QMDDs: Efficient Quantum Function Representation and Manipulation

Philipp Niemann, *Member, IEEE*, Robert Wille, *Senior Member, IEEE*, David Michael Miller, *Member, IEEE*, Mitchell A. Thornton, *Senior Member, IEEE*, and Rolf Drechsler, *Fellow, IEEE*

***Abstract*—Quantum mechanical phenomena such as phase shifts, superposition, and entanglement show promise in use for computation. Suitable technologies for the modeling and design of quantum computers and other information processing techniques that exploit quantum mechanical principles are in the range of vision. Quantum algorithms that significantly speed up the process of solving several important computation problems have been proposed in the past. The most common representation of quantum mechanical phenomena are transformation matrices. However, the transformation matrices grow exponentially with the size of a quantum system and, thus, pose significant challenges for efficient representation and manipulation of quantum functionality. In order to address this problem, first approaches for the representation of quantum systems in terms of decision diagrams have been proposed. One very promising approach is given by Quantum Multiple-Valued Decision Diagrams (QMDDs) which are able to efficiently represent transformation matrices and also inherently support multiple-valued basis states offered by many physical quantum systems. However, the initial proposal of QMDDs was lacking in a formal basis and did not allow, e.g., the change of the variable order—an established core functionality in decision diagrams which is crucial for determining more compact representations. Because of this, the full potential of QMDDs or decision diagrams for quantum functionality in general has not been fully exploited yet. In this paper, we present a refined definition of QMDDs for the general quantum case. Furthermore, we provide significantly improved computational methods for their use and manipulation and show that the resulting representation satisfies important criteria for a decision diagram, i.e., compactness and canonicity. An experimental evaluation confirms the efficiency of QMDDs.**

***Index Terms*—Decision diagrams, function representation, quantum computation, reversible logic.**

I. Introduction

EXPLOITING quantum mechanical phenomena within a quantum computer promises to solve important computation problems significantly faster than on conventional computers [1]. For instance, the superposition of basis states and the use of entanglement enables massive parallelism which has the potential to speed up tasks such as integer factorization [2], [3] or database search [4].

A quantum system is composed of a collection of particles or other entities that have r characteristic basis states. If the quantum system is composed of n such entities, the overall state of the system at some instant of time is represented as a vector with r^n complex-valued components. A change in the state of the quantum system is described by a change in the values of the quantum state vector components. The state of a quantum system at two different instances of time is represented by two corresponding state vectors and, when the two state vectors differ, the system has changed its state or evolved. Quantum system evolutions can be represented by a linear transformation matrix relating two quantum state vectors. For physical reasons, all such matrices are unitary.

Quantum algorithms or logic circuits can be expressed as a specific evolution matrix that describes the transformation of an initial quantum state vector to a desired output state representing the result of the computation. In the design and analysis of quantum systems, the availability of an efficient representation of these matrices is essential. Many analysis and synthesis tasks require that the overall transformation matrix to be decomposed into a set of product matrices. For instance, the synthesis problem can be considered as the decomposition of the overall matrix into a product of matrices where each product term is in the form of a known transformation matrix that represents a particular quantum computer instruction or logic gate [5]–[7]. The equivalence checking problem can likewise be formulated by extracting the transformation matrices from two different implementations of a quantum circuit or algorithm and verifying that the matrices are identical [8].

This viewpoint provides motivation for a compact and unique means for the representation of unitary transformation matrices as well as efficient operations like matrix multiplication to be performed directly on these representations. As the matrices grow exponentially with the size of the quantum systems, dedicated representations that exploit structures within the matrices have to be employed. For this purpose, representations in the form of decision diagrams such as the *Quantum Information Decision Diagram* (QuIDD [9]) or the *Quantum Decision Diagram* (QDD [5]) have been proposed. QuIDD and QDD are basically extensions of the well-known *Binary Decision Diagram* (BDD [10]) with multiple terminals and rotation matrices applied as edge weights, respectively.

Manuscript received December 20, 2014; revised April 25, 2015; accepted May 29, 2015. Date of publication July 21, 2015; date of current version December 18, 2015. This paper was recommended by Associate Editor Y. Chen.

P. Niemann is with the University of Bremen, 28359 Bremen, Germany (e-mail: pniemann@informatik.uni-bremen.de).

R. Wille and R. Drechsler are with the University of Bremen, 28359 Bremen, Germany and also with DFKI GmbH, 28359 Bremen, Germany (e-mail: rwille@informatik.uni-bremen.de; drechsle@informatik.uni-bremen.de).

D. M. Miller is with the University of Victoria, Victoria, BC V8P 5C2, Canada (e-mail: mmiller@uvic.ca).

M. A. Thornton is with the Lyle School of Engineering, Southern Methodist University, Dallas, TX 75205 USA (e-mail: mitch@lyle.smu.edu).

Digital Object Identifier 10.1109/TCAD.2015.2459034

Because BDD vertices can be considered as a graphical representation of a Shannon decomposition, they provide a very efficient means for representing deterministic switching functions such as the Boolean functions used to model conventional digital logic circuitry. However, the Shannon decomposition is not a basic decomposition for quantum mechanical phenomena, thus, the use of BDD-based decision diagrams is not the best choice for quantum circuits or algorithms. A decision diagram based upon a decomposition that more naturally models quantum mechanical systems provides a better alternative since it provides a compact representation while also allowing for a more natural set of manipulation algorithms to be developed. As a consequence, QuIDD and QDD rather emulate the representation of quantum functionality and, hence, show their limitations when sophisticated quantum functionality is considered on a larger scale [11].

As a complementary approach, *Quantum Multiple-valued Decision Diagrams* (QMDDs; initially proposed in [12] and [13]) have been considered. Here, the potentially complex-valued matrix entries as well as the possible multiple-valued basis states are explicitly supported. Moreover, the resulting concepts allow for a canonic representation. All this motivated the application of QMDDs for several purposes such as simulation [14], equivalence checking [8], and synthesis [6], [7].

However, the initial proposal for QMDDs had a major shortcoming: local transformations such as those required for adjacent variable swapping were not provided for the quantum case. This is a significant issue since adjacent variable interchange is a central approach for determining a variable ordering that reduces the size of a decision diagram such as sifting [15]. As a consequence, QMDDs, while effective for classical reversible functions where sifting does work, have seen limited use for complex quantum functionality. This situation has led to modifications and workarounds to the definition of QMDD as e.g., done in [16]. But no comprehensive definition and implementation addressing these drawbacks have been proposed thus far.

In this paper, we provide new formulations and new considerations of the foundations of QMDDs which allow for overcoming the shortcomings mentioned above. Focusing on the two central properties (compactness and canonicity), a comprehensive description of the underlying basic concepts is provided. Furthermore, we show that QMDDs indeed provide a proper platform for efficient quantum function manipulation. This explicitly includes the realization of matrix operations directly on QMDDs as well as the construction of QMDDs from circuit descriptions. Finally, how to efficiently conduct local QMDD modifications is covered and, thus, the main shortcoming of the initial proposal are addressed. This allows for reducing the QMDD size by changing the variable order and, hence, eventually allows for the consideration of quantum functionality even on a larger scale. An experimental evaluation confirms the applicability of QMDDs as an efficient and compact representation of quantum functionality to be applied for various purposes in this domain.

The remainder of this paper is structured along these contributions. After a brief review on the background of reversible and quantum logic in Section II, the new definition of QMDDs is provided in Section III. Section IV covers the discussion on efficient construction and manipulation of the resulting QMDDs, while Section V deals with the implementation and opportunities of local modifications. Finally, Section VI summarizes our experimental evaluations and Section VII provides our conclusions.

II. Background

To make this paper self-contained, this section reviews the basics of reversible and quantum functions with a particular emphasis on matrix representations—the major representation of quantum functionality thus far. The respective descriptions are kept brief; readers wishing an in-depth introduction are referred to the respective literature such as [1].

A. Reversible Logic

First, we consider the preliminaries on reversible functions.

Definition 1. *A Boolean function is a mapping* $f : \mathbb{B}^n \to \mathbb{B}$ *with* $n \in \mathbb{N}$. *A function* f *is defined over its primary input variables* $X = \{x_1, x_2, \ldots, x_n\}$ *and hence is also denoted by* $f(x_1, x_2, \ldots, x_n)$.

Definition 2. *A multi-output Boolean function is a mapping* $f : \mathbb{B}^n \to \mathbb{B}^m$ *with* n, $m \in \mathbb{N}$. *More precisely, it is a system of Boolean functions* $f_i(x_1, x_2, \ldots, x_n)$ *with* $1 \leq i \leq m$. *The respective functions* f_i *are also denoted as primary outputs.*

Multi-output functions can also be denoted as n-input, m-output functions, or $n \times m$ functions. The mapping defining a Boolean function or multi-output Boolean function can be described by a truth table, as a Boolean expression, or as a set of Boolean expressions.

Reversible Boolean functions are a subset of multi-output Boolean functions defined as follows.

Definition 3. *A multi-output function* $f : \mathbb{B}^n \to \mathbb{B}^m$ *is reversible iff:*

1) *its number of inputs is equal to the number of outputs (i.e.,* $n = m$*);*
2) *it maps each input pattern to a unique output pattern.*

In other words, a reversible Boolean function is a bijection that performs a permutation of the set of input patterns.

Definition 4. *A Boolean function that is not reversible is termed irreversible.*

Example 1. *Table I(a) shows the truth table of a three-input, two-output function representing a one-bit adder. This function is irreversible, since* $n \neq m$. *The function in Table I(b) is also irreversible since, although the number* n *of inputs is equal to the number* m *of outputs, the mapping is not a permutation; e.g., both inputs* 000 *and* 001 *map to the output* 000. *In contrast, the* 3×3 *function shown in Table I(c) is reversible, since each input pattern maps to a unique output pattern.*

Besides truth tables, permutation matrices are a common representation for reversible Boolean functions.

