On the "Q" in QMDDs: Efficient Representation of Quantum Functionality in the QMDD Data-structure

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

[1] Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[2] Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
{pniemann,rwille,drechsle}@informatik.uni-bremen.de

Abstract. The Quantum Multiple-valued Decision Diagram (QMDD) data-structure has been introduced as a means for an efficient representation and manipulation of transformation matrices realized by quantum or reversible logic circuits. A particular challenge is the handling of arbitrary complex numbers as they frequently occur in quantum functionality. These numbers are represented through edge weights which, however, represent a severe obstacle with respect to canonicity, modifiability, and applicability of QMDDs. Previously introduced approaches did not provide a satisfactory solution to these obstacles. In this paper, we propose an improved factorization scheme for complex numbers that ensures a canonical representation while, at the same time, allows for local changes. We demonstrate how the proposed solution can be exploited to improve the data-structure itself (e.g. through variable re-ordering enabled by the advanced modifiability) and how applications such as equivalence checking benefit from that.

1 Introduction

Exploiting quantum mechanical phenomena such as superposition and entanglement, quantum computation [1] offers the promise of efficient computing for problems that are of exponential difficulty for classical computing paradigms. For this purpose, information is stored in terms of qubits, i.e. a superposition of the Boolean states 0 and 1. This enables one to solve many important problems (e.g. database search, factorization, graph problems) significantly faster than with classical approaches (see e.g. [2–4]). The states of the qubits are modified by quantum operations which can be represented by unitary matrices that may include complex numbers.

Hence, an efficient and compact data-structure for the representation and manipulation of the respective quantum functionality is important for many design tasks in this area. Accordingly, a variety of decision diagram types have been introduced such as the *X-decomposition Quantum Decision Diagram* (XQDD) [5], the *Quantum Information Decision Diagram* (QuIDD) [6], and the *Quantum Multiple-valued Decision Diagram* (QMDD) [7]. In this work, we focus on QMDDs which already have successfully been used in applications such as equivalence checking [8], property checking [9], or synthesis [10]. However, in these applications the focus was often on the representation of different quantum realizations for reversible Boolean functions. Although pure quantum functionality has also been represented using QMDDs, some crucial aspects have not been addressed yet.

In particular, the handling of arbitrary complex numbers – a core characteristic of quantum functionality – is unsatisfactory. So far, these numbers are represented through edge weights corresponding to common scalar factors to be applied to all entries of a certain sub-matrix. But, as entries in sub-matrices can be factorized in numerous fashions, several representations of a particular quantum circuit are possible. This has a large influence with respect to canonicity, modifiability, or applicability of QMDDs (this is discussed later in detail in Section 3).

In this work, we investigate this problem of factorization and, eventually, propose a solution allowing for an efficient and modifiable representation of general quantum functionality in the QMDD data-structure. To this end, we review existing factorization efforts by normalization of edge weights and identify their drawbacks. In addition, we prove that QMDD representations of a fixed matrix have the same invariant structure of vertices and connecting edges for a wide range of normalization schemes. However, weights of corresponding edges may differ by some non-zero factor for different schemes. These observations lead to an extension to the QMDD data-structure by so called vertex weights that, independently from the considered quantum functionality, allow for a canonical representation as well as an efficient manipulation. By this, central problems of previous QMDD realizations are solved.

The remainder of this paper is structured as follows. In order to keep the paper self-contained, Section 2 briefly reviews the basics on the QMDD data-structure. Afterwards, the problem with respect to the representation of quantum functionality is discussed and investigated in Section 3. The solution derived from these observations, i.e. the use of vertex weights, is proposed in Section 4. The use of the proposed solution for adjacent variable interchange is demonstrated in Section 5 and evaluated in Section 6. Section 7 concludes the paper.

2 Preliminaries

Quantum systems are composed of *qubits*. Analogously to classical bits, a qubit can be in one of the computational *basis states* $|0\rangle$ and $|1\rangle$, but also in a so called *superposition* $\alpha|0\rangle + \beta|1\rangle$ for complex-valued α, β with $|\alpha|^2 + |\beta|^2 = 1$. The number of basis states of a qubit is called its *radix*. Usually, we use radix two but also qubits with more than two basis states (qudits) have been considered [11].

Quantum circuits are commonly represented by their complex-valued, unitary *transformation matrix*. A special case are permutation matrices which represent reversible circuits. The transformation matrix of an n-qubit circuit has dimension $r^n \times r^n$ where r is the radix. These matrices grow exponentially in size, thus standard representations as tables of complex numbers are restricted to circuits with a small number of qubits.

To represent and manipulate larger circuits, we need more elaborate representations that take advantage of the specific properties of transformation matrices:

- Quantum gates often only operate on a small subset of qubits of a quantum system. The transformation matrix for the whole system, which is the Kronecker product of the smaller gate matrix and identity matrices, contains the same pattern (gate matrix) several times. Thus, *similar structures* occur which offers the opportunity for compression.
- Transformation matrices, especially gate matrices, are often sparsely populated, i.e. they contain many zero entries. Therefore, *blocks of zero* can be marked and treated separately.

