gate of the quantum circuit in Fig. 7a (Hadamard gate) is given by $\mathbf{H} = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$. This gate has a target x_3 and no controls, i.e. $C = \emptyset$. The second gate of the circuit is a controlled NOT gate with the base transition matrix $\mathbf{X} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, a target x_2, and one positive control, i.e. $C = \{x_3\}$ with $\alpha(x_3) = |1\rangle$. This gate effectively swaps the basis states $|0\rangle$ and $|1\rangle$ on qubit x_2 if, and only if, qubit x_3 is in the $|1\rangle$-state.

The QMDD for a quantum gate is built variable by variable (qubit by qubit) in a bottom-up fashion from the terminal to the root vertex. To this end, we assume the variable order $x_1 \succ x_2 \succ \ldots \succ x_n$ from the root vertex towards the terminal vertex. In order to indicate which set of variables has been processed so far, we use the notation $\mathbf{M}_{\{x_k, \ldots, x_n\}}$. Moreover, for the sake of an easier reference, we term those edges of a QMDD vertex *diagonal* that correspond to a $|i\rangle \to |i\rangle$ mapping ($i = 0, 1$), i.e. e_{00} and e_{11}, and the remaining edges *off-diagonal*.

Although it is possible to construct the QMDD for the gate in a single run as roughly sketched in [8], for a better understanding we follow [13] and construct two QMDDs representing the cases that the gate is active (all control qubits are in their activating state) or inactive (at least one control qubit is not).[3] By adding these QMDDs, the actual QMDD for the gate results.

Case "gate is active", i.e. the base transition $\mathbf{B}$ is performed on qubit x_t if, and only if, all controls are in their activating state. All other qubits preserve their original state.

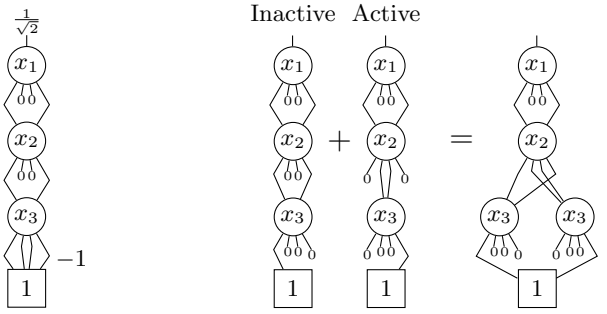
Consequently, the QMDD for the active case contains all (non-zero) paths of the final QMDD for which all decision variables (qubits) except for the target have an activating assignment.

In order to have a valid starting point, we begin at the terminal level with an edge pointing to the terminal vertex with weight 1, i.e. $\mathbf{M}_{\emptyset} = [1]_{1 \times 1}$.[4] Afterwards, the qubits are processed in a bottom-up fashion. If the current qubit x_c

- is neither a control nor the target, i.e. $x_c \neq x_t, x_c \notin C$, the gate is active regardless of the qubit's state. Consequently, at the matrix level the result is $\mathrm{id}_{\mathbf{2} \times \mathbf{2}} \otimes \mathbf{M}_{\{x_{c+1}, \ldots, x_n\}}$ which corresponds to a QMDD vertex labeled x_c where all diagonal edges point to the existing QMDD and all remaining edges are 0-edges.

[3] Without loss of generality, we consider only basis states of the underlying quantum system, i.e. each qubit is assumed to be in one of its basis states. Due to the linearity of quantum operations, these are sufficient to construct the corresponding transformation matrix which yields the correct behaviour also for the case of superposed input states.

[4] The appropriate weights of the base transition will be incorporated later.



(a) First gate (b) Second gate

Fig. 8. QMDD representations for the gates from the quantum circuit in Fig. 7a.

