Quantum Information Processing (2025) 24:319
https://doi.org/10.1007/s11128-025-04917-0

A tutorial on applying quantum multiple-valued decision diagrams to circuit simulation

Asimakis Kydros[1] · **Konstantinos Prousalis**[1] · **Nikos Konofaos**[1]

Received: 10 February 2025 / Accepted: 18 August 2025
© The Author(s) 2025

Abstract
Quantum Multiple-Valued Decision Diagrams have proven their utility in efficiently representing and manipulating reversible and quantum functions. This work, acting as a mini-review of the topic, aims to provide a comprehensive guide for implementing reversible and quantum circuit simulators using such diagrams by integrating existing methodologies into a single cohesive framework. The operation and purpose of all relevant algorithms and routines are explained, and pseudo-code schematics are also detailed for each. A proof-of-concept implementation is presented at the end, made openly available to facilitate comprehension and research in the field of decision-diagram-based simulation of Quantum Computing.

Keywords Quantum multiple-valued decision diagrams · QMDD · Quantum computing · Quantum circuit simulator · Software development kit · QOLE

1 Introduction

Quantum Computing (QC) arose as a means to power through the nearing death of Moore's law and the subsequent performance limit imposed on classical computers. Quantum information science has proven itself capable of exceeding classical machines in certain problems, most notably in database search with Grover's algorithm [14] and Shor's polynomial prime factoring [42], thus exhibiting the prospect of Quantum Supremacy. Still, current quantum hardware offers limited resources due to noise effects, and thus, the classical simulation of QC remains a highly relevant research tool.

✉ Asimakis Kydros
akydros@csd.auth.gr

Konstantinos Prousalis
kprousalis@csd.auth.gr

Nikos Konofaos
nkonofao@csd.auth.gr

[1] Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece

Published online: 25 September 2025

Owing to its exponential scaling due to the superposition principle [32], the conventional mathematical framework of QC based on unitary matrices and linear algebra cannot be efficiently simulated in classical computers, which sets a limit on the size of quantum functions that can be studied. As a matter of fact, this problem plagues not only QC, but also Reversible Computing (RC), binary or multiple-valued. For this reason, Quantum Multiple-Valued Decision Diagrams (QMDDs) were introduced [28, 29], as a means to compactly represent unitary matrices by exploiting the repeated submatrix patterns and values commonly found inside them. From a simulation perspective, decision diagrams (DDs) like QMDDs are a powerful and widely used tool for representing quantum circuits, as these circuits are composed of combinations of a small number of quantum operators and often exhibit repeating patterns well suited to a DD-based representation.

Several works have been published on QMDDs, using them in logic synthesis [7, 9, 27, 34, 43, 44, 56], equivalence checking [3, 35, 51], refining their definition and performance [8, 10–12, 19, 26, 30, 33, 36, 37, 41, 54, 55], or employing them in other real world applications [39, 40, 45, 46]. As such, the algorithmic pieces relevant to computing simulation are scattered throughout the literature, and their composition can be quite difficult to parse. For this reason, this work aims to compile them all in a single tutorial and guide the reader through their practical implementation.

The remainder of this paper is structured as follows. A short preliminary introduction of Quantum And Reversible Computing is given in Section 2. Section 3 details the theoretical structure of QMDDs and analyzes their realization into code, their normalization and the smart handling of their weights. The method of their construction from circuit blueprints is focused upon in Section 4. The extraction of the circuit output is studied in Section 5, and Section 6 introduces the open-source implementation package. Afterward, Section 7 gives a short overview of extensions to the QMDD structure. Finally, conclusions and future work are discussed in Section 8.

2 Preliminaries

A very quick summary of Reversible and Quantum Computing is given in this section, to ensure this work's modularity. For a more comprehensive introduction, the reader is kindly redirected to [5, 32].

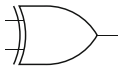
2.1 Reversible computing

A function is called reversible if it represents a bijective mapping, in other words, if it establishes an $1 : 1$ correspondence between inputs and outputs. Conventional computing is inherently irreversible because it generally discards information on each computational step. The truth tables in Fig. 1 highlight this fact: The simple NOT gate is logically reversible, because $0 \rightarrow 1$ and $1 \rightarrow 0$. XOR, however, is not, because one cannot know for certain what the input was given the output.

In [25], this irreversibility is shown to cause a minimum heat dissipation in the system. Based on this effect, [1] posited that computation based on reversible operations

In	Out
0	1
1	0

(a) The classical NOT gate

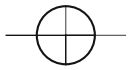


In	Out
00	0
01	1
10	1
11	0

(b) The classical XOR gate

In	Out
00	00
01	11
10	10
11	01

(c) The reversible XOR (CNOT)



In	Out
0	1
1	0

(d) The reversible NOT

Fig. 1 Classical and reversible logic gates and their truth tables. The NOT gate remains logically reversible in both versions, since the incoming bit gets flipped in the output. XOR, however, discards one of the inputs and outputs a single bit, leading identical outputs being resulted from different inputs. CNOT, the reversible XOR, solves this problem by introducing an unchanging ancilla bit in the output

can offer useful performance at arbitrarily low energy cost. Thus, RC remains relevant as a novel and potentially beneficial alternative to today's computers.

In RC, information is represented as vectors in d^t-dimensional space, where t is the number of digits (called *dits*) and d is the number of unique values each digit can take (the system radix). Correspondingly, reversible functions acting on t dits are represented as $d^t \times d^t$ permutation matrices, called *gates*. For the binary case, for example, the binary digit (bit) has logical states "0" $= \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, "1" $= \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and the two reversible analogues of NOT and XOR shown in Fig. 1 are defined by the matrices shown in Eq. (1).

$$NOT = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1}$$

CNOT, the analogue of XOR, affects two bits, one control (•) and one target (⊕), with the target bit flipping its state if the control bit has a satisfactory value ("1"). This gate can be generalized to $m \geq 1$ controls, each activating on either "1" or "0" (∘). This gate collection is called the Toffoli family. As is evident, an m-Toffoli gate is represented by a $2^{m+1} \times 2^{m+1}$ matrix, which grows in memory extremely fast.