Definition 5. *A reversible Boolean function with* n *variables describes a permutation* σ *of the set* $\{0, \ldots, 2^n - 1\}$. *This permutation can also be described using a permutation matrix, i.e., a* $2^n \times 2^n$ *matrix* $\mathbf{P} = [p_{i,j}]_{2^n \times 2^n}$ *with* $p_{i,j} = 1$ *if* $i = \sigma(j)$ *and* 0 *otherwise, for all* $i, j = 0, \ldots, 2^n - 1$.

TABLE I
SAMPLE BOOLEAN FUNCTIONS

(a) One-bit adder

x_1	x_2	x_3	f_1	f_2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(b) Irreversible function

x_1	x_2	x_3	f_1	f_2	f_3
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

(c) Reversible function

x_1	x_2	x_3	f_1	f_2	f_3
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	0	0	1
1	0	1	0	0	0
1	1	0	1	1	0
1	1	1	1	1	1

Each column (row) of the matrix represents one possible input pattern (output pattern) of the function. If $p_{i,j} = 1$, then the input pattern corresponding to column j maps to the output pattern corresponding to row i.

Example 2. *The reversible Boolean function defined by the truth table from Table I(c) is also represented by the permutation matrix as shown in Fig. 1.*

The above concepts can readily be extended from the Boolean domain to the multiple-valued case. Here, variables can take values in $\{0, \ldots, r-1\}$, where r is called the radix. Then, a truth table for a multiple-valued function over n variables has r^n rows. The concept of reversibility remains the same, i.e., a multiple-output multiple-valued function is reversible if it has the same number of inputs and outputs and each input pattern maps to a unique output pattern. Such a function can be represented by an $r^n \times r^n$ permutation matrix. The entries of the matrix are 0's and 1's since they denote correspondence in the mapping and not logical values.

B. Quantum Logic

Next we consider the preliminaries on quantum logic. Again, we begin with the two-valued case. Quantum operations manipulate qubits rather than bits. A qubit can represent 0 or 1 as well as superpositions of the two. More formally,

Definition 6. *A qubit is a two-level quantum system, described by a two-dimensional complex Hilbert space. The two orthogonal quantum states $|0\rangle \equiv \binom{1}{0}$ and $|1\rangle \equiv \binom{0}{1}$ are used to represent the Boolean values 0 and 1. The state of a qubit may be written as $|x\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers with $|\alpha|^2 + |\beta|^2 = 1$.*

The quantum state of a single qubit is denoted by the vector $\binom{\alpha}{\beta}$. The state of a quantum system with $n > 1$ qubits is given by an element of the tensor product of the respective state spaces and can be represented as a normalized vector of length 2^n, called the state vector.

According to the postulates of quantum mechanics, the evolution of a quantum system can be described by a series of transformation operations satisfying the following.

Definition 7. *A quantum operation over n qubits can be represented by a unitary matrix, i.e., a $2^n \times 2^n$ matrix $\mathbf{U} = [u_{i,j}]_{2^n \times 2^n}$ with:*

1) *each entry $u_{i,j}$ assuming a complex value;*
2) *the inverse $\mathbf{U}^{-1}$ of $\mathbf{U}$ being the conjugate transpose matrix (adjoint matrix) $\mathbf{U}^\dagger$ of $\mathbf{U}$ (i.e., $\mathbf{U}^{-1} = \mathbf{U}^\dagger$).*

Every quantum operation is reversible since the matrix defining any quantum operation is invertible. At the end of the computation, a qubit can be measured causing it to collapse to a basis state. Then, depending on the current state of the qubit, either a 0 (with probability of $|\alpha|^2$) or a 1 (with probability of $|\beta|^2$) results. The state of the qubit is destroyed by the act of measuring it.

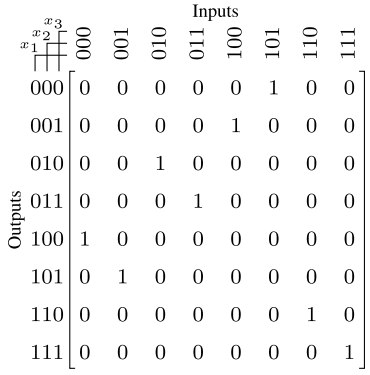


Fig. 1. Matrix representation of the reversible function from Table I(c).

Example 3. *Consider the quantum operation H defined by the unitary matrix $\mathbf{H} = (1/\sqrt{2})\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ which is the well-known Hadamard operation [1]. Applying H to the input state $|x\rangle = \binom{1}{0}$, i.e., computing $\mathbf{H} \times |x\rangle$ yields a new quantum state $|x'\rangle = (1/\sqrt{2})\binom{1}{1}$. For $|x'\rangle$, $\alpha = \beta = 1/\sqrt{2}$. Measuring this qubit would either lead to a Boolean 0 or a Boolean 1 with a probability of $|1/\sqrt{2}|^2 = 0.5$ each. This computation represents one of the simplest quantum computers—a single-qubit random number generator.*

Complex quantum operations are usually realized by a set of elementary quantum operations (e.g., represented in terms of gates) that are performed in a predetermined serial fashion. On the matrix level, a composition of gates (e.g., in terms of a quantum circuit) can be expressed by a direct matrix multiplication of the corresponding gate matrices. This is addressed in detail later in Section IV. Alternatively, this process can be viewed as the implementation of a quantum algorithm in which a series of low-level quantum operations or quantum computational instructions are represented as individual transformation (i.e., gate) matrices.

The above concepts can readily be extended to the multiple-valued case of so-called qudits where there are r states $|0\rangle, \ldots, |r-1\rangle$ and the state of a qudit can be written as $|x\rangle = \Sigma_{i=0}^{r-1} \alpha_i |i\rangle$ with $\Sigma_{i=0}^{r-1} |\alpha_i|^2 = 1$. The transformations in this case are described by $r^n \times r^n$ unitary matrices. One example of a multiple-valued qudit implementation is the use of an ion with more than two energy levels that represent basis states.

III. QUANTUM MULTIPLE-VALUED DECISION DIAGRAMS

In this section, we introduce QMDD which can represent both binary and multiple-valued quantum functionality in a compact and efficient manner. Note that this also includes reversible functions, since transformation matrices for the reversible case are permutation matrices which are a special case of the unitary transformation matrices considered for the quantum case.

We first intuitively motivate basic concepts of the QMDD structure. Afterwards, we provide a formal definition and argue

why QMDDs are not only a compact, but also a canonic representation of quantum functionality.

A. Basic Concepts

As shown in the previous section, quantum operations are commonly represented by unitary transformation matrices, i.e., complex-valued $r^n \times r^n$ matrices where r is the radix and n is the number of variables (inputs and outputs). These matrices exponentially grow in size which is why conventional matrix representation and manipulation techniques are applicable for rather small instances only. However, two observations can be made which build the basis for a compact representation.

Observation 1. *Elementary quantum operations typically affect only a small number of the qubits of a quantum system. The transformation matrix for the whole system, which is the Kronecker product of the respective (smaller) operation matrix and identity matrices, often contains the same pattern repeatedly throughout the matrix. These similar structures which may be equal or equal up to a constant multiplier can be exploited in reducing the representation of a matrix.*

Observation 2. *Transformation matrices are often sparse with many zero entries frequently appearing in blocks. Blocks of zeros can be represented very compactly leading to operation efficiencies particularly in matrix multiplication which is central to dealing with reversible and quantum logic.*

To make use of these possible reductions, the fundamental idea of QMDDs is to use a partitioning of the original matrix into sub-matrices which in turn are partitioned in the same manner. These decomposition steps are represented by vertices eventually forming a decision diagram. Then, the redundancies following from the observations above can be avoided by using shared structures. More precisely, we observe—starting with $r = 2$—that a $2^n \times 2^n$ matrix can be partitioned into 4 sub-matrices of dimension $2^{n-1} \times 2^{n-1}$ as follows:

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}_{00} & \mathbf{U}_{01} \\ \mathbf{U}_{10} & \mathbf{U}_{11} \end{bmatrix}. \tag{1}$$

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

Example 4. *Consider again the matrix shown in Fig. 1. 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. For instance, $\mathbf{U}_{10}$ corresponds to the top-right sub-matrix which describes the mapping of the remaining variables when x_1 is mapped from input value 1 to output value 0. Using this partition, a matrix can be represented as a graph with a vertex as shown in Fig. 2. The vertex is labeled by the variable associated with the partition and has directional edges pointing to vertices corresponding to the sub-matrices.*

The partitioning process can be applied recursively 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 tree. By traversing the tree, one can access

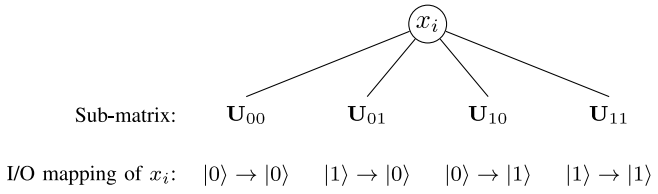


Fig. 2. Vertex representing the matrix partitioning (for $r = 2$).

the successively partitioned sub-matrices of the original matrix down to the individual elements.

The partitioning in (1) can readily be extended to the multiple-valued case as follows:

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}_{00} & \mathbf{U}_{01} & \cdots & \mathbf{U}_{0,r-1} \\ \mathbf{U}_{10} & \mathbf{U}_{11} & \cdots & \mathbf{U}_{1,r-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{U}_{r-1,0} & \mathbf{U}_{r-1,1} & \cdots & \mathbf{U}_{r-1,r-1} \end{bmatrix} \tag{2}$$

where matrix $\mathbf{U}$ has dimension $r^n \times r^n$ and the r^2 sub-matrices each have dimension $r^{n-1} \times r^{n-1}$.