Taking this into account, we observe that an $r^n \times r^n$ matrix can be partitioned into r^2 sub-matrices of dimension $r^{n-1} \times r^{n-1}$ as

$$M = \begin{bmatrix} M_0 & M_1 & \cdots & M_{r-1} \\ M_r & M_{r+1} & \cdots & M_{2r-1} \\ \vdots & \vdots & \ddots & \vdots \\ M_{(r-1)r} & M_{(r-1)r+1} & \cdots & M_{r^2-1} \end{bmatrix}.$$

This partitioning can be repeated until we reach the level of single matrix entries. Now, the fundamental idea is to create a vertex for each of these matrices with unidirectional edges pointing to the vertices of the respective sub-matrices. More precisely:

Definition 1. *A* Quantum Multiple-valued Decision Diagram (QMDD) *is a directed acyclic graph with the following properties:*

- *There is a single* terminal vertex *representing the complex number* 1 *without any outgoing edge.*
- Non-terminal vertices *are labelled by an* r^2*-valued selection variable and have* r^2 *outgoing edges designated* $e_0, e_1, \ldots, e_{r^2-1}$.
- *There is a single* root vertex *which has a single incoming edge (the* root edge*) that itself has no source vertex.*
- *Every edge (including the root edge) has an associated complex-valued* weight *and edges with a weight of 0 (*0-edges*) point to the terminal vertex.*
- *The selection variables are* ordered*, assume with no loss of generality* $x_0 \prec x_1 \prec \cdots \prec x_{n-1}$*. On each path from the root vertex to the terminal vertex the variables appear in this order while each variable appears at most once.*
- *There are no* redundant *vertices, i.e. no non-terminal vertex has* r^2 *identical outgoing edges (destinations and weights).*
- *Non-terminal vertices are* unique*, i.e. no two non-terminal vertices labelled by the same selection variable have the same set of outgoing edges (destinations and weights).*
- *Non-terminal vertices are* normalized *(see details in the following section).*

Each assignment to the selection variables corresponds to choosing the respective sub-matrices in the partitioning process and, therefore, to some matrix entry. Thus, for any entry of the $r^n \times r^n$ matrix the QMDD can be evaluated in at most n steps by multiplying the weights on the path from the root vertex to the terminal vertex that is determined by the respective assignment.

Example 1. Fig. 1 shows QMDD representations of a 2-qubit quantum circuit. Outgoing edges point to the vertices representing the top left, top right, bottom left, and bottom right sub-matrix from left to right. For example, the highlighted matrix entry $-i$ in Fig. 1a corresponds to the paths highlighted in bold in Fig. 1b and Fig. 1c. Its value can be determined by multiplying the edge weights on these paths.

For simplicity we omit edge weights equal to 1 in illustrations of QMDDs and indicate 0-edges, i.e. edges that point to the terminal vertex with weight 0, by stubs.

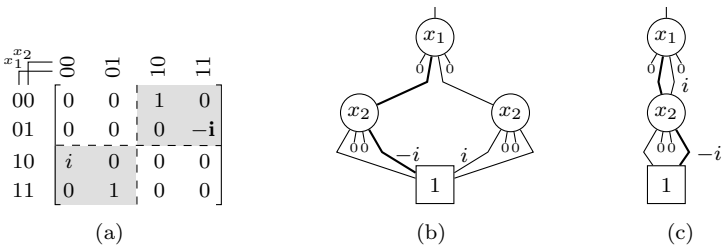


Fig. 1. Matrix and QMDD representations of a 2-qubit quantum circuit.

3 Normalization of Edge Weights in QMDDs

The main difference between QMDDs and decision diagrams for conventional logic are the complex-valued *edge weights*. They represent common scalar factors to be applied to all entries in a sub-matrix represented by the vertex to which the respective edge is pointing to. Hence, the precise value of a particular matrix entry is determined by multiplying the weights of all edges in the corresponding path from the root vertex to the terminal.

Using weighted edges allows for the representation of structurally equivalent sub-matrices whose entries differ only by a scalar factor with a single vertex. For example, the matrix in Fig. 1a includes two structurally equivalent sub-matrices (highlighted in gray) which differ by a common scalar factor only. But instead of representing each sub-matrix separately (as illustrated in Fig. 1b), weighted edges allow for a representation with a shared vertex (as illustrated in Fig. 1c).

However, as entries in the respective matrices can be factorized in numerous fashions, *normalization* of edge weights plays a significant role. More precisely:

- The representation of structurally equal matrices by the same vertex is only possible if a decomposition into a scalar factor and a normal form is available. In order to determine that, factorization has to be conducted using a normalization scheme.
- Data-structures like QMDDs benefit from providing a possibly canonical representation to be exploited e.g. in equivalence checking [8]. In QMDDs, canonicity is achieved with respect to an applied variable ordering but also depends on how the edge weights (scalar factors) have been determined.
- In many applications, e.g. synthesis, the determination of the smallest or largest magnitude (or even greatest common factors) of matrix entries is of interest. Here, normalization can be exploited as it allows for an *upwards propagation* of the desired values.