2.2 Quantum computing

QC is a kind of Reversible Computing because the evolution of a closed quantum system is unitary and thus inherently reversible [32]. Of course, this ignores practical considerations such as decoherence which are outside the scope of this paper.

In QC, information is represented through quantum systems, the basis states of which act as the information carriers. Although multiple-valued quantum systems can be defined, here only the binary case is considered. The quantum bit (qubit) is a two-state system, with its basis states written as $|0\rangle, |1\rangle$ [6], connecting them to logical "0" and logical "1", respectively. As a quantum object, the general state of a qubit is

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$$

$$C^m(X) = \begin{bmatrix} I_{2^{m+1}-2} & \mathbf{0} \\ \mathbf{0} & X \end{bmatrix}$$

Fig. 2 Common quantum gates. The top 6 operate on single qubits. The gates X, Y and Z, alongside the 2×2 identity matrix I, form the Pauli set. Further note the relation $Z = S^2 = T^4$. The bottom gate represents the entire Toffoli family, with $C^1(X)$ being the aforementioned $CNOT$ and $C^2(X)$ being the most often used Toffoli gate, a NOT with 2 controls. One could consider X as a variable here and use this formula to define controlled versions for the other gates as well

a *superposition* between these two basis states, written as in Eq. (2).

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \quad \alpha, \beta \in \mathbb{C}. \tag{2}$$

The manipulation of the superposition of a system allows for all basis states to be operated upon simultaneously. This scales exponentially with groupings of qubits, called *registers*, taking the form of Eq. (3).

$$|\psi\rangle = \sum_{i=0}^{2^n-1} c_i|i\rangle, \quad c_i \in \mathbb{C}. \tag{3}$$

The vector $|\psi\rangle$ is called the system's *state vector*, and it holds the output after each computational step, mapping each basis state $|i\rangle$ to its probability amplitude c_i.

Just like in RC, QC represents quantum functions as quantum gates, which operate on one or multiple qubits and are defined by square matrices. One notable difference is that these matrices no longer have to be permutation matrices, so long as they are *unitary*, meaning they satisfy $UU^\dagger = U^\dagger U = I$, where $U^\dagger$ is U's conjugate transpose. An infinite number of quantum gates can be defined, all performing an arbitrary change of the amplitudes α, β, so long as $|\alpha|^2 + |\beta|^2 = 1$ stays true. For this reason, only finite subsets of gates are considered in each work, called *gate-sets*, which must be defined and be provably universal according to the Solovay–Kitaev theorem [32].

An example gate-set consisting of the most common quantum gates is shown in Fig. 2. The sub-collection of $\{CNOT, H, S, T\}$, known formally as the Extended Clifford set (Clifford+T), is known to be universal [32]. For the rest of this work, the term *gate-set* refers to the full collection of single-qubit gates shown in Fig. 2, alongside any controlled version thereof (e.g., CZ).

A *computational step* in QC consists of multiple such gates operating on different qubits, and a *quantum circuit* is a cascade of such steps. To form a step, the member gates are merged using the *tensor product*. For example,

$$X \otimes X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & 1 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ 1 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & 0 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

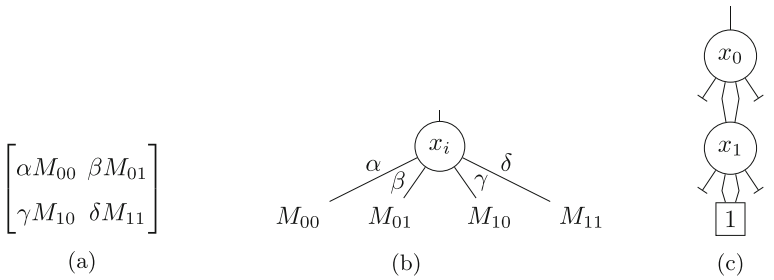


Fig. 3 Examples of matrix QMDDs. **b** shows the recipe for creating QMDDs from the generic matrix (**a**). M_{oi} are the quadrants and they can be further composed into QMDDs recursively. The indices of the submatrices reveal the I/O mapping for the represented (qu)bit: $|i\rangle \rightarrow |o\rangle$. The common scalar of each quadrant is placed onto the edge as weight. In **c**, the QMDD of the matrix in Eq. (4) is shown. One can notice that the vertex x_1 represents the NOT matrix and the vertex x_0 represents a matrix where the diagonal quadrants are the zero matrix and the other quadrants are the NOT matrix

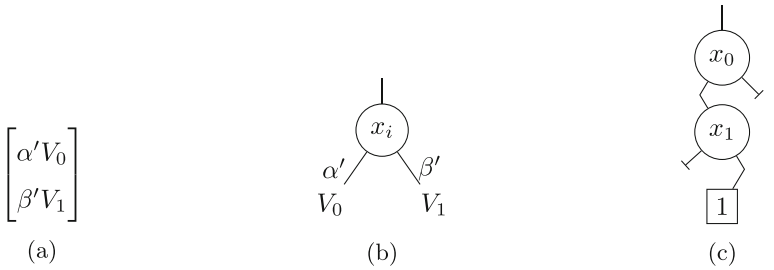


Fig. 4 Examples of vector QMDDs. **b** shows the recipe for creating QMDDs from the generic vector (**a**). Similarly to matrix QMDDs, the common scalar of each quadrant is placed onto the edge as weight. In **c**, the vector QMDD of the state $|10\rangle$ is shown. Based on the vector definitions $|0\rangle = [1 \quad 0]^T$, $|1\rangle = [0 \quad 1]^T$, one can notice that the edges of x_0 (x_1) follow $|0\rangle$ ($|1\rangle$)

These steps are then multiplied together in order of appearance. Thus, a quantum algorithm is $\mathcal{O}(2^n)$ for n qubits. This scaling highlights the problem of simulating QC mechanics on classical computers, as such memory requirements are untenable.