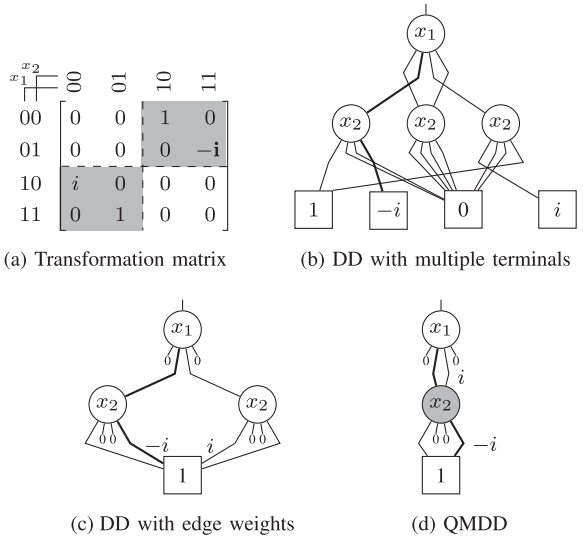
This structure allows for representing equal sub-matrices by shared parts of the diagram. However, further vertex sharing and, thus, reduction is possible by extracting common multipliers as illustrated in the following.

Example 5. *Consider the matrix in Fig. 3(a). Applying the recursive partitioning above would yield a tree as depicted in Fig. 3(b): with a root labeled x_1, three internal vertices labeled x_2 (the two zero blocks sharing the same vertex), and four terminal vertices (one for each value $0, 1, i, -i$). By extracting common multipliers and using them as edge weights, we can reduce the tree to the graph in Fig. 3(c). Then, the four terminal vertices in the tree can be collapsed to a single terminal vertex with value 1. The actual value of a matrix entry is the product of the weights on the path corresponding to the sub-matrix partitioning through the matrix that leads to the entry. For example, the highlighted matrix entry $-i$ in Fig. 3(a) corresponds to the path highlighted in bold in Fig. 3(c). Since we are taking the product of weights, edges with 0 weight can point directly to the terminal vertex. To avoid clutter in the diagrams we show 0 weight edges as stubs and do extend them to the terminal. Also, to avoid clutter, we do not explicitly indicate the edge weight if it is 1.*

However, even more reduction is possible: the matrix in Fig. 3(a) has two structurally equivalent sub-matrices (highlighted in gray) which differ only by a common multiplier. These correspond to the two vertices labeled x_2 in Fig. 3(c). By factoring out i from the lower left sub-matrix as a weight, we can represent the two sub-matrices by one shared structure as shown in Fig. 3(d).

Clearly, the choice of the edge weights is not unique, though it is central for our objective to share common matrix structures and to efficiently deal with sub-matrices entirely composed of 0's. To this end, we introduce the concept normalizing a (sub)matrix.

1) To achieve an optimal structure sharing, we want to store only normalized forms of the sub-matrices. For this purpose, the normalized form $\widehat{\mathbf{M}}$ of a matrix $\mathbf{M}$



(a) Transformation matrix (b) DD with multiple terminals

(c) DD with edge weights (d) QMDD

Fig. 3. Representations of a two-qubit quantum operation.

shall be the same for any (nonzero) multiple of $\mathbf{M}$, that is

$$\widehat{\mathbf{M}} = \widehat{\alpha\mathbf{M}} \text{ for all } \alpha \neq 0. \tag{3}$$

2) Moreover, as the terminal vertex represents the matrix $[1]_{1\times 1}$, this matrix shall be the normalized form of any 1×1 matrix, that is

$$\widehat{[\alpha]}_{1\times 1} = [1]_{1\times 1} \text{ for all } \alpha \in \mathbb{C}. \tag{4}$$

To obtain normalized forms we need a normalization scheme to determine which common multiplier $N(\mathbf{M})$ (normalization factor) is extracted from a matrix $\mathbf{M}$, such that

$$\mathbf{M} = N(\mathbf{M}) \cdot \widehat{\mathbf{M}}. \tag{5}$$

Formally, interpreting the normalization scheme as a mapping N from the set of matrices to the set of normalization factors (complex numbers), we require the following properties.

1) $N(\alpha\mathbf{M}) = \alpha N(\mathbf{M})$ for any complex-valued matrix $\mathbf{M}$ and any complex-valued number α.
2) $N([\alpha]_{1\times 1}) = \alpha$ for any complex number α.
3) $N(\mathbf{M}) = 0 \Leftrightarrow$ all entries in $\mathbf{M}$ are zero.

Example 6. *A very simple normalization scheme is obtained by defining the normalization factor of a matrix to be (1) the first nonzero entry of the matrix that is found when scanning the matrix row by row and entry by entry or (2) zero if no nonzero entry is found. It is readily observed that this scheme indeed satisfies the required properties.*

Note that property 3) allows us to directly compute the normalized form of nonzero matrices from (5) as follows:

$$\widehat{\mathbf{M}} = \frac{1}{N(\mathbf{M})} \cdot \mathbf{M} \quad (\mathbf{M} \neq 0) \tag{6}$$

and one can easily check that this definition satisfies the desired properties of normalized forms given in (3) and (4). For $\mathbf{M} = [0]_{k\times k}$, we could choose the normalized form arbitrarily according to (5), but for consistency with $k = 1$ we set

$$\widehat{[0]}_{k\times k} = [1]_{k\times k} \text{ for any } k. \tag{7}$$

Recall that our aim was to identify structurally equivalent sub-matrices and extract common multipliers in order to obtain as much structure sharing as possible by using normalized forms of the sub-matrices. For a formal description of how the above normalization is integrated into the decomposition process, we make use of the *Khatri–Rao* (KR) product (introduced in [17] and [18]). The KR product provides for a mathematical description of the QMDD vertex decomposition in analogy to the Shannon decomposition for BDDs and other BDD-like structures such as the QuIDD and QDD. It is defined as follows.

Definition 8. *The KR product $*$ is a particular type of matrix multiplication that operates over matrix partitions and can be described in terms of the tensor product, denoted by $\otimes$, as $\mathbf{A} * \mathbf{B} = [\mathbf{A}_{ij} \otimes \mathbf{B}_{ij}]_{ij}$.*

Example 7. *Assume two matrices $\mathbf{A}$ and $\mathbf{B}$ are of the form*

$$\mathbf{A} = \left[\begin{array}{c|c} \mathbf{A}_{00} & \mathbf{A}_{01} \\ \hline \mathbf{A}_{10} & \mathbf{A}_{11} \end{array}\right]_{2k\times 2k} \quad \mathbf{B} = \left[\begin{array}{c|c} \mathbf{B}_{00} & \mathbf{B}_{01} \\ \hline \mathbf{B}_{10} & \mathbf{B}_{11} \end{array}\right]_{2l\times 2l}.$$

Then, the KR product $\mathbf{A} * \mathbf{B}$ becomes

$$\mathbf{A} * \mathbf{B} = \left[\begin{array}{c|c} \mathbf{A}_{00} \otimes \mathbf{B}_{00} & \mathbf{A}_{01} \otimes \mathbf{B}_{01} \\ \hline \mathbf{A}_{10} \otimes \mathbf{B}_{10} & \mathbf{A}_{11} \otimes \mathbf{B}_{11} \end{array}\right]_{2(k\cdot l)\times 2(k\cdot l)}.$$

Note that, according to the definition of the tensor product, the block $\mathbf{A}_{ij} \otimes \mathbf{B}_{ij}$ has dimension $k \cdot l \times k \cdot l$ assuming that $\mathbf{A}_{ij}$ and $\mathbf{B}_{ij}$ have dimensions $k \times k$ and $l \times l$, respectively.

Conceptually, the proposed normalization from (5) is to be applied to each sub-matrix separately in the course of the partitioning and the extracted normalization factors (edge weights) are to be applied only to the particular sub-matrix. This can be formally described using the KR matrix product by the following decomposition relationship:

$$\mathbf{M} = \mathbf{W}(\mathbf{M}) * \widehat{\mathbf{M}}. \tag{8}$$

In this central equation.

1) $\mathbf{M}$ is the original $r^n \times r^n$ matrix to be decomposed.
2) $\mathbf{W}(\mathbf{M}) = [N(\mathbf{M}_{ij})]_{0\leq i,j<r}$ is the $r \times r$ matrix containing all scalar factors (weights) which are extracted from the sub-matrices $\mathbf{M}_{ij}$.
3) $\widehat{\mathbf{M}}$ has dimension $r^n \times r^n$, partitioned into the r^2 normalized sub-matrices $\widehat{\mathbf{M}}_{ij}$ of dimension $r^{n-1} \times r^{n-1}$.

Here, the KR product essentially ensures that the extracted weights will be associated with the appropriate sub-matrix.

Example 8. *For $r = 2$, the decomposition is given by the equation*

$$\mathbf{U} = \left[\begin{array}{c|c} N(\mathbf{U}_{00}) & N(\mathbf{U}_{01}) \\ \hline N(\mathbf{U}_{10}) & N(\mathbf{U}_{11}) \end{array}\right] * \left[\begin{array}{c|c} \widehat{\mathbf{U}}_{00} & \widehat{\mathbf{U}}_{01} \\ \hline \widehat{\mathbf{U}}_{10} & \widehat{\mathbf{U}}_{11} \end{array}\right].$$

The decomposition is represented by a vertex labeled by the corresponding partition variable and, as shown in Fig. 3(c), the extracted normalization factors are attached to the edges that point to the vertices representing the decomposition of the corresponding sub-matrices. Note, however, that the diagram

in Fig. 3(c) does not correspond to a proper decomposition since the same weight is extracted for the two nonzero sub-matrices though they differ by a constant multiplier only. Conversely, the diagram in Fig. 3(d) results from a proper decomposition using normalized forms as they would result from using the normalization scheme outlined in Example 6.

B. Formal Definition

Based on the concepts described above, we now present a formal definition of QMDDs.

Definition 9. *A QMDD is a representation of an $r^n \times r^n$ complex matrix as a rooted directed acyclic graph with a set V containing two types of vertices: a single terminal vertex and zero or more non-terminal vertices. The terminal vertex, labeled 1, represents the matrix $[1]_{1\times 1}$. It has no outgoing edges. Each non-terminal vertex is labeled by an r-valued variable and has r^2 outgoing edges, each pointing to a vertex in V. Each non-terminal vertex denotes the partitioning of a matrix by the application of (8) and, thus, has r^2 outgoing edges e_p, $0 \leq p < r^2$, which are labeled $00, 01, \ldots, r-1\ r-1$ and have associated complex weights $w(e_{00}), w(e_{01}), \ldots, w(e_{r-1,r-1})$. There is an initial edge with no source vertex which points to the root vertex and has an associated complex weight representing the normalization factor of the represented matrix according to (5). We term this the root edge.*

A QMDD has the following properties.