Unfortunately, ensuring and maintaining a normalized representation is subject to severe obstacles. In the past, proper normalization rules and corresponding schemes have been proposed. However, they either do not ensure a canonical representation or suffer from the fact that local modifications in the QMDD structure (caused e.g. through re-ordering of vertices as commonly applied in optimization approaches like sifting) possibly destroy the normalized representation.

In this section, we review existing and propose new normalization rules and illustrate the obstacles with them. Afterwards, we discuss the application of these vertex-based rules within generic normalization schemes for entire QMDDs. We focus on canonicity as the primary requirement. Here, we prove that QMDD representations of the same matrix that follow (possibly different) normalization schemes always have

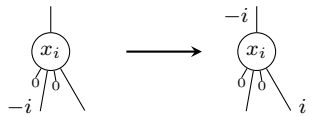


Fig. 2. Normalizing a vertex using Normalization Rule 1.

the same invariant structure of vertices and edges, and only differ in the weights of corresponding edges by some non-zero factor. These observations lead to an extension to the QMDD data-structure by so called *vertex weights* that maintain the normalized structure after local modifications by only using local re-normalizations.

3.1 Normalization Rules

With the introduction of the QMDD data-structure in [7], various rules for the normalization of edge weights have been proposed. In the following, we define *normalization rules* as follows:

Definition 2. *A* normalization rule *defines a property that the weights of the outgoing edges of a QMDD vertex must exhibit in order to call the vertex* normalized. Normalizing *a vertex means that we divide the weights of all outgoing edges by a* normalization factor *such that this leads to a normalized vertex.*

The first normalization rule that was used for QMDDs is defined as follows:

Normalization Rule 1 *A QMDD vertex is* normalized *if the first edge with a non-zero edge weight has weight* $+1$*, i.e. for some* k *(*$0 \leq k \leq r^2 - 1$*) the edge* e_k *has weight* $+1$ *and all edges* e_i *have weight 0 for* $i = 0, \ldots, k-1$.

Example 2. The application of Rule 1 is illustrated in Fig. 2. Since normalization factors (here: $-i$) can easily be propagated to incoming edges when building a QMDD bottom-up, the common way to ensure normalized vertices according to this rule is to apply the normalization rule as the QMDD is built.

Normalization Rule 1 enables a canonical representation [7] and, hence, is very useful for applications like equivalence checking (see e.g. [8]). However, once a QMDD has been built following this scheme, local modifications on the data-structure often require a re-normalization of the entire QMDD.

Example 3. Consider the QMDD shown in Fig. 3a which has been built following Normalization Rule 1. Afterwards, e.g. as part of a re-ordering process, the variables x_1 and x_2 shall be interchanged. According to the corresponding matrices (see Fig. 3b), this leads to a QMDD structure as shown in Fig. 3c, i.e. the weight of the leftmost edge of the x_0-vertex changes from 1 to i. Thus, this vertex is not normalized anymore. In the worst case, changes like this propagate through the entire QMDD structure. As a result, variable interchanges are no longer local operations and, hence, significantly complicate established optimization approaches such as variable re-ordering by adjacent variable interchange as used in sifting [12].

In order to address this problem, an alternative normalization rule has been proposed in [12].

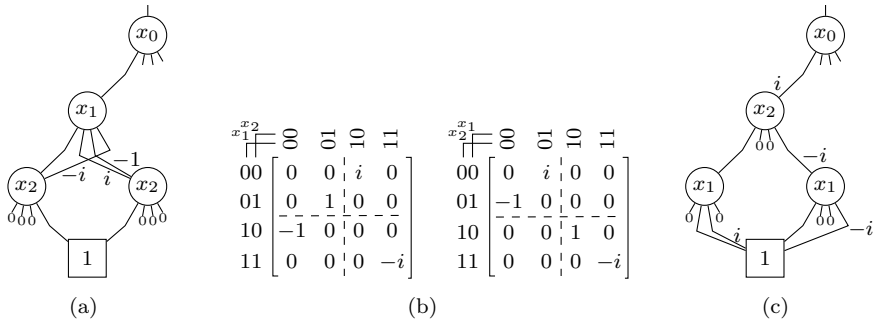


Fig. 3. Matrix and QMDD representations for interchanged variables.

Normalization Rule 2 *A QMDD vertex is* normalized *if its edges are of the form that the largest*[1] *weight on any outgoing edge is 1.*

This normalization indeed enables local operations since local maxima, i.e. matrix elements with the largest magnitude appearing in the respective sub-matrix, are propagated upwards to the root edge weight. Since these maxima do not change during a local variable reordering, the resulting structure of the QMDD is not affected by this and normalization is preserved. In contrast, this scheme destroys the canonicity of the representation as illustrated in the next example.

Example 4. Consider again the QMDDs shown in Fig. 1. Inspection shows that both are properly normalized according to Rule 2. However, although both QMDDs represent the same functionality and follow the same variable ordering and normalization scheme, their structure is not equivalent. That is, the proposed normalization does not lead to unique representations.