3 Quantum multiple-valued decision diagrams

3.1 The general structure

The *Quantum Multiple-Valued Decision Diagram* structure was conceived in [28] to combat the previously stated exponential scaling of the matrix representation. QMDDs actively exploit the repeating patterns caused by tensor product arithmetic. Consider Eq. (4) again. Splitting the final matrix into the four 2×2 quadrants, the two matrices $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ are repeated twice each. Splitting these two again in the same fashion leads to just two scalars, 0 and 1, being the final and only unique information pieces that are being recycled throughout the matrix. A recursive relationship can thus be

defined, where a large matrix points only to its quadrants, and those quadrants to their quadrants, which can very much be shared between them, until scalars are reached [12, 28, 29, 33, 36, 37, 57]. The formal definition of a QMDD is outlined in Def. 1, according to [28, 36, 41, 57].

Definition 1 A QMDD is a directed acyclic graph (DAG), such that:

1. There is a single source and a single sink.
2. Every edge in this DAG is weighted by a complex value called the edge **weight**.
3. The sink vertex is called the **terminal**, it has no outgoing edges, and it represents the scalar value 1. Graphically, it is notated as a square labeled "1."
4. All other vertices have r^2 (r) outgoing edges, for QMDDs representing matrices (vectors), pointing to other vertices. Here, r is the radix of the system. For the binary case, $r = 2$, and thus, each non-terminal matrix (vector) vertex has 4 (2) outgoing edges.
5. The edge pointing to the source vertex is called the **entry**, and it is unique.
6. Each vertex is assigned a **variable** index, linking it to the (qu)bit it represents.
   - The terminal has no defined variable.
   - The variable order must be common between vectors and matrices.
   - These variables follow a strict and predefined **order**. This order must be respected on every path, meaning that each vertex can only point to a variable preceding it or to the terminal, and every variable must appear on each path from the source to the sink at most once. Here, $x_{n-1} \prec x_{n-2} \prec \cdots \prec x_0$ order is used, meaning that smaller indices are closer to the entry than the terminal. The reverse order is also popular, $x_0 \prec x_1 \prec \cdots \prec x_{n-1}$, meaning indices decrease the closer, the variables are to the terminal.
7. No vertex is **redundant**, meaning that no vertex has all its edges point to the same destination with the same weight. Vertices resembling the identity (the outer edges pointing to the same destination with the same weight and the inner ones pointing to the terminal with weight 0) are also considered redundant. Vertex redundancy affects only matrix QMDDs.
8. Vertices are **unique**. There cannot be two vertices sharing the same variable and edges.
9. Vertices are **normalized** in terms of the weights of their outgoing edges.

The application of the above definition, for both matrices and vectors, is visualized in Figs. 3 and 4, respectively. The typical convention in drawing QMDDs posits that unlabeled weights are implied to be 1 and 0-weight edges are drawn as stubs. Edges with weight 0 will be called *zero edges* from now on, and they always point directly to the terminal.

The recursive nature of Def. 1 warrants a class implementation for the QMDD structure. Modeling each vertex as a matrix (vector) "QMDD," its main properties are easily identified as **variable** and **edges**, of cardinality 4 (2). Another useful property is a unique **identifier** for each vertex, either an integer or the memory address of the object itself, if accessible by the programming language chosen. This is needed for caching vertices, thus minimizing their number by enforcing Rule (8). An easy way to link the vertex definition with the identifier is to hash the variable and edges of

the vertex together and map it to the identifier using a hash table, as by the previous Rule these pairs should be unique across vertices. Thus, the class looks something like Alg. 1.

Algorithm 1: QMDD Vertex Class Definition

Property: $variable$
Property: $edges$
Property: id
Constructor
  Data: $x, e : \{e_0\ e_1\ (e_2\ e_3)\} \vee \emptyset$
  $variable \leftarrow x$
  $edges \leftarrow e$
  Perform normalization
  Perform trivialization
  (Optional) Do other initialization routine
  if *not unique* **then return** *the previously saved instance*
  Cache the new instance
  Assign a unique id
end

The Edge structure is even simpler. From Def. 1, it needs only hold the **weight** and the **destination**, a reference to the pointed-to vertex. One could try to merge the Vertex and Edge classes into a single class, but this needlessly complicates important algorithms such as multiplication and addition, which recursively make use of a single edge as the entry. The simplicity of Edge can instead be leveraged as a container for useful subroutines to be used in the rest of the implementation, as shown in Alg. 2.

Algorithm 2: QMDD Edge Class Definition

Property: $weight$
Property: $destination$
Constructor
  Data: w, d
  $weight \leftarrow w$
  $destination \leftarrow d$
end
Function $x(e)$: Returns the variable of the vertex pointed by e
Function $T(e)$: Returns $true$ if e points to the terminal
Function $w(e)$: Returns $weight$
Function $v(e)$: Returns $destination$
Function $E_i(e)$: Returns the i-th edge of the vertex pointed by e

3.2 Normalization

Rule (9) of Def. 1 dictates that each vertex ought to be normalized. *Normalization* in the sense of edge weight factorization is beneficial in maximizing vertex sharing

[33, 36, 54, 55] and ensuring the canonicity of the QMDD representation for each reversible function, enabling constant-time equivalence checking between circuits [3, 35, 51]. How this is to be done is a subtle matter. Four different normalization rules can be found in the literature:

NR1 *Ensure that the first nonzero weight is exactly* $+1$ *by factoring out its weight from all other edge weights. This factor is then multiplied to all incoming edge weights*. This ensures canonicity [28] and has enjoyed wide application [28, 29, 35, 36, 51]. Although simple and straightforward, it is known to break on local variable reordering [31, 33]. Also, it runs the risk of severe numerical inaccuracies, as this greedy division and propagation of the first viable number can lead to the weights either becoming extremely large or extremely small; as explained in [54], if, for example, the weight of an edge shrinks close to 0 beyond the considered precision threshold ϵ, then the pointed-to vertex and all its children will be pruned away from the diagram, leading to loss of information.