1) *The non-terminal vertex labels (variables) are ordered which means that if a non-terminal vertex with label x_i has an edge pointing to a non-terminal vertex labeled x_j, x_i is the more significant variable in the row and column labeling of the matrix represented by the QMDD.*
2) *A QMDD is reduced which means: a) there is no non-terminal vertex where all outgoing edges point to the same vertex and have the same weight and b) all vertices are unique and no two non-terminal vertices represent the same sub-matrix, i.e., are labeled by the same variable and have all corresponding edges pointing to the same vertex with the same weight.*
3) *Any constant (sub)matrix, regardless of its size, is represented as an edge pointing directly to the terminal vertex.*

It is important to note that a QMDD is a recursive representation as every edge in the QMDD can be seen as the root edge of the QMDD representation of a sub-matrix. This is a key observation used to describe how matrix operations are implemented with QMDDs.

Another key factor in interpreting QMDDs concerns the variables labeling the non-terminal vertices. Each non-terminal vertex is labeled by an r-valued variable and that variable labels both the rows and columns of the matrix. The corresponding matrix partitioning divides the rows into r sections and the columns into r sections for a total of r^2 sub-matrices [see (8)].

The notion of variable ordering introduced in Definition 9 means that if the matrix row and column variables are ordered by a function *index*() such that $index(x_i) < index(x_j)$ iff x_i precedes x_j, then the QMDD satisfies the following two properties.

1) Each variable appears at most once on each path from the root vertex to the terminal vertex.
2) An edge from a non-terminal vertex labeled x_i points to a non-terminal vertex labeled x_j, $index(x_j) > index(x_i)$ or to the terminal vertex. Hence, the variable indices along any path from the root to the terminal satisfy the order imposed by *index*() and that order corresponds to the variables order for the matrix and column labeling from most to least significant.

Another important observation is that all edges with weight 0 point directly to the terminal vertex. This is because they represent the normalization factor 0 which, by definition, may only occur if the edge represents a sub-matrix $[0]_{k\times k}$. Consequently, the edge points to a vertex representing the normalized form, which is $[1]_{k\times k}$ by (7). However, since the QMDD is reduced, there may not be a non-terminal vertex representing this matrix, because all its outgoing edges would point to the same vertex with the same weight. Similarly, any constant sub-matrix is represented by an edge directly pointing to the terminal vertex with the appropriate weight, as stated in property 3) of Definition 9. In summary, these structural sharing allow for a compact representation of the corresponding reversible or quantum functionality.

C. Canonicity

In addition to the desired feature of allowing for a compact representation of functionality, the uniqueness of a function representation is also of interest. In particular, this is of significant importance for many application areas such as equivalence checking. QMDDs satisfy this criteria, i.e., any normalized QMDD is canonic with respect to a given variable order and normalization scheme as is proven in this section.

Theorem 1. *Given a normalization scheme N, the corresponding QMDD representation of any complex-valued $r^n \times r^n$ matrix is unique up to variable order.*

Proof: The proof is by contradiction, i.e., we assume that there exists a matrix M that has two different QMDD representations G_1 and G_2 which adhere to the same normalization scheme and variable order, and show that such a matrix may not exist. Recall that, according to the decomposition in (8), any non-terminal vertex of a QMDD represents the normalized form of the corresponding sub-matrix. Consequently, each vertex in G_1 has an equivalent in G_2 and vice versa. Roughly speaking, both representations employ the same set of vertices. Since G_1 and G_2 are different, one representation must include an edge e_1 that is not present in the other representation. This edge must have a different weight or a different target compared to the corresponding edge e_2 in the other representation. However, both cases may not happen as e_1 and e_2 have the same source and, thus, represent the same sub-matrix for which the normalization factor (edge weight) and normalized form (target vertex) are uniquely defined. This contradicts the assumption that there are two representations for matrix M. ■

Clearly, QMDD representations may be different for different variable orders. Moreover, even for a fixed order,

the resulting QMDDs may differ for different normalization schemes as well. However, the following theorem shows that the resulting diagram structure is always the same.

Theorem 2. *Given a complex-valued $r^n \times r^n$ matrix and a fixed variable order, the corresponding QMDD representations are isomorphic for any two normalization schemes.*

Proof: Consider two QMDD representations of the same matrix corresponding to different normalization schemes. Again, the non-terminal vertices in both representations represent normalized forms of the corresponding sub-matrices. Since QMDDs are reduced, a single vertex exists for each of a sub-matrix and its multiples. Consequently, there is a bijection between the vertex sets of both representations that identifies vertices representing the same class of multiples of a sub-matrix. We want to show that this map is indeed a graph isomorphism, i.e., it also preserves edges. To do that, we have to show that each pair of corresponding edges indeed point to corresponding vertices. For this purpose, consider an arbitrary pair of corresponding edges, i.e., two outgoing edges e_{ij} and e'_{ij} (both labeled *ij*) from corresponding vertices from both representations. Since the source vertices correspond to the same class of multiples of a sub-matrix M, both vertices represent a multiple of M: its respective normalized form. Consequently, both edges represent multiples of the same sub-matrix M_{ij} of M and, thus, point to corresponding vertices (representing the respective normalized forms of M_{ij}). Overall, the bijection between the vertex sets preserves edges and is, therefore, a graph isomorphism. ■

The two theorems proven in this section indeed show that QMDDs provide unique representations (up to variable order) while the resulting structure does not depend on which normalization scheme is actually used. This is beneficial since it ensures the highest possible structure sharing and, hence, the most compact representation regardless of how exactly the weights are determined. Thus, even simple normalization schemes as the one discussed in Example 6 are sufficient. This is even more important as it is infeasible for large matrices to compute sophisticated normalization factors in a top-down fashion and, as a consequence, normalization of QMDDs is practically performed in a bottom-up fashion as we will see in the following section.

IV. Construction and Manipulation of QMDDs

Thus far, we showed that the proposed QMDDs provide a compact and canonic representation of arbitrary reversible and quantum functionality. However, to be of practical use, QMDDs must additionally allow for an efficient construction and manipulation. These issues are covered in this section. More precisely, we will discuss how essential matrix operations can efficiently be performed on QMDDs and show that QMDD representations for elementary quantum operations can easily be derived. These issues are exemplarily illustrated by means of constructing a QMDD for a given quantum circuit.

A. Normalization

Before we describe how essential matrix operations can efficiently be performed directly on the QMDD structure, we first need to consider how normalization, the key for QMDDs being canonic representations, is achieved in practice. For larger matrices, it is infeasible to determine the edge weights, i.e., the normalization factors as they arise from the partitioning of a matrix as given in (8), in a top-down fashion. As the QMDD is rather built bottom-up, the edge weights also have to be determined this way. More precisely, by performing vertex normalization of each non-terminal vertex when it is constructed, edge weights are determined subsequently, finally making the QMDD a canonic representation.

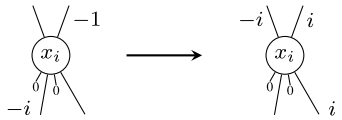


Fig. 4. Normalizing a vertex.

Consider a QMDD non-terminal vertex v with outgoing edges e_p, $0 \leq p \leq r^n - 1$ that are pointing to sub-matrices $\mathbf{U}_{00}, \mathbf{U}_{01}, \ldots, \mathbf{U}_{r-1,r-1}$, respectively, and let $w(e_p)$ denote the weight on edge e_p.

Definition 10. *The non-terminal vertex v is normalized if $w(e_j) = 1$ for the lowest j for which $w(e_j) \neq 0$.*

It is straightforward to normalize a given QMDD non-terminal vertex v according to the above definition: a single normalization factor equal to the $w(e_j)$ for the lowest j for which $w(e_j) \neq 0$ is identified and the weights on all edges leading from the vertex are divided by that factor. Note that the existence of at least one edge with nonzero weight is for sure as we have shown earlier that edges with weight 0 point directly to the terminal vertex and, since the QMDD is reduced, there may not be a vertex with all edges pointing to the same vertex with the same weight.

The vertices that v points to are not affected, but the edges pointing to v have to be adjusted by multiplying their weights by the normalization factor. An example for the binary case is shown in Fig. 4. The vertex on the left is normalized as shown on the right. In this case, the normalization factor is $-i$.

Note that, as QMDDs are built and normalized bottom-up, this propagation of normalization factors to the top can easily be performed without possibly destroying the normalization of existing vertices. Finally, note that this procedure indeed establishes a normalization scheme similar to the one defined in Example 6, though now the matrix is scanned for nonzero entries in a more elaborate fashion.

Example 9. *For a 4×4 matrix, the order in which the matrix is being scanned for nonzero entries is given by*

$$\left[\begin{array}{cc|cc} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \\ \hline 9 & 10 & 13 & 14 \\ 11 & 12 & 15 & 16 \end{array}\right]$$

for vertex normalization. In contrast, we obtain

$$\left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}\right]$$

for the normalization scheme from Example 6.

B. Matrix Operations

Knowing how a normalized representation is actually achieved, we can now describe how matrix operations can be computed directly on a QMDD. We will do this for the three common operations addition, direct multiplication, and tensor or Kronecker multiplication. It is noted that the calculation of the tensor product and the Kronecker product are identical multiplicative operations over the class of unitary matrices used to represent quantum system evolutions. Other operations are readily implemented using the methods described in the following text.