As another alternative, we propose the following normalization rule:

Normalization Rule 3 *A QMDD vertex is* normalized *if the edges are of the form that (1) no edge has a magnitude larger than* 1 *and (2) the first edge exhibiting this largest magnitude has exactly weight* $+1$.

Example 5. In most cases, Rule 3 coincides with Rule 2. E.g., in Fig. 1 the only vertex that is normalized according to Rule 2, but does not obey to Rule 3 is the x_2-vertex on the right of Fig. 1b, where we have edge weight i "before" $+1$.

Rule 3 again enables a canonical representation of QMDDs as it provides a unique tie breaking mechanism in case of several edge weights having the same (largest) magnitude. But as for Rule 1, swapping adjacent variables is not a local operation in general. This can be seen from Example 3 where the QMDDs are the same when Rule 3 is used instead of Rule 1.

The benefit of Rule 3 is that it still ensures canonicity when using the smallest non-zero magnitude. Moreover, it then realizes an upward propagation of smallest absolute values and, hence, enables a fast determination of how "far" a matrix is "away" from being a Boolean permutation matrix by just looking at the root edge weight.

[1] We refer to [13] for a more detailed consideration on the magnitude-based order of complex numbers that is meant here.

3.2 Normalization Schemes

As discussed above, several normalization rules for QMDDs exist. Now, we consider their generic application in the normalization process for an entire QMDD. For each vertex in a QMDD, these rules basically compute a normalization factor which has to be applied to the represented (sub-)matrix. More precisely:

Definition 3. *Let* $\mathrm{Mat}(\mathbb{C}) := \{M \in \mathbb{C}^{m\times n} |\ m, n \in \mathbb{N}\}$ *be the set of complex-valued matrices. A map* $\mathrm{N} : \mathrm{Mat}(\mathbb{C}) \to \mathbb{C}$ *is called a* normalization scheme *if it satisfies*

$$\mathrm{N}(\alpha M) = \alpha\,\mathrm{N}(M) \text{ for all } M \in \mathrm{Mat}(\mathbb{C}), \alpha \in \mathbb{C} \tag{1}$$

$$\mathrm{N}(M) = 0 \Leftrightarrow \text{ all entries in } M \text{ are zero.} \tag{2}$$

Using the normalization scheme N*, a QMDD vertex representing a matrix* M *will be* normalized *by dividing all outgoing edge weights by its* normalization factor $\mathrm{N}(M)$ *and will afterwards represent the matrix* $M'_{\mathrm{N}} := \frac{M}{\mathrm{N}(M)}$.

Remark 1. Note that property (1) guarantees that structurally equal matrices (which only differ by a scalar factor $\alpha \neq 0$) are compressed to a shared vertex, i.e.

$$(\alpha M)'_{\mathrm{N}} = \frac{\alpha M}{\mathrm{N}(\alpha M)} = \frac{\alpha M}{\alpha \mathrm{N}(M)} = \frac{M}{\mathrm{N}(M)} = M'_{\mathrm{N}},$$

while (2) is just another way of saying that all 0-edges must point to the terminal vertex. Thus, normalization schemes lead to unique QMDD representations. Conversely, any normalization *rule* that ensures canonicity, e.g. Rules 1 and 3, can be extended to a normalization scheme. Note that a vertex is normalized if, and only if, it represents a matrix with normalization factor 1, i.e.

$$\mathrm{N}(M'_{\mathrm{N}}) = \mathrm{N}\left(\frac{M}{\mathrm{N}(M)}\right) = \frac{1}{\mathrm{N}(M)}\mathrm{N}(M) = 1.$$

A generic normalization scheme as defined in Definition 3 is not limited to rules that take into account only local information about edge weights. It may rather rely on arbitrary knowledge about the represented matrix. Thus, one could assume that significantly different QMDD structures result for the same matrix. However, as the following theorem shows, QMDDs representing the same matrix and following (possibly different) normalization schemes indeed exhibit an isomorphic structure of vertices and edges.

Theorem 1. *Consider a QMDD that uniquely represents a complex-valued matrix using a normalization scheme. Any QMDD that represents the same matrix using a different normalization scheme has the same structure of vertices and edges, while the weights of corresponding edges may differ by some non-zero factor.*

Proof (Sketch). Assume there are matrices that lead to different QMDD structures for different normalization schemes. From these we can choose a matrix of minimal size. Clearly, this matrix cannot be a single complex number, so without loss of generality we may consider the regular structure of r^2 sub-matrices. Since these are smaller and the matrix was chosen minimal, their QMDD structure must be the same for any normalization scheme. Now, when creating the top vertex of a QMDD, edges to these sub-structures are used and applying different normalization schemes may result in different (non-zero) edge weights, but does not change the structure of the vertex. This is a contradiction to our assumption and proves the theorem. □

This is an important result. While it was already known that QMDD representations are canonical for certain normalization rules, Theorem 1 now tells that even regardless of the normalization scheme, the QMDD structure is an *invariant* of a matrix. This can be exploited in order to provide a normalization approach that not only guarantees this invariant canonical QMDD structure (as Rule 1 and Rule 3), but additionally also allows for certain local operations (as possible in Rule 2). For this purpose, the QMDD data-structure needs to be slightly extended as described next.