NR2 *Ensure that the edge weight with the largest magnitude is exactly* $+1$. This rule alleviates the uncontrolled growth concern imposed by NR1, as it constricts the real and imaginary parts of all weights inside the interval [0, 1] [54]. It is also compatible with local variable reordering [31, 33]. However, since multiple edges of a vertex could exhibit the maximum magnitude, this rule is no longer canonical [33, 54].

NR3 *Ensure that no edge weight has a magnitude larger than* $+1$ *and the first edge weight that has the maximum magnitude is exactly* $+1$. An extension of NR2, this idea reintroduces canonicity while retaining the value range to [0, 1], thus achieving the best of both previous rules [33, 54]. However, its compatibility with local variable reordering remains imperfect [33].

NR4 *Divide all weights by the norm of the complex vector formed by the edge weights, and adjust incoming edges accordingly*. This idea was introduced in [19], and it mainly relates to vector QMDD vertices only. Normalizing in this way fixes the sum of the squared weights to 1. As a result, edge weights are semantically enriched, as now the squared magnitude of each weight corresponds to the probability of the qubit represented by the vertex collapsing to the respective basis state ($|w(e_0)|^2$ for $|0\rangle$ and $|w(e_1)|^2$ for $|1\rangle$). This interpretation becomes useful during output simulation, see Sec. 5.2.

In [33], the concept of vertex scalars is first introduced. This extension thwarts the propagation of the factor of the edge weights to the incoming edges, the very thing that causes variable reordering to break normalization, by storing it inside the vertex instead. This trick enables the usage of a normalization rule that enforces canonicity for variable reordering.

Local variable reordering is a technique that aims to change the variable order locally throughout the diagram in order to achieve further minimization of the representation [7, 11, 26, 31, 36]. While very interesting and useful, it makes little sense to implement this optimization method in the context of a simulator. Each circuit diagram remains in memory only until it is fully traversed, and the length of this traversal should never change as it depends directly on the output of the circuit in question. Another reason as to why this minimization is pointless is that creating a "bad" representation

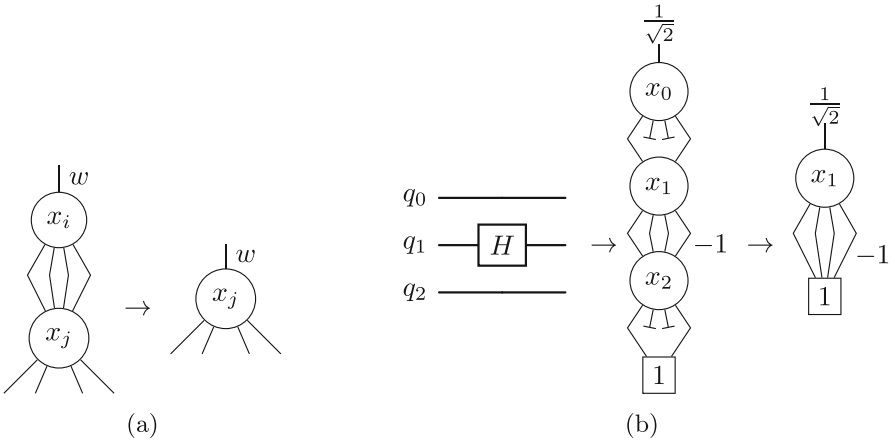


Fig. 5 Trivialization examples. **a** The x_i variable produces a vertex with all edges pointing to the same x_j vertex with weight w. This weight travels to the entry through normalization. Vertex x_i can be completely eliminated. **b** Matrix QMDD representation of a 3-qubit step applying the Hadamard gate on qubit 1. A full representation requires 3 vertices, 2 of which (x_0 and x_2) are identities. The final shortening produces a much more compact diagram. The same logic extends to multiple idle qubits, massively saving on resources

is step 1 in this procedure, and therefore, if a representation is too memory-heavy for the machine, it is already too late.

Even without having to care about reordering, NR3 remains the safest choice, at least for matrix vertices. It maintains canonicity while thwarting the generation of numerical errors as much as possible [54]. Depending on the implementation of output simulation, vector vertices could also utilize the same normalization rule, or flock to NR4. Normalization should happen inside the Vertex constructor, crucially *before* checking the cache.

3.3 Trivialization

QMDD matrix construction algorithms, explained in Sec. 4, can potentially lead to a vertex pointing unilaterally to the same descendant with the same weight. At the same time, the identity matrix very frequently appears in quantum circuits, and in great numbers. This is because the quantum gates most commonly used affect only few qubits directly, and the rest must therefore participate in the gate application in a way that leaves their states unchanged [32]. Thus, vertices representing such cases introduce unnecessary clutter to the QMDD representation [41].

Rule (7) of Def. 1 exists to eliminate these cases by bypassing these vertices completely. This work dubs this process *trivialization*. Trivialization is a very quick and simple augmentation to do on a vertex that further enriches the compactness of the resulting diagram [41]. Algorithm 3 elucidates its function. Importantly, this operation should happen *after* Normalization has been performed. An example of the effect can be found in Fig. 5.

An important observation to make here is that the enforcement of Rule (7) potentially introduces skipped variables in the diagram [8, 10, 31, 36, 41]. Simply put, this means that an edge path skips one or more members of the variable order. This fact has implications on circuit construction, but it is combated through QMDD addition and multiplication, described in Sec. 4.