Implementing these operations uses the fact that a QMDD is a recursive representation that represents a matrix as a composition of sub-matrices which are in turn represented by smaller sub-matrices. This allows the operations to be expressed in terms of operations on sub-matrices. For example, matrix addition for the binary case can be expressed as the addition of sub-matrices, that is

$$\begin{bmatrix} \mathbf{A}_{00} & \mathbf{A}_{01} \\ \mathbf{A}_{10} & \mathbf{A}_{11} \end{bmatrix} + \begin{bmatrix} \mathbf{B}_{00} & \mathbf{B}_{01} \\ \mathbf{B}_{10} & \mathbf{B}_{11} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{00} + \mathbf{B}_{00} & \mathbf{A}_{01} + \mathbf{B}_{01} \\ \mathbf{A}_{10} + \mathbf{B}_{10} & \mathbf{A}_{11} + \mathbf{B}_{11} \end{bmatrix}.$$

Matrix multiplication for the binary case is expressible as follows:

$$\begin{bmatrix} \mathbf{A}_{00} & \mathbf{A}_{01} \\ \mathbf{A}_{10} & \mathbf{A}_{11} \end{bmatrix} \begin{bmatrix} \mathbf{B}_{00} & \mathbf{B}_{01} \\ \mathbf{B}_{10} & \mathbf{B}_{11} \end{bmatrix} = \begin{bmatrix} \mathbf{C}_{00} & \mathbf{C}_{01} \\ \mathbf{C}_{10} & \mathbf{C}_{11} \end{bmatrix}$$

with

$$\begin{aligned} \mathbf{C}_{00} &= \mathbf{A}_{00}\mathbf{B}_{00} + \mathbf{A}_{01}\mathbf{B}_{10} \\ \mathbf{C}_{01} &= \mathbf{A}_{00}\mathbf{B}_{01} + \mathbf{A}_{01}\mathbf{B}_{11} \\ \mathbf{C}_{10} &= \mathbf{A}_{10}\mathbf{B}_{00} + \mathbf{A}_{11}\mathbf{B}_{10} \\ \mathbf{C}_{11} &= \mathbf{A}_{10}\mathbf{B}_{01} + \mathbf{A}_{11}\mathbf{B}_{11}. \end{aligned}$$

In order to formalize these operations on the QMDD, the following definitions are applied.

Definition 11. *Given an edge e, we use $w(e)$ to denote the weight on e, $v(e)$ to denote the vertex e points to, $x(e)$ to denote the variable that labels the vertex e points to (not defined for the terminal vertex), $E_i(e)$ to denote the ith edge out of the vertex that e points to, and $T(e)$ to denote a Boolean test that is true if e points to the terminal vertex.*

Furthermore, we assume that the variables adhere to the same order in all considered QMDDs and we shall use $\prec$ to denote the fact that one variable precedes another and, hence, appears closer to the terminal vertex in the QMDD. For generality, we consider the multiple-valued case where r is the radix, i.e., every non-terminal vertex has r^2 outgoing edges (for the Boolean case, r can simply be set to $r = 2$).

Having that, matrix operations can be conducted on QMDDs as follows noting that a matrix is uniquely identified by the root edge for the QMDD.

1) Matrix Addition: Let e_0 and e_1 be the root edges of two QMDDs (matrices) to be added. The procedure is recursive and involves the following steps.

a) If $T(e_1)$, swap e_0 and e_1.
b) If $T(e_0)$:
   i) if $w(e_0) = 0$, the result is e_1;
   ii) if $T(e_1)$, the result is an edge pointing to the terminal vertex with weight $w(e_0) + w(e_1)$.
c) If $x(e_0) \prec x(e_1)$, swap e_0 and e_1.
d) For $i = 0, 1, \ldots, r^2 - 1$.
   i) Create an edge p pointing to $v(E_i(e_0))$ with weight $w(e_0) \times w(E_i(e_0))$.
   ii) If $x(e_0) = x(e_1)$, create an edge q pointing to $v(E_i(e_1))$ with weight $w(e_1) \times w(E_i(e_1))$, else set $q = e_1$.
   iii) Recursively invoke this procedure to add p and q giving z_i.
e) The result is an edge pointing to a vertex labeled $x(e_0)$ with outgoing edges $z_i, i = 0, 1, \ldots, r^2 - 1$. This vertex and the edge pointing to it are normalized.

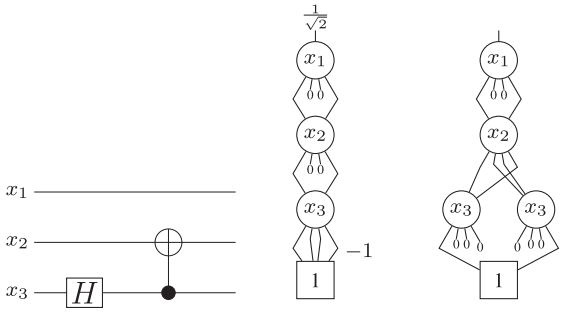
2) Matrix Multiplication: Let e_0 and e_1 be the root edges of two QMDD (matrices) to be multiplied. The procedure is recursive and involves the following steps.

a) If $T(e_1)$, swap e_0 and e_1.
b) If $T(e_0)$ then:
   i) if $w(e_0) = 0$, the result is e_0;
   ii) if $w(e_0) = 1$, the result is e_1;
   iii) otherwise, the result is an edge pointing to $v(e_1)$ with weight $w(e_0) \times w(e_1)$.
c) If $x(e_0) \prec x(e_1)$, swap e_0 and e_1.
d) For $i = 0, r, 2r, \ldots, (r-1)r$.
   For $j = 0, 1, \ldots, r - 1$.
   Set z_{i+j} to be an edge with weight 0 pointing to the terminal vertex.
   For $k = 0, 1, \ldots, r - 1$.
   i) Create an edge p pointing to $v(E_{i+k}(e_0))$ with weight $w(e_0) \times w(E_{i+k}(e_0))$.
   ii) If $x(e_0) = x(e_1)$, create an edge q pointing to $v(E_{j+r\times k}(e_1))$ with weight $w(e_1) \times w(E_{j+r\times k}(e_1))$, else set $q = e_1$.
   iii) Recursively invoke this procedure to multiply the QMDD pointed to by p and q and then use the procedure above to add the result to the QMDD pointed to by z_{i+j}. The result of the addition replaces z_{i+j}.
e) The result is an edge pointing to a vertex labeled $x(e_0)$ with outgoing edges $z_i, i = 0, 1, \ldots, r^2 - 1$. This vertex and the edge pointing to it are normalized.

3) Kronecker Product: Let e_0 and e_1 be the root edges of two QMDD (matrices) for which we want to compute the Kronecker product $\mathbf{A} \otimes \mathbf{B}$ (note that this operation is not commutative). For the application considered here, the selection variables for $\mathbf{B}$ precede the selection variables for $\mathbf{A}$. This greatly reduces the complexity of the algorithm for computing the Kronecker product of two QMDD.

The procedure is recursive and involves the following steps.

a) If $T(e_0)$ then:
   i) if $w(e_0) = 0$, the result is e_0;
   ii) if $w(e_0) = 1$, the result is e_1;
   iii) otherwise, the result is an edge pointing to $v(e_1)$ with weight $w(e_0) \times w(e_1)$.



(a) Quantum Circuit (b) First gate (c) Second gate

Fig. 5. Quantum circuit built from elementary quantum gates.

b) For $i = 0, 1, \ldots, r^2 - 1$.
Recursively invoke this procedure to find the Kronecker product of $E_i(e_0)$ and e_1 setting z_i to the result.

c) The result is an edge pointing to a vertex labeled $x(e_0)$ with outgoing edges $z_i, i = 0, 1, \ldots, r^2 - 1$. This vertex and the edge pointing to it are normalized.

Performing these matrix operations directly on the QMDD structure makes them very effective which we will exemplarily illustrate in the next section by means of constructing a QMDD for a given quantum circuit.

C. Construction

For many applications, it is a very important task to efficiently construct a representation of quantum circuits, i.e., cascades of elementary quantum operations that form a more complex operation.

Example 10. *Consider the quantum circuit shown in Fig. 5(a) which realizes a quantum operation on three qubits using two elementary quantum gates. The qubits are represented by horizontal lines and the gates are applied to the qubits successively from left to right. The first gate is a Hadamard gate realizing the Hadamard operation* $\mathbf{H}$ *introduced in Example 3. The gate operates on the target qubit* x_3 *only and does not affect any other qubit. The second gate is a controlled NOT gate. That means it performs the NOT operation—defined by the matrix* $\mathbf{X} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$*, on the target qubit (here:* x_2*) if and only if all the control qubits (here:* x_3*) are in the activating state (here:* $\langle 1|$*). If at least one control is not in the appropriate state, the gate has no effect on the target. Controls and unconnected qubits are not affected in any case. For the Hadamard gate, the resulting matrix is* $\mathbf{I} \otimes \mathbf{I} \otimes \mathbf{H}$*. The controlled NOT is represented by* $\mathbf{I} \otimes C(\mathbf{X})$ *where* $C(\mathbf{X})$ *is the* 4×4 *matrix* $\left(\begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{smallmatrix}\right)$*. The corresponding QMDD representations are shown in Fig. 5(b) and (c), respectively.*

The representation for the whole quantum circuit is successively built from representations for single quantum gates. More precisely, for a cascade or series of gates $G_0 G_1 \ldots G_{t-1}$ where the transformation for gate G_i is defined by matrix $\mathbf{M}_i$, the transformation for the complete circuit is given by the direct matrix product $\mathbf{M}_{t-1} \times \mathbf{M}_{t-2} \times \ldots \times \mathbf{M}_0$. Note that the order of the matrices has to be reversed to achieve the correct order of applying the gates (first G_0, then G_1, etc.).

To construct this matrix product, the QMDDs for the single gates simply have to be multiplied using the algorithm presented in the previous section. Consequently, for the remainder of this section we will focus on how the QMDD representations for elementary quantum gates can be constructed efficiently.