4 Introducing Vertex Weights

So far, we discussed (1) that local changes of edge weights might require to rework a large part of the QMDD in order to restore the normalization (as illustrated in Example 3) and (2) that regardless of the normalization scheme the structure of a QMDD is invariant and can already be established using simple normalization rules such as Rule 1 or Rule 3. Now, these observations are exploited in order to propose a slightly revised QMDD data-structure which is capable of both, representing matrices in a canonical fashion and enabling local operations.

The basic idea is to store weight changes (as they result from local modifications) within the vertices instead of propagating them to incoming edges. Therefore, we suggest to extend the QMDD definition as follows:

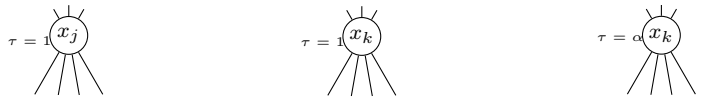
Definition 4. *Each non-terminal QMDD vertex is enriched with a complex-valued* vertex weight (v-weight) $\tau \neq 0$. *A vertex weight* $\tau \neq 1$ *is called* effective.

Remark 2. Vertex weights represent scalar factors to be applied to all entries in the sub-matrix represented by the respective vertex. Hence, to determine the value of a particular matrix entry they have to be included when computing the product of the edge weights on the respective path. Besides this *v-weight interpretation*, also the *standard interpretation* can be used to evaluate QMDDs, i.e., ignoring vertex weights. Thus, a QMDD vertex can represent two different matrices depending on which interpretation we use.

Having this extended structure, local operations become possible as illustrated in the following example.

Example 6. Consider a standard QMDD (without effective vertex weights) that is normalized according to a normalization scheme which operates on the standard interpretation of a vertex and ignores vertex weights (denoted by $\mathrm{N}_{\mathrm{std}}$ in the following). At the beginning, v-weight and standard interpretation match since all vertex weights are ineffective (Fig. 4a). Then, local modifications are applied (Fig. 4b). This can be a variable interchange, but anything that preserves structural equivalence is allowed. More precisely, we require that afterwards the same matrices are structurally equal as before. We will see later that variable interchanges indeed have this property. For now, we note that destroying structural equivalence will definitely require modifications on referencing vertices. Hence, in current implementations which do not store referencing (incoming) edges of a vertex this property is a necessary condition for local operations. As a by-product, this requirement ensures that we do not get vertices representing the same matrix in standard interpretation, but contain different weights.

Finally, the respective vertices are normalized according to $\mathrm{N}_{\mathrm{std}}$. But instead of propagating the normalization factor to all referencing edges, now this is stored locally in the vertex weight (see Fig. 4c).



$\mathrm{N}_{\mathrm{std}}(M_a) = 1$	$\mathrm{N}_{\mathrm{std}}(M_b) = \alpha$	$\mathrm{N}_{\mathrm{std}}(M_c) = 1$
$M_a = M_a^{\mathrm{v-weight}}$	$\mathrm{N}_{\mathrm{adjusted}}(M_b) := 1$	$M_c^{\mathrm{v-weight}} = M_b$
(a) Initial form	(b) Modifications	(c) With vertex weight

Fig. 4. Using vertex weights to restore normalization ($\mathrm{N}_{\mathrm{std}}$). $M_{\{a,b,c\}}$ or $M^{\mathrm{v-weight}}$ denotes the matrix represented by the particular vertex in standard or v-weight interpretation, respectively.

The crucial point is that the resulting QMDD structure is isomorphic to the canonical invariant structure of the equivalent standard representation, i.e. without effective vertex weights. To justify that, note that a QMDD vertex with an effective weight (Fig. 4c) can be transformed to a vertex that is equivalent in v-weight interpretation but has a weight of 1. For this purpose, we simply apply the effective weight to all outgoing edges of the considered vertex. This basically undoes the normalization and leads back to the vertices as in Fig. 4b. The resulting QMDD – now without effective vertex weights – has the same structure as before but is no longer normalized according to $\mathrm{N}_{\mathrm{std}}$. However, it can be viewed as normalized according to a normalization scheme $\mathrm{N}_{\mathrm{adjusted}}$ with "adjusted" normalization factors, i.e. $\mathrm{N}_{\mathrm{adjusted}}(\tilde{M}_v) = 1$ for all vertices v where $\tilde{M}_v$ denotes the standard interpretation of the vertex after the transformation (see Fig. 4b). Hence, this QMDD representation of the matrix has the canonical structure which is consequently also present in the QMDD with effective vertex weights.

Overall, the proposed extension maintains canonical representations even after local modifications as they are used by optimization techniques such as sifting.