- is a control, i.e. $x_c \in C$, the gate is only active for one control value $|i\rangle = \alpha(x_c)$. Consequently, the result is a vertex labeled x_c with only 0-edges except from the edge $|i\rangle \to |i\rangle$ which points to the existing QMDD.
- is the target, i.e. $x_c = x_t$, the base transition is performed. Consequently, the result is $\mathbf{B} \otimes \mathbf{M}_{\{x_{c+1},\ldots,x_n\}}$, i.e. a vertex labeled x_t with all edges pointing to the existing QMDD with the corresponding edge weight taken from the base transition matrix $\mathbf{B}$ (if a weight is zero, the corresponding edge is a 0-edge directly pointing to the terminal).

During this construction, the QMDD is normalized as described in Example 4.

Example 12. Consider the QMDD in Fig. 8a which represents the first gate of the quantum circuit shown in Fig. 7a. As this gate does not have any controls, it is always active and, thus, it suffices to build the QMDD representing the active part. We start with an edge to the terminal vertex with weight 1. As the bottom-most qubit is already the target qubit, all edges of the x_3-vertex point directly to this terminal with the appropriate weight of the Hadamard transformation matrix $\mathbf{H} = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$. Note that normalization will propagate the common multiplier $\frac{1}{\sqrt{2}}$ of this matrix to the root edge. The remaining qubits are neither control nor target. Thus, vertices representing an identity mapping of these qubits are inserted.

The QMDD for the inactive case is constructed similarly.

Case "gate is inactive", i.e. the identity transition is performed on qubit x_t since at least one control is not in its activating state. All qubits preserve their original state, i.e. none but diagonal edges are populated at all.
Consequently, the QMDD for the inactive case contains all (non-zero) paths of the final QMDD for which at least one decision variable (qubit) does not have an activating assignment.
However, when constructing the QMDD in a bottom-up fashion, we always use the hypothesis that all controls above the current qubit are in their activating states and at least one control below is not.
To make sure that this hypothesis gives the correct result even for the bottom-most control (for which no inactive control may exist below), we start at the terminal level with an edge pointing to the terminal vertex with

weight 0, i.e. $\mathbf{M}_{\emptyset} = [0]_{1\times 1}$. This ensures that all edges corresponding to the activating value of this bottom-most control are 0-edges.
The remaining qubits are processed as follows. If the current qubit x_c

- is neither a control nor the target, i.e. $x_c \neq x_t, x_c \notin C$, the gate is inactive regardless of the qubit's state. Consequently, at the matrix level the result is $\mathrm{id}_{\mathbf{2}\times\mathbf{2}} \otimes \mathbf{M}_{\{x_{c+1},\ldots,x_n\}}$ which corresponds to a QMDD vertex labeled x_c where all diagonal edges point to the existing QMDD and all remaining edges are 0-edges.
- is a control, i.e. $x_c \in C$, the gate is definitely inactive for all but one control value $|i\rangle = \alpha(x_c)$. For the latter, the activity of the gate depends on the remaining qubits. Consequently, the result is a vertex with all diagonal edges pointing to the k-fold tensor product $\mathrm{id}_{\mathbf{2}\times\mathbf{2}}{}^{\otimes k}$ (nothing happens to all k qubits below the current one) except from the edge $|i\rangle \to |i\rangle$. The latter handles the case that the qubit is in its activating state and is pointing to the existing QMDD $\mathbf{M}_{\{x_{c+1},\ldots,x_n\}}$.[5] All off-diagonal edges are 0-edges.
- is the target, i.e. $x_c = x_t$, the identity transformation is performed on the target. Consequently, the result is $\mathrm{id}_{\mathbf{2}\times\mathbf{2}} \otimes \mathbf{M}_{\{x_{c+1},\ldots,x_n\}}$ like in the unconnected case.

Example 13. The QMDDs for the circuit's second gate is shown in Fig. 8b.

For the inactive part, we start with a 0-edge. For the control on x_3, we construct a vertex which uses this 0-edge as e_{11} and for which the other diagonal edge e_{00} represents the identity $\mathrm{id}_{\mathbf{2}\times\mathbf{2}}{}^{\otimes 0} = [1]_{1\times 1}$, i.e. it points to the terminal vertex with weight 1. As x_3 is the only control, we simply add vertices representing an identity mapping for the remaining qubits.