Again for generality, we consider the multiple-valued case. Assume, as above, the variable order $x_1 \succ x_2 \succ \ldots \succ x_n$ from the root vertex toward the terminal vertex. A gate G is specified by the $r \times r$ base transition matrix $\mathbf{M}$, the target qubit x_t and a possible empty set of control qubits C. Though it is possible to construct the QMDD for the gate in one step, for a better understanding we will construct two QMDDs representing the cases that the gate is active (all controls have the required value) or inactive (nothing is done). By adding these QMDDs, the actual QMDD for the gate results.

Example 11. *Consider the QMDD in Fig. 5(b) which represents the first gate of the quantum circuit examined in Example 10. Recall that, in terms of matrix representations, the unconnected qubits correspond to identity matrices that are applied to the Hadamard gate matrix via Kronecker products. On the QMDD level, this is done by connecting vertices labeled* x_1 *and* x_2 *(representing the identity) to the* x_3*-vertex that represents the Hadamard matrix. Note that normalization propagates the common multiplier* $1/\sqrt{2}$ *of this matrix to the root edge.*

The QMDD of the circuit's second gate is shown in Fig. 5(c). The two vertices labeled x_3 *represent the active and inactive part of the gate, respectively. For the left vertex which represents the inactive part, only the edge representing the input/output mapping* $|0\rangle \rightarrow |0\rangle$ *has nonzero weight. Similarly, for the active part this is true for the* $|1\rangle \rightarrow |1\rangle$ *edge. These parts are put together at the* x_2*-level. The inactive part which shall leave* x_2 *untouched is connected to the* $|0\rangle \rightarrow |0\rangle$ *and* $|1\rangle \rightarrow |1\rangle$ *edge. The active part which shall perform the NOT operation, i.e., interchange* $|0\rangle$ *and* $|1\rangle$*, is connected to the* $|0\rangle \rightarrow |1\rangle$ *and* $|1\rangle \rightarrow |0\rangle$ *edge. Finally, the unconnected qubit* x_1 *is connected as for the first gate.*

This procedure for constructing QMDDs for quantum gates from the terminal vertex toward the root vertex can be generalized as follows. It has three phases.

1) *Variables Below Target:* For these variables, r^2 separate QMDD is constructed for the active case. The $(i \times r + j)$th QMDD has a path following the active values, i.e., the values required to activate the controls, for the control variables to the terminal value $\mathbf{M}_{i,j}$ with all other paths leading to 0. Noncontrol variables (unconnected lines) are taken into account through certain identity matrices combined with the active part of the QMDD using Kronecker products.
For the inactive case, only one QMDD has to be constructed since the gate action is the identity matrix which only has nonzero entries on the diagonal which are all equal to 1. The off-diagonal QMDDs are simply edges to the terminal vertex with weight 0. The QMDD for the diagonal is build recursively (bottom-up) starting with an edge to the terminal with weight 1. Noncontrol variables are again taken into account through certain identity matrices combined with the already constructed part of the QMDD using Kronecker products. For control variables, we create a vertex that points: a) to the

already constructed part for the active value; b) to an identity matrix of appropriate dimension for the inactive values; and c) to 0 otherwise.

2) *Target Variable:* A single QMDD is formed with the source vertex labeled by the target variable and the edges from that vertex leading to the QMDDs constructed in the first phase.
3) *Variables Above Target:* For the active case, the upper part of the QMDD has a path following the active values for the control variables with all other paths leading to 0. Again, noncontrol variables are taken into account through appropriate identity matrices and Kronecker products.

   For the inactive case, the variables above the target are treated equally as variables below the target.

Space here only allows us to outline the procedure and a detailed understanding is best derived from the C code implementation available from the authors. However, the most important aspect of this procedure is that it builds the QMDD corresponding to a gate from terminal to root vertex in a single pass by iterating from x_n to x_1 with no backtracking or recursion. It is thus quite efficient and, to a large extent, the time required is independent of the complexity of the gate under consideration.

Overall, by employing matrix multiplication as outlined above, QMDD representations for quantum circuits as well as other quantum logic representations can be build from the gate representations very efficiently.

V. Changing the Variable Order

As we have seen so far, QMDDs offer a compact and canonic representation of reversible and quantum logic and promise an efficient handling and manipulation of quantum functionality. However, it is a common observation for decision diagrams that the variable order may have a large impact on the size of the representation and, hence, is crucial for the overall efficiency. In this section, we will first analyze in which way variable reordering can possibly reduce the vertex count of QMDDs. After that, we illustrate which obstacles arise when performing local modifications (like variable interchanges) on QMDDs, provide a solution to this problem and, finally, present the resulting interchange scheme for adjacent variables in QMDDs. This scheme enables to use many established reordering techniques for decision diagrams that rely on interchanges of adjacent variables like, e.g., sifting or window permutation [15]. An experimental evaluation how variable reordering affects the diagram size will be presented later in Section VI.

A. Shared Vertices and Skipped Variables

Changing the variable order of a QMDD can be interpreted as permuting rows and columns of the corresponding matrix. In fact, this change can have a significant impact on structural equivalence and, hence, on shared vertices. In some cases, it allows the joining of identical blocks which leads to redundant vertices for which all outgoing edges would point to the same vertex with the same weight. However, redundant vertices are, by definition, not allowed in QMDDs and will be represented by an edge that skips the particular variable (say x_j) and points to a vertex which is labeled by a variable that succeeds x_j in the variable order. Both are illustrated by means of the following.

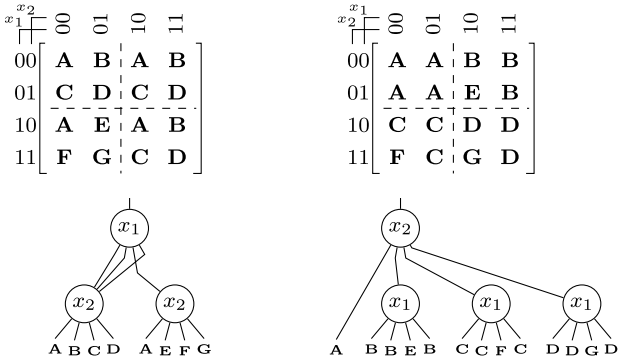


Fig. 6. Variable interchange: structural equivalence and skipped variables.

Example 12. *Consider the matrices in Fig. 6. Suppose* $\mathbf{A}, \ldots, \mathbf{G}$ *are mutually different sub-matrices of the same size* $2^k \times 2^k$. *Both matrices represent the same functionality (though employing a different variable order where* x_1 *and* x_2 *change places) and each matrix can be obtained from the other by variable interchange, i.e., by swapping the* 01 *and* 10 *rows and columns. The matrix on the left has three identical blocks which can be represented by a shared* x_2*-vertex. The matrix on the right does not offer shared vertex compression, but the top-right sub-matrix consists of four identical blocks which gives rise to a skipped variable.*

A special case of skipped variables are blocks of zero which result in a 0-edge that directly points to the terminal vertex and skips all succeeding variables. It can be shown that 0-edges are the only type of skipped variables that can occur for QMDDs which represent a reversible function (permutation matrix) [19]. This is because skipped variables always indicate identical sub-matrices and since in permutation matrices there is a single nonzero entry in each row and column, the identical sub-matrices can only be blocks of zeros.

From this perspective, the aim of QMDD minimization for reversible operations can hence be described as changing the position of the non-terminal vertices such that those vertices that have more outgoing 0-edges are closer to the root vertex. Corresponding metrics for guiding the reordering process, based upon the ratio of the number of nonzero weight edges versus the total number of vertices, are reported in [20].

However, in quantum computation there are also functions whose transformation matrices are completely populated, i.e., which do not contain a single zero entry. For instance, this is the case for *Quantum Fourier Transforms* (QFT) which occur as part of Shor's factorization algorithm [2], [3]. In fact, these illustrate nicely how skipped variables and a high rate of shared vertices can reduce the QMDD size. More precisely, the corresponding QMDD representations do not show any shared vertices in standard variable order as there are no structurally equivalent sub-matrices and, hence, have the maximum QMDD size with respect to the matrix size. However, when applying the inverse variable order as shown in Fig. 7, the QMDD size is reduced significantly, e.g., from 21 to 8 non-terminal vertices for $n = 3$.

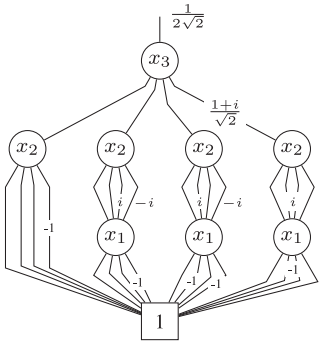
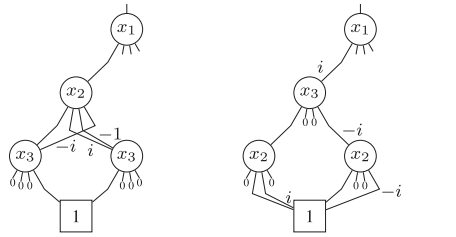


Fig. 7. QFT for $n = 3$ qubits and inverse variable order.



(a) Given QMDD (b) W/ interchanged variables

Fig. 8. Variable interchange in a QMDD.

The existence of skipped variables, excluding the special case of 0-edges, seems to be a rare phenomenon for unitary matrices representing arbitrary quantum operations. However, it is possible to construct such matrices with edges that skip an arbitrary number of variable levels [21]. Therefore, skipped variables have to be taken into account carefully in each QMDD algorithm.

B. Local Modifications and Vertex Weights

We have already seen that large effort is put on normalization (of edge weights) in order to ensure canonical representations. Local modifications within a QMDD, e.g., due to a variable interchange, may lead to changes of edge weights which destroy normalization and, hence, require a rework of a large part of the QMDD in order to restore the normalization.