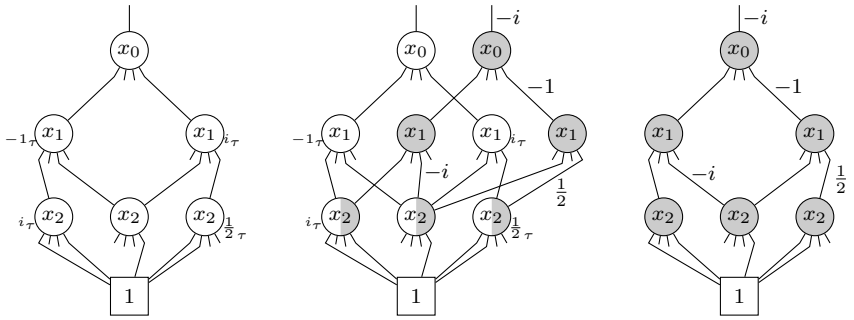
At the same time, a transformation back to trivial vertex weights is always possible. The only problem here is that we might need to replace some vertices with almost identical copies, but without an effective weight (see e.g. the x_2-vertices in Fig. 5).

This can be overcome by building an intermediate QMDD that is not functionally equivalent in v-weight interpretation, but represents the correct matrix in standard interpretation, i.e. when ignoring vertex weights. This intermediate QMDD can be obtained as follows:

Algorithm: "Build intermediate QMDD" for an edge e pointing to vertex v with weight w:

1. If v is the terminal vertex, return e unchanged.
2. Perform "Build intermediate QMDD" on all outgoing edges of v.
3. Create a vertex r (with $\tau_r = 1$) from the edges resulting from Step 2, normalize it and store the normalization factor π.
4. If r is identical to an already existing vertex (up to the vertex weight), the result is an edge pointing to that vertex with edge weight $w \cdot \pi \cdot \tau_v$.
   Else, the result is an edge pointing to r with the same edge weight.

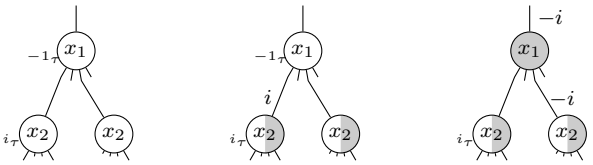
Remark 3. Note that this algorithm can also be used to switch between different normalization rules if we start with a QMDD in one normalization and use the other normalization in step 3 of the algorithm.



(a) Initial QMDD (with effective vertex weights)

(b) Intermediate QMDD

(c) QMDD without effective vertex weights



(d) Building the intermediate QMDD

Fig. 5. Computing equivalent QMDDs without effective vertex weights.

We illustrate the algorithm by the following example:

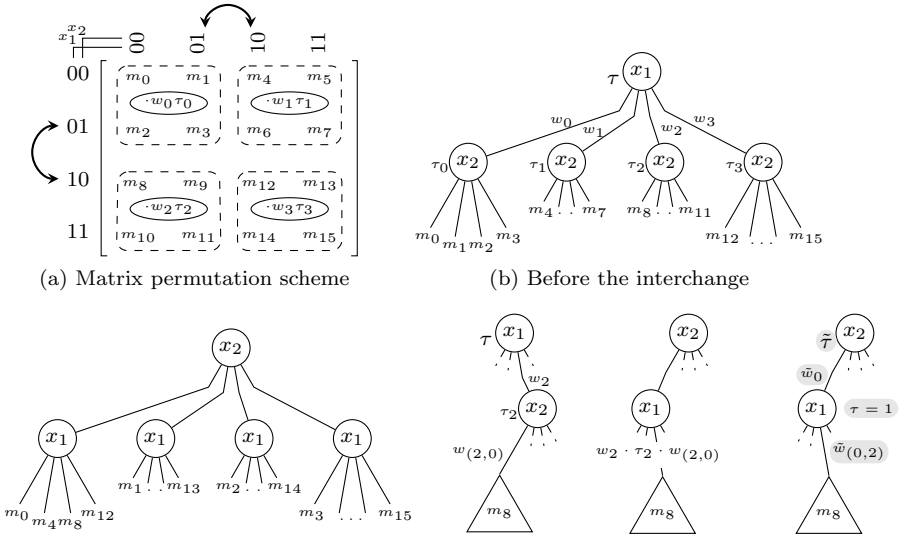
Example 7. The QMDD in Fig. 5a has a few effective vertex weights indicated by $(\cdot)_\tau$. In order to transform it to the equivalent representation without effective vertex weights, we compute the intermediate QMDD (highlighted in gray in Fig. 5b) which shares the x_2-vertices with the still valid initial QMDD representation. Figure 5d shows for the left x_1-vertex how our algorithm first pulls vertex weights from the child vertices to the respective outgoing edges before the new vertex is normalized and the normalization factor (i) and vertex weight (-1) are applied to the referencing edge.

The final QMDD (Fig. 5c) is obtained from the intermediate QMDD (Fig. 5b) by setting all vertex weights to 1.

5 Using Vertex Weights for Variable Interchange

As discussed above, 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. *Binary Decision Diagrams* (BDDs) [14]: 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 sub-trees representing the respective cofactors [14]. 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. 6a, i.e. the swapping of certain columns and rows.