For the active part, we start with an edge to the terminal vertex which becomes the e_{11} edge of the x_3-vertex, as the activating state of x_3 is $|1\rangle$. For the target qubit x_2 with the base transition matrix $\mathbf{X} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, an x_2-vertex is added. For this vertex, both off-diagonal edges point to the x_3-vertex constructed before (with weight 1 as the corresponding entry in $\mathbf{X}$ is 1) and both diagonal edges are 0-edges (as the corresponding entry in $\mathbf{X}$ is 0). Last, but not least, for the unconnected qubit x_1 a vertex representing its identity mapping is added. Finally, by adding the QMDDs for the inactive and active part, we obtain the actual QMDD for the CNOT gate.

Overall, the resulting QMDDs for the active as well as the inactive part of the gate are linear in the number of variables—regardless of the complexity of the gate under consideration. Both QMDDs can be constructed in parallel while iterating through the variables in a bottom-up fashion. In addition, they describe disjoint parts of the gate matrix, while they are padded with zeros outside of that particular part. Consequently, their sum can be computed in linear time and will also be linear in size. In fact, there are only trivial additions where at least one of the summands is a 0-matrix and, as already recognized in [8], the addition could be saved entirely, such that the whole construction could be performed in a single pass from the terminal to the root vertex with no backtracking or recursion. Either way, QMDD representations for single gates can be computed very efficiently and the potentially rather expensive part of constructing a QMDD

[5] If there is no further control below the current qubit, the gate inactivity is ensured by choosing a 0-edge as the initial QMDD.

representation for quantum algorithms or quantum circuits (as well as any other quantum logic representation) is given by the (QMDD-based) matrix multiplication that is required to concatenate the representations of single modules/gates.

5 Feasibility Study

In this section, we demonstrate the applicability of the discussed methods for QMDD construction. To this end, we implemented them in C++ on top of the QMDD package (provided together with [13]) and the BDD package *CUDD* [18]. As benchmarks for the construction of QMDDs representing Boolean functions, we considered functions from *RevLib* [19]. For quantum benchmarks, we considered quantum realizations of the Boolean functions from RevLib, realizations of the Quantum Fourier Transformation and Grover's search algorithm (cf. [9]), implementations of error-correcting codes (taken from [7]) as well as randomly generated Clifford group circuits. All experiments have been conducted on a 4 GHz processor with 32 GB of memory running Linux 4.4.

Table 1a lists the results for the QMDD construction for Boolean functions (reversible as well as non-reversible ones). The first three columns list the name of the benchmark as well as the number of primary inputs and primary outputs (denoted by PI and PO, respectively). Note that the number of variables of the resulting QMDD is accordingly given by $n = \max(PI, PO)$. The remaining two columns of Table 1a list the run-time (in CPU seconds) required for constructing the QMDD and the size of the resulting QMDD (i.e. its number of vertices).

The numbers show that the QMDDs construction could either be conducted in negligible run-time (i.e. in less than a second) or fails by running into a timeout of 10 000 seconds (denoted by TO). However, the latter case was only observed for two benchmarks with more than 100 QMDD variables. The limiting factor in these cases was the construction of the characteristic function, since the variables in the BDD have to adhere to a certain order. More precisely, the variables representing primary inputs and primary outputs are interleaved – allowing that the transformation of the characteristic function into a QMDD can be conducted as described in Section 3.3. While certainly a limitation, this is in line with the characteristic matrix partitioning of QMDDs, i.e. QMDDs eventually employ a similar order (only with the difference that corresponding PIs and POs are jointly considered in a single vertex). That is, the QMDD data-structure itself is the limiting factor for these two functions; not the proposed construction method.