Example 13. *Consider the QMDD shown in Fig. 8 which was built using vertex normalization. Assume that, as part of a reordering process, we interchange variables x_2 and x_3. This leads to a QMDD structure as shown in Fig. 8, i.e., the weight of the leftmost edge of the x_1-vertex changes from* 1 *to i. Thus, this vertex is not normalized anymore according to Definition 10. In the worst case, changes like this propagate through the entire QMDD structure. As a result, variable interchanges (and local modifications in general) are no longer local operations which can have a significant effect on the overall efficiency.*

The basic idea to overcome this problem is to store weight changes (as they result from local modifications) within the vertices as vertex weights instead of propagating them to incoming edges. The advantage of this approach is that we easily maintain a normalized structure. More precisely, vertex weights can be interpreted as the normalization factors of the respective (sub)matrices. So far, these were aimed to be equal to 1, i.e., all vertices were supposed to represent a normalized matrix with a normalization factor of 1. In this case, vertex weights do not have any effect. Otherwise, they can be removed by simply applying them to the weights of all outgoing edges and then performing vertex normalization. However, we are not actually performing these multiplications. This is because they would only affect the weights of certain edges, but the whole QMDD would still be normalized—w.r.t. a slightly different normalization scheme, for which the normalization factor of that particular sub-matrix is adjusted. In short, the change or introduction of vertex weights is just a small change to the applied normalization scheme, but maintains the normalized structure. Consequently, the use of vertex weights preserves the (optimal) structure sharing according to Theorem 2 and, hence, enables local operations to be performed efficiently. This will be outlined in detail in the following section for the purpose of variable interchange.

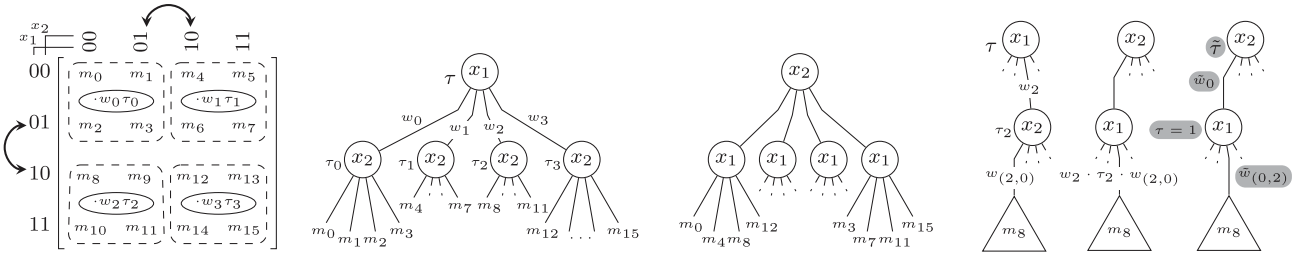
C. Variable Interchange Scheme for QMDDs

As discussed above, achieving normalization can be a severe obstacle when performing modifications on QMDDs such as adjacent variable interchanges. However, using the concept of vertex weights, this problem is solved, i.e., a local modification such as a variable interchange can be performed without ramifications to other parts of the QMDD structure. The particular way of employing vertex weights is demonstrated in this section.

We use an interchange scheme which is similarly applied in other decision diagram types, e.g., BDDs: consider a BDD where two adjacent variables x_1 and x_2 shall be interchanged. Then, each x_1-vertex is replaced by an x_2-vertex which shall represent the same Boolean function in order to make the swap a local operation. This is done by interchanging the labels of the vertices and permuting the subtrees representing the respective cofactors [10]. Analogously, for QMDDs each x_1-vertex is replaced by an x_2-vertex which shall represent the same functionality. By doing so, an interchange of variables x_1 and x_2 for a given matrix leads to a permutation of sub-matrices as illustrated in Fig. 9(a), i.e., the swapping of certain rows and columns. This accordingly needs to be conducted in the QMDD structure in which each of the affected sub-matrices is represented by a vertex as well as weighted edges.

That is, to interchange two adjacent variables x_1 and x_2 in a QMDD (where x_1 precedes x_2 in the variable order), we process all vertices that are labeled x_1. We skip all such vertices that do not point to any x_2-vertex. For each of the remaining x_1-vertices V with outgoing edges $e_i^V (i = 0, \ldots, r^2 - 1)$, from which at least one edge points to a v_2-vertex, we perform the following three steps.

1) Create an $r^2 \times r^2$ square matrix $\mathbf{T} = (t_{ij})$ and set t_{ij} to be the jth outgoing edge of the x_2-vertex pointed to by e_i^V and multiply the weight of t_{ij} with the weight of e_i^V and the (vertex) weight of the v_2-vertex. If the destination of e_i^V is not labeled with x_2, set $t_{ij} = e_i^V$ instead.



(a) Matrix permutation scheme (b) Before the interchange (c) Structure after the interchange (d) Refactoring edge- and vertex weights

Fig. 9. Sketch of the variable interchange procedure for binary QMDDs ($r = 2$).

2) From each column j of $\mathbf{T}$ create a vertex labeled x_1 with outgoing edges $e_i = t_{ij}$ and let e_j^V point to this vertex. Relabel V to x_2.
3) Apply the normalization scheme and store the normalization factor of V by multiplying it to the current vertex weight τ_V.

This procedure is illustrated by the following example.

Example 14. *Consider the case of a binary QMDD ($r = 2$) in which two adjacent variables x_1 and x_2 are to be interchanged. On matrix level, this corresponds to a permutation of rows and columns as illustrated in Fig. 9(a). According to step 1, a matrix containing all subtrees representing the submatrices m_0 until m_{15} is created first [see Fig. 9(b)]. Then, these subtrees are rearranged in step 2 eventually leading to the structure shown in Fig. 9(c). Finally, the respective vertices are normalized in step 3. This is illustrated in Fig. 9(d) for the subtree m_8. First, this subtree is relocated (according to the previous steps). Then, the product of the corresponding edge and vertex weights is concentrated at the bottom level. The final factorization of this product (highlighted in gray) is achieved by applying vertex normalization to the new structure.*

The interchange procedure operates in the same fashion on each sub-matrix of the particular partitioning level that corresponds to the interchanged variables. Thus, it preserves structural equivalence. This guarantees that (an optimal) vertex sharing is maintained and we will not create vertices that only differ by their vertex weight. By using vertex weights, a normalized structure can be achieved without the need to correct ramifications in possibly large parts of the QMDD. The potentially expensive transformation to a QMDD with trivial vertex weights (canonical representation) has to be performed at most once, after we have arrived at the final variable order. However, most effective vertex weights ($\neq 1$) vanish by further variable interchanges.

Overall, this enables us to perform variable interchanges efficiently as local operations and to use these interchanges as building blocks for variable reordering techniques. The effectiveness of using variable reordering for reducing the QMDD size will be evaluated next.

VI. Evaluation

In this section, we provide an evaluation of the overall efficiency of QMDDs. In a first step, we build the QMDD representations for a set of important quantum algorithms to demonstrate (a) the compactness that is achieved by QMDD representations as well as (b) the efficiency of constructing and manipulation these representations. In a second step, we consider the scheme for modifying the applied variable order of QMDDs by interchanging adjacent variables which was proposed in the previous section. This scheme explicitly overcomes the limitation of the initial proposal of QMDDs (as introduced in [12] and [13]) and allows for determining an even more efficient representation of quantum functionality.

Our experimental results are summarized in Table II. Here, the respective QMDD sizes (i.e., the number of non-terminal vertices; denoted by size) are presented for a selection of benchmark functions. We distinguish between: 1) the original approach (according to [12] and [13]) in which the variable order (given from the initial description of the quantum functionality) cannot be modified during the construction of the QMDD; 2) an improved approach which allows changes in the variable order based on sifting; and 3) an exact method which establishes the optimal variable order with respect to the size of the resulting QMDD. In addition to the absolute size values, we also provide the percentaged improvements w.r.t. the number of nodes (denoted by Impr.) as well as the run-time (in CPU seconds) required to generate the QMDDs (denoted by time). As benchmarks we applied circuits realizing Grover algorithms (Grover-N), error correction functionality (k-qubit-code, taken from [22]), and Quantum Fourier Transforms (QFT-N) where N denotes the number of qubits. All experiments have been conducted on a 2.8 GHz Intel Core i7 machine with 8 GB of main memory running Linux.

It can be seen that the initial representations (according to [12] and [13]) are already rather compact—given the fact that the corresponding transformation matrices are of dimension $2^N \times 2^N$. Moreover, these representations can be established in negligible run-time. But much better results can be achieved when changes in the variable order are allowed. Then, reductions of up to two orders of magnitude can be observed. A comparison of the sifting and the exact approach shows that near-to-optimal results can already be achieved in almost no run-time by the heuristic. Note that this efficiency mainly results from the improvements on local modifications introduced in this paper and would not be possible when using the initial QMDD definition from [12] and [13].

Overall, these evaluations show that QMDDs offer a very compact representation of quantum functionality which—in contrast to alternatives such as QuIDDs [9] or QDDs [5] —do not rely on Shannon decomposition, but rather on a decomposition scheme that more naturally models quantum mechanical

TABLE II
EVALUATION

Benchmark		Original [12], [13]		Sifting			Exact			
Name	#Qubits	Size	Time	Size	Impr. over Original	Time	Size	Impr. over Original	... over Sifting	Time
Grover-7	7	187	0.01	36	-81%	<0.01	35	-81%	-3%	0.37
Grover-9	9	722	0.02	52	-93%	0.01	51	-93%	-2%	29.14
Grover-11	11	2817	0.15	67	-98%	0.02	66	-97%	-1%	3709
5-qubit-code	9	90	0.01	57	-37%	0.01	43	-52%	-25%	24.73
7-qubit-code	7	44	<0.01	26	-41%	<0.01	26	-41%	–	0.35
9-qubit-code	9	40	<0.01	22	-45%	0.01	22	-45%	–	24.47
9-qubit-code	17	1172	0.01	60	-95%	0.04	(84)*	(-93%)	(+40%)	>7200
QFT-3	3	22	<0.01	9	-59%	<0.01	9	-59%	–	<0.01
QFT-4	4	86	<0.01	24	-72%	<0.01	24	-72%	–	0.01
QFT-5	5	342	<0.01	40	-88%	<0.01	40	-88%	–	0.01
QFT-6	6	1366	<0.01	103	-92%	<0.01	103	-92%	–	0.1
QFT-7	7	5462	0.02	167	-97%	0.02	167	-97%	–	1.2

* The exact approach did not terminate within the limit of 7200 CPU seconds. At that time, the best result achieved so far was a QMDD size of 84 vertices.

systems. The proposed concepts allow for an efficient construction and manipulation of practically relevant quantum circuits. Due to the fundamental improvements presented in this paper, different variable orders can efficiently be applied which has a significant impact on the resulting QMDD size and, thus, on the overall efficiency of the representation.