Algorithm 3: Trivialization

Data: $e = (v, w)$, v representing a matrix
Result: e', pointing to a non-redundant vertex
if $T(e)$ **then return** e
$\{e_0, e_1, e_2, e_3\} \leftarrow \{E_0(e), E_1(e), E_2(e), E_3(e)\}$
$case1 \leftarrow \bigwedge_{i=1,2,3}(v(e_i) = v(e_0)) \wedge (w(e_i) = w(e_0))$
$case2 \leftarrow (v(e_3) = v(e_0)) \wedge (w(e_3) = w(e_0)) \wedge (e_1, e_2$ zero edges$)$
if $case1 \vee case2$ **then**
  return (e_0, w)
else
  return e
end

3.4 Complex weight handling

A surprisingly "complex" issue in QMDD realization to code is how to handle complex numbers. Since QC is involved, the edges of the diagram have complex number weights in general. Most programming languages do not have native support for complex arithmetic, but even those that do (like Python), or external libraries that introduce this extra behavior, use an implementation which is essentially a pair of floating point numbers wrapped inside syntactic sugars. This approach most closely resembles reality and maximizes the numbers that can be represented. However, it comes with loss of precision, due to floating point representation and arithmetic. This is crucial for QMDDs because the weights participate in the hashing of each vertex, meaning that due to this loss it could be that two equivalent (up to a constant scalar) matrices hash to different vertices, leading to loss of compactness [54, 55].

3.4.1 Integer-based Handling

For suitable gate-sets, the formulation of a complex number can be re-imagined in order to maintain its precision. A common way to do this, that complements Clifford+T (as well as appropriate super-sets, like the gate-set considered here) is to map it to an integer tuple, as in Eq. (5) [29, 55].

$$z = \frac{a_1 + b_1\sqrt{2}}{c_1} + \frac{a_2 + b_2\sqrt{2}}{c_2}i, \quad a_j, b_j, c_j \in \mathbb{Z} \tag{5}$$

Defining integer coefficients as such allows for all relevant operations between complex numbers to happen in the integer regime. This frees the program from having to

invoke floating point arithmetic until the very end, when the real and imaginary parts are desired, thus maintaining precision throughout the procedure. Another popular choice similar to Eq. (5) is the one shown in Eq. (6) [4, 47, 55].

$$z = \frac{1}{\sqrt{2}^k}(a\omega^3 + b\omega^2 + c\omega + d), \quad \omega = \frac{1+i}{\sqrt{2}}, \quad a, b, c, d, k \in \mathbb{Z} \tag{6}$$

Of course, this is not universal; not *all* complex numbers can be written as in Eq. (5) or (6). However, all complex numbers that appear in Clifford+T can. Since that set is universal, any desired state can be approximated at arbitrary precision [32, 55]. Thus, universal quantum computation is maintained, without sacrificing the compactness of QMDDs.

It should be noted here that these integers can grow very large. From an implementation perspective, if this growth is expected to breach normal integer bidwidths, then the representation would demand application of multiple-precision libraries (BigNum) for handling arbitrarily large integers. In such cases, the integer scheme loses its value, because BigNum operations are considerably slower than standard floating point arithmetic.[1] Moreover, a trade-off in speed is expected even in the average case when compared to "raw" floating point, because even though every complex manipulation consists of integer operations that are (typically) faster than floating point ones, an increased amount of them are performed for each manipulation.

Despite its aforementioned limitations, the integer scheme continues to find use [4, 47]. This work considers a slight alteration that nonetheless remains equivalent to Eqs. (5) and (6), shown in Eq. (7).

$$z = \frac{A + \frac{B}{\sqrt{2}} + Ci + \frac{D}{\sqrt{2}}i}{E}, \quad A, B, C, D, E \in \mathbb{Z}, \quad E > 0 \tag{7}$$

All relevant operations (addition, multiplication, and division) remain in the integer regime. Although somewhat complex and not immediately intuitive, they follow directly from the formal mathematical definitions of these operations. The procedures are presented in Algorithms 4, 5, and 6, and their proofs are given in Appendix A.

Following [29], common complex numbers are cached in convenient integer positions to facilitate their easier invocation. In particular, $0 + 0i$ and $1 + 0i$ can be bound to positions 0 and 1, respectively, so that comparisons and assignments using them can be done through the position index itself. Table 1 shows the 9 complex numbers that appear in the chosen gate-set and are cached from the beginning, alongside their bound index.

Further caching happens dynamically, both for newly created numbers and for results of operations. A hash map should be devoted to each operation, linking the indices of the arguments with the index of their result, and a bidirectional mapping structure can be used for saving number definitions, linking the quintuples with their indices and vice-versa, ensuring $\mathcal{O}(1)$ access to all necessary information at all times.

[1] This is because, in opposition to standard floating point that can be performed at the hardware level, BigNum is implemented in software [23].

Algorithm 4: Complex Addition with Integers

Data: $(A_z, B_z, C_z, D_z, E_z), (A_t, B_t, C_t, D_t, E_t)$,
$A_i, B_i, C_i, D_i, E_i \in \mathbb{Z}, E_i > 0$
Result: The sum $(A_s, B_s, C_s, D_s, E_s)$
if *the operation is cached* **then return** *the old result*
if $index(z) = 0$ **then return** $(A_t, B_t, C_t, D_t, E_t)$
if $index(t) = 0$ **then return** $(A_z, B_z, C_z, D_z, E_z)$
$A_s \leftarrow E_z A_t + E_t A_z$
$B_s \leftarrow E_z B_t + E_t B_z$
$C_s \leftarrow E_z C_t + E_t C_z$
$D_s \leftarrow E_z D_t + E_t D_z$
$E_s \leftarrow E_z E_t$
Cache the result
return $(A_s, B_s, C_s, D_s, E_s)$