Table 1b shows the obtained results for constructing QMDDs for quantum computations. The first three columns of the table list the name of the benchmarks, the number of qubits n as well as the number of gates of the quantum circuit $|G|$. The remaining two columns list the run-time required to construct the QMDD and its number of vertices. Here, the numbers show that a QMDD can be constructed for quantum circuits composed of more than one thousand gates quite efficiently. Indeed, we observe a run-time of less than one second for most cases. However, there are also a few benchmarks for which the time to construct the QMDD takes longer (e.g. *Clifford-20* or *Clifford-25*), but these are exactly the cases where the size of the resulting QMDD is large.

6 Conclusions

In this work, we considered how to efficiently construct a QMDD representation for Boolean functions, reversible and non-reversible ones, as well as quantum

Table 1. Feasibility Study

(a) Boolean functions

Benchmark	PI	PO	t	size
5xp1_90	7	10	0.10	342
sao2_199	10	4	0.11	138
urf3_75	10	10	0.15	1001
urf4_89	11	11	0.22	2774
add6_92	12	7	0.11	309
alu1_94	12	8	0.12	189
apla_107	10	12	0.12	288
cycle10_2_61	12	12	0.13	66
sqr6_204	6	12	0.11	112
0410184_85	14	14	0.26	38
alu4_98	14	8	0.16	1471
cu_141	14	11	0.10	165
misex3c_181	14	14	0.13	522
table3_209	14	14	0.12	934
tial_214	14	8	0.14	1503
ham15_30	15	15	0.50	2021
in0_162	15	11	0.10	492
urf6_77	15	15	0.53	2312
cmb_134	16	4	0.11	86
decod_137	5	16	0.12	111
apex4_103	9	19	0.16	1189
cm151a_129	19	9	0.11	141
mux_185	21	1	0.15	145
cordic_138	23	2	0.11	132
bw_116	5	28	0.11	432
frg1_160	28	3	0.12	417
apex2_101	39	3	0.22	1797
pdc_191	16	40	0.29	1800
seq_201	41	35	0.35	1881
spla_202	16	46	0.25	1538
ex5p_154	8	63	0.20	1139
e64_149	65	65	0.22	1161
cps_140	24	109	0.92	3763
apex5_104	117	88	TO	–
frg2_161	143	139	TO	–

(b) Quantum functionality

Benchmark	n	$\|G\|$	t	size
QFT-3	3	9	0.11	21
QFT-4	4	16	0.10	85
QFT-5	5	21	0.11	341
QFT-6	6	30	0.11	1365
7-qubit-code	7	18	0.11	29
Grover-3	7	83	0.10	209
hwb6_56	7	1153	0.15	87
QFT-7	7	37	0.11	5461
5-qubit-code	9	24	0.11	83
9-qubit-code-A	9	11	0.11	25
Grover-4	9	106	0.12	1075
rd73_252	10	660	0.18	42
9symml_195	11	2945	0.49	53
dc1_221	11	290	0.11	64
Grover-5	11	131	0.18	2816
cycle10_2_110	12	722	0.13	66
adr4_197	13	426	0.18	174
dist_223	13	3544	2.42	284
radd_250	13	327	0.16	151
co14_215	15	1290	0.29	65
dc2_222	15	1218	0.67	360
ham15_107	15	1101	0.25	4521
Clifford-15	15	100	1.43	22697
5xp1_194	17	931	0.36	719
9-qubit-code-B	17	40	0.11	1075
Clifford-18	18	100	0.84	21609
Clifford-20	20	100	10.21	76473
decod_217	21	845	0.15	187
pcler8_248	21	289	0.26	1083
apla_203	22	2051	1.43	421
cu_219	25	752	1.28	525
Clifford-25	25	100	284.16	580992
cm151a_211	28	639	45.13	6408
cm163a_213	29	575	3.44	1205
add64_184	193	576	0.14	701

functionality. These representations are essential for the efficiency of various approaches in reversible/quantum logic design, but are usually not the originally provided description means. For the Boolean case, we developed a methodology to obtain and transform the BDD of the characteristic function which is structurally already very similar to the desired QMDD. For the quantum case, we focused on the construction of QMDD representations for elementary quantum gates from which the representation of the entire circuit or algorithm can be obtained using (QMDD-based) matrix multiplication. The feasibility of the proposed methods has been confirmed on several examples. In fact, the obtained results showed that the construction can be conducted in negligible run-time when the characteristic matrix partitioning of QMDDs allows for an efficient representation. Overall, this work closes an important gap for several design solutions based on QMDDs e.g. for embedding, synthesis, or verification.