VII. CONCLUSION

In this paper, we considered QMDDs, a compact, and canonic representation of quantum functionality which has initially been proposed in [12] and [13]. While the basic ideas of QMDDs are already around for a while, significant shortcomings limited their applicability thus far. As a consequence, the full potential of QMDDs or decision diagrams for quantum functionality in general has hardly been exploited yet. In order to address these problems, we presented a comprehensive definition of QMDDs for the first time. Furthermore, we showed that, based on this definition, QMDDs indeed provide a proper platform for an efficient quantum function manipulation and additionally proposed a solution that allows for local modifications. This provides the basis for a more sophisticated application of decision diagrams in the domain of quantum computation including solutions for synthesis, simulation, and verification. An implementation of QMDDs is publicly available at `http://www.informatik.uni-bremen.de/agra/eng/qmdd.php`.

In future work, we plan to further substantiate this by studying the formal complexity of the QMDD (matrix) operations as well as QMDD sizes for different classes of quantum functionality.

REFERENCES

[1] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. New York, NY, USA: Cambridge Univ. Press, 2000.

[2] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. Found. Comput. Sci.*, Santa Fe, NM, USA, 1994, pp. 124–134.

[3] L. M. K. Vandersypen *et al.*, "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance," *Nature*, vol. 414, pp. 883–887, Dec. 2001.

[4] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. Theory Comput.*, Philadelphia, PA, USA, 1996, pp. 212–219.

[5] A. Abdollahi and M. Pedram, "Analysis and synthesis of quantum circuits by using quantum decision diagrams," in *Proc. Design Autom. Test Europe*, 2006, pp. 317–322.

[6] M. Soeken, R. Wille, C. Hilken, N. Przigoda, and R. Drechsler, "Synthesis of reversible circuits with minimal lines for large functions," in *Proc. ASP Design Autom. Conf.*, Sydney, NSW, Australia, 2012, pp. 85–92.

[7] P. Niemann, R. Wille, and R. Drechsler, "Efficient synthesis of quantum circuits implementing clifford group operations," in *Proc. ASP Design Autom. Conf.*, Singapore, 2014, pp. 483–488.

[8] R. Wille, D. Große, D. M. Miller, and R. Drechsler, "Equivalence checking of reversible circuits," in *Proc. Int Symp. Mult.-Valued Logic*, Naha, Japan, 2009, pp. 324–330.

[9] G. Viamontes, I. Markov, and J. Hayes, "High-performance QuIDD-based simulation of quantum circuits," in *Proc. Design Autom. Test Europe*, Paris, France, 2004, pp. 1354–1355.

[10] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[11] G. F. Viamontes, M. Rajagopalan, I. L. Markov, and J. P. Hayes, "Gate-level simulation of quantum circuits," in *Proc. ASP Design Autom. Conf.*, Kitakyushu, Japan, 2003, pp. 295–301.

[12] D. M. Miller and M. A. Thornton, "QMDD: A decision diagram structure for reversible and quantum circuits," in *Proc. Int. Symp. Mult.-Valued Logic*, 2006, p. 6.

[13] D. M. Miller and M. A. Thornton, *Multiple-Valued Logic: Concepts and Representations*. San Rafael, CA, USA: Morgan and Claypool, 2008.

[14] D. Goodman, M. A. Thornton, D. Y. Feinstein, and D. M. Miller, "Quantum logic circuit simulation based on the QMDD data structure," in *Proc. Int. Reed-Muller Workshop*, Oslo, Norway, 2007, pp. 99–105.

[15] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Proc. Int. Conf. CAD*, Santa Clara, CA, USA, 1993, pp. 42–47.

[16] P. Niemann, R. Wille, and R. Drechsler, "On the 'Q' in QMDDs: Efficient representation of quantum functionality in the QMDD data-structure," in *Proc. Int. Conf. Revers. Comput.*, Victoria, BC, Canada, 2013, pp. 125–140.

[17] C. G. Khatri and C. R. Rao, "Solutions to some functional equations and their applications to characterization of probability distributions," *Sankhya*, vol. 30, no. 2, pp. 167–180, 1968.

[18] X. Zhang, Z. Yang, and C. Cao, "Inequalities involving Khatri-Rao products of positive semi-definite matrices," *Appl. Math. E-notes*, vol. 2, pp. 117–124, 2002.

[19] D. M. Miller, D. Y. Feinstein, and M. A. Thornton, "QMDD minimization using sifting for variable reordering," *Mult.-Valued Logic Soft Comput.*, vol. 13, nos. 4–6, pp. 537–552, 2007.

[20] D. Y. Feinstein, M. A. Thornton, and D. M. Miller, "Minimization of quantum multiple-valued decision diagrams using data structure metrics," *Mult.-Valued Logic Soft Comput.*, vol. 15, no. 4, pp. 361–377, 2009.

[21] D. Y. Feinstein and M. A. Thornton, "On the skipped variables of quantum multiple-valued decision diagrams," in *Proc. Int. Symp. Mult.-Valued Logic*, Tuusula, Finland, 2011, pp. 164–169.

[22] N. D. Mermin, *Quantum Computer Science: An Introduction*. New York, NY, USA: Cambridge Univ. Press, 2007.

Philipp Niemann (M'15) received the Diploma degree in mathematics from the University of Bremen, Bremen, Germany, in 2012, where he is currently pursuing the Ph.D. degree with the Group of Computer Architecture, under the supervision of R. Drechsler.

He has published several papers on international conferences such as ASP Design Automation Conference, Design Automation Test Europe, and Reversible Computation. His current research interests include design of reversible and quantum circuits with a focus on decision diagrams as well as in the verification of formal models.

Robert Wille (SM'15) received the Diploma and Dr.-Ing. degrees in computer science from the University of Bremen, Bremen, Germany, in 2006 and 2009, respectively.

Since 2006, he has been with the Group of Computer Architecture, University of Bremen, and the German Research Center for Artificial Intelligence, Bremen, since 2013. He was a Lecturer with the University of Applied Science, Bremen, and a Visiting Professor with the University of Potsdam, Potsdam, Germany, and Technical University Dresden, Dresden, Germany. In the eight years of his research activity, he has published over 100 papers in journals and conferences and served in program committees of numerous conferences such as ASP Design Automation Conference, International Conference Computer Aided Design, and International Symposium Multiple-Valued Logic. His current research interests include design of circuits and systems for both conventional and emerging technologies with a focus in the domain of verification and proof engines.

David Michael Miller (M'85) received the B.Sc. degree in physics and mathematics from the University of Winnipeg, Winnipeg, MB, Canada, in 1971, and the M.Sc. and Ph.D. degrees in computer science from the University of Manitoba, Winnipeg, in 1973 and 1976, respectively.

He was the Chair of the Department of Computer Science, University of Victoria, Victoria, BC, Canada, in 1987. He was a Faculty Member with the University of New Brunswick, Fredericton, NB, Canada, the University of Winnipeg, and the University of Manitoba. He served as the Dean of the Faculty of Engineering, University of Victoria, from 1997 to 2008 and currently holds the position of Associate Vice-President Research. He has co-authored two books on spectral logic and published over 100 papers in the above areas. His current research interests include decision diagrams, reversible and quantum logic, spectral logic, and multiple-valued logic.

Dr. Miller is the Secretary of the IEEE Computer Society Technical Committee on Multiple-Valued Logic.

Mitchell A. Thornton (M'85–SM'99) received the B.S. degree in electrical engineering from Oklahoma State University, Stillwater, OK, USA, in 1985, the M.S. degree in electrical engineering from the University of Texas-Arlington, Arlington, TX, USA, in 1990, the M.S. degree in computer science from Southern Methodist University, Dallas, TX, in 1993, and the Ph.D. degree in computer engineering from Southern Methodist University, in 1995.

He was a Senior Electronic Systems Engineer with E-Systems, Inc., Greenville, TX, from 1986 to 1991. He was employed as a Design Engineer with Cyrix Corporation, Richardson, TX, from 1992 to 1993. He has served as a full-time Faculty Member with the University of Arkansas, Fayetteville, AR, USA, from 1995 to 1999, and Mississippi State University, Starkville, MS, USA, from 1999 to 2002, and currently holds the Cecil H. Green Chair of Engineering with Southern Methodist University where he is a Professor. Additionally, he serves as the Technical Director and Interim Associate Director of the Darwin Deason Institute for Cyber Security. He is a Licensed Professional Engineer in the states of Arkansas, Mississippi, and Texas. Within IEEE, he has served as Chair of several committees. His current research interests include electronic design automation algorithms for synthesis and verification, computer arithmetic, disaster tolerant systems and modeling, and computer security hardware.

Rolf Drechsler (F'05) received the Diploma and Dr. phil. nat. degrees in computer science from the J. W. Goethe University Frankfurt am Main, Frankfurt am Main, Germany, in 1992 and 1995, respectively.

He was with the Institute of Computer Science, Albert-Ludwigs University, Freiburg im Breisgau, Germany and with the Corporate Technology Department, Siemens AG, Munich, Germany. Since 2001, he has been with the University of Bremen, Bremen, Germany, where he is currently a Full Professor and the Head of the Group for Computer Architecture, Institute of Computer Science. Since 2011, he has also been the Director of the Cyber-Physical Systems Group with the German Research Center for Artificial Intelligence, Bremen. He has published over 250 papers and served in program committees of international conferences such as Design Automation Conference, Design Automation Test Europe, and International Conference Computer Aided Design. His current research interests include the development and design of data structures and algorithms with a focus on circuit and system design.