Algorithm 5: Complex Multiplication with Integers

Data: $(A_z, B_z, C_z, D_z, E_z), (A_t, B_t, C_t, D_t, E_t)$,
$A_i, B_i, C_i, D_i, E_i \in \mathbb{Z}, E_i > 0$
Result: The product $(A_p, B_p, C_p, D_p, E_p)$
if *the operation is cached* **then return** *the old result*
if $index(z) = 0 \vee index(t) = 1$ **then return** $(A_z, B_z, C_z, D_z, E_z)$
if $index(z) = 1 \vee index(t) = 0$ **then return** $(A_t, B_t, C_t, D_t, E_t)$
$A_p \leftarrow 2A_z A_t + B_z B_t - 2C_z C_t - D_z Dt$
$B_p \leftarrow 2(A_z B_t + B_z A_t - C_z D_t - D_z C_t)$
$C_p \leftarrow 2A_z C_t + 2C_z A_t + B_z D_t + D_z B_t$
$D_p \leftarrow 2(A_z D_t + D_z A_t + B_z C_t + C_z B_t)$
$E_p \leftarrow 2E_z E_t$
Cache the result
return $(A_p, B_p, C_p, D_p, E_p)$

Algorithm 6: Complex Division with Integers

Data: $(A_z, B_z, C_z, D_z, E_z), (A_t, B_t, C_t, D_t, E_t)$,
$A_i, B_i, C_i, D_i, E_i \in \mathbb{Z}, E_i > 0$
Result: The quotient $(A_q, B_q, C_q, D_q, E_q)$
if $index(t) = 0$ **then** throw an error or handle some other way
if *the operation is cached* **then return** *the old result*
if $index(z) = 0 \vee index(t) = 1$ **then return** $(A_z, B_z, C_z, D_z, E_z)$
if $index(z) = index(t)$ **then return** $(1, 0, 0, 0, 1)$
$\alpha \leftarrow 2(A_t^2 + C_t^2) + B_t^2 + D_t^2$
$\beta \leftarrow A_t B_t + C_t D_t$
$\gamma \leftarrow 2(A_z A_t + C_z C_t) + B_z B_t + D_z D_t$
$\delta \leftarrow 2(A_z B_t + B_z A_t + C_z D_t + D_z C_t)$
$\epsilon \leftarrow 2(C_z A_t - A_z C_t) + D_z B_t - B_z D_t$
$\zeta \leftarrow 2(C_z B_t + D_z A_t - A_z D_t - B_z C_t)$
$A_q \leftarrow E_t(\alpha\gamma - 2\beta\delta)$
$B_q \leftarrow E_t(\alpha\delta - 4\beta\gamma)$
$C_q \leftarrow E_t(\alpha\epsilon - 2\beta\zeta)$
$D_q \leftarrow E_t(\alpha\delta - 4\beta\epsilon)$
$E_q \leftarrow E_z(\alpha^2 - 8\beta^2)$
Cache the result
return $(A_q, B_q, C_q, D_q, E_q)$

Table 1 Caching of the 9 most common complex numbers in the chosen gate-set. Complex 0 and 1 are bound to index 0 and 1, respectively, so they can be used through the index only

Real representation	As $\mathbb{Z}$-quintuple	Index
$0+0i$	$(0, 0, 0, 0, 1)$	0
$1+0i$	$(1, 0, 0, 0, 1)$	1
$\frac{1}{\sqrt{2}}+0i$	$(0, 1, 0, 0, 1)$	2
$-1+0i$	$(-1, 0, 0, 0, 1)$	3
$0+i$	$(0, 0, 1, 0, 1)$	4
$0-i$	$(0, 0, -1, 0, 1)$	5
$-\frac{1}{\sqrt{2}}+0i$	$(0, -1, 0, 0, 1)$	6
$\frac{1}{\sqrt{2}}+\frac{1}{\sqrt{2}}i$	$(0, 1, 0, 1, 1)$	7
$\frac{1}{\sqrt{2}}-\frac{1}{\sqrt{2}}i$	$(0, 1, 0, -1, 1)$	8

To ensure a canonical representation, each tuple is normalized by dividing all five integers by their greatest common divisor, with the denominator E always enforced to be positive.

3.4.2 Tolerance-based Handling

In order to avoid the limitations introduced by the previous approach, a sophisticated strategy involving floating point numbers has also been proposed [54]. This scheme leverages the fact that, by normalizing via NR3, the real and imaginary parts of all complex numbers are constrained in the range [0, 1]. This range is then split into N equally sized sub-ranges

$$[0, \frac{1}{N}), \quad [\frac{1}{N}, \quad \frac{2}{N}), \quad \cdots \quad [\frac{N-1}{N}, 1] \tag{8}$$

and a lookup table of N entries, each representing one such sub-range, is defined. Each entry is a linked list, containing real-valued elements belonging in the described range in order. Then, all complex numbers are split into their real and imaginary parts, and those are saved into this lookup table as independent, absolute-valued numbers. Doing so offers further opportunity for value recycling, since complex numbers might share parts numerically. Following this logic, each complex number can be represented as a pair of pointers to its elements in the lookup table. This preserves representation canonicity, because there is a unique pointer pair for each complex value. Sign information can also be maintained efficiently.

Each time a weight operation is required, its respective numerical information is retrieved from the lookup table via the pointers. Afterward, normal floating point arithmetic can occur. In order not to litter the lookup table with one-time values, it is updated on the very last step and after the normalization of the new weight. This is also when the compactness preservation aspect of the approach shines. For each real-valued part of the new complex number, the respective sub-region is consulted. If at least one element is found with difference less than some predefined tolerance ϵ,

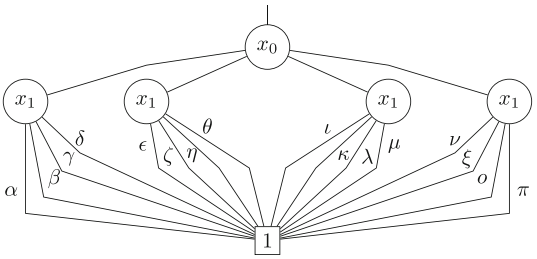


Fig. 6 A maximal QMDD for two variables. Such a diagram occurs from a matrix containing the elements $\alpha - \pi$, all being different from one another (the worst-case example). Thus, since no repeating patterns emerge, each edge leads to a new maximal sub-diagram. For more variables, this pattern would extend upward, growing exponentially

then that old element's pointer is returned, effectively rounding the complex number. Otherwise, the new part is inserted into the list in the correct place (by order). This further minimizes the amount of numbers in the diagram, thus enforcing compactness for small numerical deviations [54].