Acknowledgements

This work has partially been supported by the European Union through the COST Action IC1405.

References

1. Athas, W., Svensson, L.: Reversible logic issues in adiabatic CMOS. In: Proc. Workshop on Physics and Computation, 1994. PhysComp '94. pp. 111–118 (1994)
2. Berut, A., Arakelyan, A., Petrosyan, A., Ciliberto, S., Dillenschneider, R., Lutz, E.: Experimental verification of Landauer's principle linking information and thermodynamics. Nature 483, 187–189 (2012)
3. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. on Comp. 35(8), 677–691 (1986)
4. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Theory of computing. pp. 212–219 (1996)
5. Houri, S., Valentian, A., Fanet, H.: Comparing CMOS-based and NEMS-based adiabatic logic circuits. In: Conference on Reversible Computation, pp. 36–45 (2013)
6. Merkle, R.C.: Reversible electronic logic using switches. Nanotechnology 4(1), 21 (1993)
7. Mermin, N.D.: Quantum Computer Science: An Introduction. Cambridge University Press (2007)
8. Miller, D.M., Thornton, M.A.: QMDD: A decision diagram structure for reversible and quantum circuits. In: Int'l Symp. on Multi-Valued Logic. p. 6 (2006)
9. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge Univ. Press (2000)
10. Niemann, P., Wille, R., Drechsler, R.: On the "Q" in QMDDs: Efficient representation of quantum functionality in the QMDD data-structure. In: Conference on Reversible Computation. pp. 125–140 (2013)
11. Niemann, P., Wille, R., Drechsler, R.: Efficient synthesis of quantum circuits implementing Clifford group operations. In: ASP Design Automation Conf. pp. 483–488 (2014)
12. Niemann, P., Wille, R., Drechsler, R.: Equivalence checking in multi-level quantum systems. In: Conference on Reversible Computation. pp. 201–215 (2014)
13. Niemann, P., Wille, R., Miller, D.M., Thornton, M.A., Drechsler, R.: QMDDs: Efficient quantum function representation and manipulation. IEEE Trans. on CAD 35(1), 86–99 (2016)
14. Ren, J., Semenov, V., Polyakov, Y., Averin, D., Tsai, J.S.: Progress towards reversible computing with nSQUID arrays. IEEE Transactions on Applied Superconductivity 19(3), 961–967 (2009)
15. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. Foundations of Computer Science pp. 124–134 (1994)
16. Soeken, M., Wille, R., Hilken, C., Przigoda, N., Drechsler, R.: Synthesis of reversible circuits with minimal lines for large functions. In: ASP Design Automation Conf. pp. 85–92 (2012)
17. Soeken, M., Wille, R., Keszocze, O., Miller, D.M., Drechsler, R.: Embedding of large Boolean functions for reversible logic. J. Emerg. Technol. Comput. Syst. 12(4), 41:1–41:26 (2015)
18. Somenzi, F.: Efficient manipulation of decision diagrams. Software Tools for Technology Transfer 3(2), 171–181 (2001)
19. Wille, R., Große, D., Teuber, L., Dueck, G.W., Drechsler, R.: RevLib: an online resource for reversible functions and reversible circuits. In: Int'l Symp. on Multi-Valued Logic. pp. 220–225 (2008), RevLib is available at http://www.revlib.org
20. Zulehner, A., Wille, R.: Make it reversible: Efficient embedding of non-reversible functions. In: Design, Automation and Test in Europe. pp. 458–463 (2017)