(a) Matrix permutation scheme

(b) Before the interchange

(c) Structure after the interchange

(d) Refactoring edge- and vertex weights

Fig. 6. Sketch of the adapted variable interchange procedure for binary QMDDs.

This accordingly needs to be conducted in the QMDD structure [12, 13] in which each of the affected sub-matrices is represented by a vertex as well as weighted edges and vertices.

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 labelled 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 $T = (t_{ij})$ and set t_{ij} to be the j^{th} 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 labelled with x_2, set $t_{ij} = e_i^V$ instead.
2. From each column j of T create a vertex labelled 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.

Remark 4. We could also deal with an effective vertex weight at x_1-level by applying it to all edges in T, but since normalization would propagate this common factor back to its origin, we rather keep it and adapt it appropriately after relabelling.

This procedure is illustrated by the following example:

Example 8. Consider Fig. 6 showing a part of a binary QMDD ($r = 2$) in which both variables x_1 and x_2 should be interchanged. First, a matrix containing all sub-trees representing the sub-matrices m_0 until m_{15} is created according to Step 1 (see Fig. 6b). Then, these sub-trees are re-arranged in Step 2 eventually leading to the structure shown in Fig. 6c. Finally, the respective vertices are normalized in Step 3. This is illustrated in Fig. 6d for the sub-tree m_8. First, this sub-tree 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 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 allows for the use of effective vertex weights during the variable reordering process and – as discussed in the previous section – thereby allows for the determination of essential information about the QMDD structure without having to perform renormalization after each variable interchange. Hence, a large variety of objective functions (which we try to minimize by variable reordering) will give the same result for the intermediate variable orders as if we had transformed the QMDD to its normalized equivalent without effective vertex weights. We need to perform this potentially expensive transformation at most once, after we have arrived at the final variable order and only if there are effective vertex weights left. For this purpose we can use the algorithm presented in the previous section.

6 Application and Evaluation

The extension of the data-structure as described above has been implemented in C on top of the original QMDD package presented in [7]. In this section, we discuss and evaluate the application of the proposed vertex weights. For this purpose, we consider the task of equivalence checking of quantum circuit functionality. Equivalence checking is an important design task and aims e.g. for checking whether two circuits (the initial realization as well as an optimized version) realize the same functionality. This constitutes a representative application as characteristics like canonicity (for fast equivalence checking) as well as modifiability of the data-structure (allowing for a compact representation) are crucial here.

QMDDs have already been used for checking the equivalence of different quantum realizations of reversible Boolean functions [8]. However, we focus on functionality of general quantum computation like phase shifting, superposition, and entanglement [1] which requires various quantum values to be adequately represented in the QMDD. In this context an extended definition of equivalence is applied:

Definition 5. *Unitary transforms M_1 and M_2 of a quantum system are called equal up to* global phase *if $M_1 = e^{i\theta} M_2$ for some real number θ, where $e^{i\theta}$ is called the* global phase factor*.*

Remark 5. Classical equivalence is a special case for $e^{i\theta} = 1$. The reason for using this extended definition is that for global phase equivalent transforms it can not be physically distinguished which of the transforms has been applied to a quantum system since the outcomes have the same measurement statistics [1].

Verifying for global phase equivalence can easily be performed if canonical representations of the two functions are available. Canonicity ensures that global phase equivalent transforms have the same representation up to the weights of the root edges that differ by the global phase factor. Thus, it is sufficient to check whether (1) the root edges of the two QMDDs point to the same vertex and (2) the weights of these edges have the same magnitude. This can be performed in constant time using proper *unique tables*.

Table 1. Size reduction of QMDDs through variable re-ordering

Benchmark	Initial		Sifting		Exact	
	Size	Time (s)	Size	Time (s)	Size	Time (s)
Grover-7	187	0.01	36	<0.01	35	0.37
Grover-9	722	0.02	52	0.01	51	29.14
Grover-11	2817	0.15	67	0.02	66	3709
5-qubit-code-9	90	0.01	57	0.01	43	24.73
7-qubit-code-7	44	<0.01	26	<0.01	26	0.35
9-qubit-codeFigN1-9	40	<0.01	22	0.01	22	24.47
9-qubit-codeFigN2-17	1172	0.01	60	0.04	(84)	>7200
QFT-3	22	<0.01	9	<0.01	9	<0.01
QFT-4	86	<0.01	24	<0.01	24	0.01
QFT-5	342	<0.01	40	<0.01	40	0.01
QFT-6	1366	<0.01	103	<0.01	103	0.1
QFT-7	5462	0.02	167	0.02	167	1.2

First, we consider the applicability of the previously available QMDD-based approaches relying on the normalization rules as discussed in Section 3.1:

- QMDDs following the Normalization Rule 1 would allow for a fast check for equivalence as canonicity is ensured. However, the QMDDs would be restricted to the given initial variable order. Optimizations through variable re-ordering (e.g. using sifting which heavily relies on variable interchanges) could destroy the canonical, normalized structure as described in Example 3 or require to rework large parts of the data-structure many times. This lack of modifiability prevents this approach from deriving an efficient representation – a serious obstacle particularly for a task like equivalence checking where a major issue is to maintain a manageable diagram size while building the circuit representation.
- Normalization Rule 2 is not applicable as it does not guarantee canonicity of the representation as demonstrated before in Example 4. Here, an equivalence check would require a complete traversal of the entire data-structure and, hence, becomes computationally expensive. Moreover, the lack of canonicity can make it hard to find a good variable order for a compact representation. Once a promising variable order was found it might not be possible to reproduce the particular diagram (size) and we might end up with a significantly different representation.