This strategy achieves a nice balance in maintaining both performance and compactness. It has shown significant performance improvements in decision diagram-based simulation [54], making it a strong choice in this context. However, the unavoidable accuracy loss induced by floating point representation renders it less appropriate for cases where exact precision is required.

4 Construction

A pivotal point in creating a circuit simulator using QMDDs is how to create the desired circuit optimally. In Section 3.1, a rough outline of how to map a matrix into a QMDD was drawn, but this is not practically useful because to create the matrix, $\mathcal{O}(2^n)$ space is required for n qubits, defeating the purpose of using QMDDs. To avoid the exponential ceiling, the final QMDD must result from a collection of small gates, built iteratively.

Of course, QMDDs do not guarantee total protection from exponential scaling. Indeed, matrices with no exploitable structure will lead to graphs resembling the one shown in Fig. 6, with exponentially many vertices. Examples include the cluster states—highly entangled quantum states influential in measurement-based quantum computation and fault tolerance [2, 38]—which are known to require exponential QMDD representations [48]. Nevertheless, in many practical quantum algorithms, the unitary matrices involved exhibit structural redundancy, rendering QMDDs useful.

Construction can be subdivided into two main parts: gate construction and gate application. Each gate in a circuit, whether a single-qubit or a multi-qubit one, can be translated into a matrix QMDD in cost polynomial to the amount of qubits it affects. Then, starting from a vector QMDD implementing the initial state (e.g., all qubits to $|0\rangle$), each gate QMDD can be multiplied onto the vector QMDD, leading step by step to the final state vector QMDD representation.

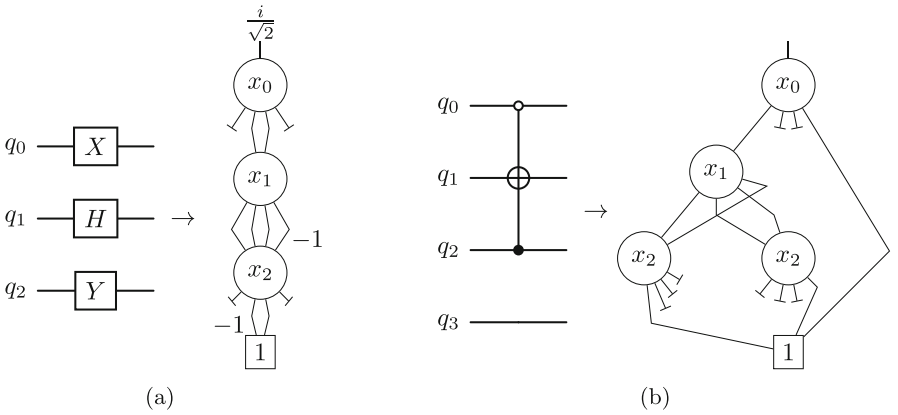


Fig. 7 Example of constructing gates into matrix QMDDs. **a** shows the construction of a non-controlled step using Alg. 8. Each vertex' edge weights follow the corresponding gate definition, and the total scalar gets normalized to the entry. **b** shows the construction of a controlled gate, using Alg. 7. Notice how the unused line, as well as the entire circuit for unsatisfying control values, is mapped to the identity gate implicitly by skipping variables

4.1 Defining the circuit blueprint

The first stage in defining construction is articulating how the desired circuit is to be described. There is probably a myriad of ways to do this. A practical approach is to allow the user to declare gates on qubit indices, through methods on a circuit object, an idea employed by modern popular simulators [21, 50]. Through these methods, the gate specifications and order can either be recorded inside a data structure, to be constructed and applied all at once, or each gate can be constructed and applied to the vector immediately, in a first-come-first serve fashion. The latter approach seems simpler and straightforward, however the former has advantages when circuits with intermediary measurements are considered. This is explored further in Sec. 5.

4.2 Gate construction

Whether or not a desired gate is controlled and in what capacity affects the construction procedure. An efficient strategy in gate translation to matrix QMDDs is to generalize the concept of a controlled gate to the uncontrolled ones, as "controlled" gates with zero controls. Doing so allows for a unified construction algorithm.

In quantum circuits, a gate can be controlled in multiple ways. Controls can exist *above* the target qubit (on smaller indices following the selected order $x_{n-1} \prec x_{n-2} \prec \cdots \prec x_0$), *below* the target qubit (greater indices), and satisfy at either $|0\rangle$ (a *negative* control) or $|1\rangle$ (*positive*). Both location and "sign" matter in translating control lines into QMDD vertices.

Gate construction can be defined efficiently, as an algorithm of $\mathcal{O}(k)$ cost for gates acting on k qubits, that works in a bottom-up fashion and performing only a single pass [41, 50]. The algorithm is shown in Alg. 7. Control handling is split between the

ones below the target and the ones above the target. If neither case exists, then these parts are ignored and only the target is created. Therefore, the generalization correctly encapsulates single-qubit gates as well.