In contrast, the proposed extended data-structure supports both, a canonical representation as well as an advanced modifiability. While the canonicity allows for a fast check for equivalence as described above, the modifiability ensures a compact representation of the respective functionality. This is also demonstrated by experimental results summarized in Table 1. Here, the respective QMDD sizes (i.e. the number of vertices; denoted by *Size*) for a selection of benchmark functions is presented if either (1) the initial variable ordering is applied, (2) if the data-structure has been improved through a heuristic approach (sifting technique), and (3) if an exact approach is applied that establishes the optimal variable ordering. As benchmarks, we applied circuits realizing Grover algorithms (*Grover-N*), error correction functionality (*k-qubit-code-N*, taken from [15]), and quantum Fourier transforms (*QFT-N*) where N denotes the number of qubits. Note that the quantum Fourier transforms actually do not show shared vertex compression in the standard variable order and, thus, exhibit the largest possible number of QMDD vertices ($\frac{4^N-1}{3}$ non-terminal vertices) for the respective matrix size. The run-time (in CPU seconds) is additionally provided in the columns denoted by *Time*. 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 size of the QMDD significantly depends on the applied variable ordering. Reductions of up to a factor of 42 (for the Grover-11 circuit) can be observed. This clearly emphasizes the necessity of a canonical, but also easily modifiable data-structure. While this has not been achieved for general quantum functionality with the previously introduced approaches, the proposed solution relying on vertex weights satisfies these needs.

7 Conclusions

In this paper, we proposed an extension to the QMDD data-structure by so called vertex weights. They provide a method supplemental to edge weights to represent common factors of sub-matrices composed of complex numbers. Vertex weights ensure a canonical representation and allow for an advanced modifiability and applicability of QMDDs – even for complex quantum functionality. An evaluation demonstrated how this can be exploited to improve the data-structure itself (e.g. through variable re-ordering enabled by the advanced modifiability) and how applications such as equivalence checking benefit from that.

Acknowledgments

We would like to sincerely thank D. Michael Miller for many helpful suggestions and discussions as well as for providing us with an implementation of the QMDD package introduced in [7]. This work was supported in part by the German Academic Exchange Service (DAAD).

References

1. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge Univ. Press (2000)
2. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Theory of computing. (1996) 212–219
3. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. Foundations of Computer Science (1994) 124–134
4. Dürr, C., Heiligman, M., Høyer, P., Mhalla, M.: Quantum query complexity of some graph problems. SIAM J. Comput. **35**(6) (2006) 1310–1328
5. Wang, S.A., Lu, C.Y., Tsai, I.M., Kuo, S.Y.: An XQDD-based verification method for quantum circuits. IEICE Transactions **91-A**(2) (2008) 584–594
6. Viamontes, G.F., Markov, I.L., Hayes, J.P.: Quantum Circuit Simulation. Springer, Dordrecht, Heidelberg, London, New York (December 2009)
7. Miller, D.M., Thornton, M.A.: QMDD: A decision diagram structure for reversible and quantum circuits. In: Int'l Symp. on Multi-Valued Logic. (2006) 6
8. Wille, R., Große, D., Miller, D.M., Drechsler, R.: Equivalence checking of reversible circuits. In: Int'l Symp. on Multi-Valued Logic. (2009) 324–330
9. Seiter, J., Soeken, M., Wille, R., Drechsler, R.: Property checking of quantum circuits using quantum multiple-valued decision diagrams. In: Reversible Computation. Volume 7581 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 183–196
10. Soeken, M., Wille, R., Hilken, C., Przigoda, N., Drechsler, R.: Synthesis of Reversible Circuits with Minimal Lines for Large Functions. In: Asia and South Pacific Design Automation Conference. (January 2012)
11. Bullock, S.S., O'Leary, D.P., Brennen, G.K.: Asymptotically optimal quantum circuits for d-level systems. Phys. Rev. Lett. **94** (Jun 2005) 230502
12. Miller, D.M., Feinstein, D.Y., Thornton, M.A.: Qmdd minimization using sifting for variable reordering. In: Journal of Multiple-valued Logic and Soft Computing. (2007) 537–552
13. Miller, D.M., Thornton, M.A.: Multiple-Valued Logic: Concepts and Representations. Morgan and Claypool (2008)
14. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. on Comp. **35**(8) (1986) 677–691
15. Mermin, N.D.: Quantum computer science: an introduction. Volume 1. Cambridge University Press (2007)