Algorithm 7: Generalized gate construction as QMDD

Data: The gate's matrix m, the index of the target qubit t, control information $\mathcal{C}$, and the global terminal $\mathcal{T}$
Result: e, the entry of the matrix QMDD implementing the specified gate
$r \leftarrow \{(\mathcal{T}, m_{00})(\mathcal{T}, m_{01})(\mathcal{T}, m_{10})(\mathcal{T}, m_{11})\}$
$x \leftarrow$ an array of 4 zero-edges
for *c in* $\mathcal{C}$ *below* t **do**
  for $i, j = 0, 1$ **do**
    $k \leftarrow 2i + j$
    $\alpha, \beta \leftarrow 3, 0$ if c a positive control, 0, 3 otherwise
    $x_\alpha \leftarrow r_k$
    $x_\beta \leftarrow (\mathcal{T}, 1)$ if $i = j$, otherwise a zero-edge
    $r_k \leftarrow$ An edge pointing to a (normalized) vertex created via Alg. 1, with variable the index c acts on and x as edges
  end
end
$e \leftarrow$ An edge pointing to a (normalized) vertex created via Alg. 1, with variable t and edges r
for *c in* $\mathcal{C}$ *above* t **do**
  $\alpha, \beta \leftarrow 3, 0$ if c a positive control, 0, 3 otherwise
  $x_\alpha \leftarrow e$
  $x_\beta \leftarrow (\mathcal{T}, 1)$
  $e \leftarrow$ An edge pointing to a (normalized) vertex created via Alg. 1, with variable the index c acts on and x as edges
end
return e

In Alg. 7, one can notice that, for empty $\mathcal{C}$, a single vertex is defined, pointing to the terminal and implementing the gate's matrix in its weights. Thus, the algorithm correctly avoids redundant gates by trivializing implicitly [41]. The same effect appears in the two control loops; the controls point directly to the terminal with weight 1 for edges representing unsatisfactory values.[2] Otherwise, they point to the iteratively growing diagram. The application of Alg. 1 throughout further guarantees that the resulting vertices are unique and recycled, ensuring the compactness of the gate's representation. Figure 7b shows an example application and the resulting QMDD as produced by Alg. 7.

The above algorithm perfectly encapsulates the gate-set considered here, and it does so for every gate-set based on single-qubit operations. However, it should be mentioned that there are well-defined quantum gates that operate on multiple targets (excluding controls). Prominent examples include the SWAP gate, and its controlled version, the Fredkin gate, shown in Fig. 8. Although both are implementable by elementary gates

[2] Because control qubits can never have their value change under an operation, the edges e_{01}, e_{10} on a QMDD vertex implementing a control are forbidden and thus always 0. The other two represent the control either activating or not. For example, a positive control activates on $e_{11}(|1\rangle \rightarrow |1\rangle)$, while it stays idle for e_{00} $(|0\rangle \rightarrow |0\rangle)$.

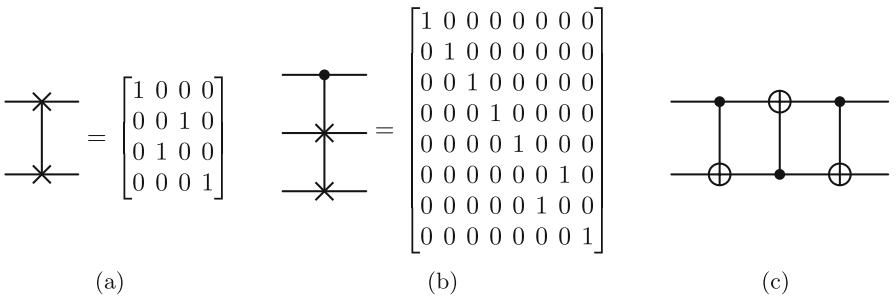


Fig. 8 Two-target quantum gates. **a** The SWAP gate, implementing $|b_1 b_0\rangle \rightarrow |b_0 b_1\rangle$. **b** The controlled version of SWAP, labeled Fredkin. **c** A decomposition of SWAP using CNOTs. A decomposition of Fredkin using CNOTs and Toffolis follows from this as well

found in the considered gate-set [32], there also exists a generalization of Alg. 7 that considers two targets [50].

Furthermore, for a cascade of single-qubit uncontrolled gates acting on disjoint sets of qubits (an *uncontrolled step*), an alternative procedure can merge them all in a single QMDD in a single pass [37]. This procedure also happens bottom-up and it achieves the same efficiency of $\mathcal{O}(k)$ cost for k gates. The algorithm is provided in Alg. 8. The simple routine starts from the terminal and builds a vertex for each gate, following its matrix for the weights similarly to before. An example is visualized in Fig. 7a.

Algorithm 8: Uncontrolled step construction as QMDD

Data: The desired gates' matrices $m_0, m_1, \cdots, m_n$, the target qubit indices $t_0, t_1, \cdots t_n$ and the global terminal $\mathcal{T}$
Result: e, the entry of the matrix QMDD implementing the step
$e \leftarrow (\mathcal{T}, 1)$
for m_i, t_i *iterated in bottom-up order* **do**
  $r \leftarrow$ an empty array of size 4
  for $j = 0, 1, 2, 3$ **do**
    $w \leftarrow w(e) \cdot m_{i;j}$
    $e' \leftarrow (v(e), w)$ if $w \neq 0$, a zero-edge otherwise
    $r_j \leftarrow e'$
  end
  $e \leftarrow$ An edge pointing to a (normalized) vertex created via Alg. 1, using t_i as variable and r as edges
end
return e

4.3 Circuit construction

Armed with a way to construct individual gates, all that is left is to define a manner by which they get applied to the vector QMDD implementing the current state of the circuit. As previously proposed, starting from the vector QMDD implementing the

initial state of the circuit, each gate QMDD can be applied onto it through matrix–vector multiplication [12, 19, 41, 52]. This fundamental operation carries a recursive function that is highly exploitable by QMDDs [28, 41], based on the observation shown in Eq. (9).

$$\begin{bmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{bmatrix} \times \begin{bmatrix} V_0 \\ V_1 \end{bmatrix} = \begin{bmatrix} M_{00}V_0 + M_{01}V_1 \\ M_{10}V_0 + M_{11}V_1 \end{bmatrix}, \quad \begin{bmatrix} V_0 \\ V_1 \end{bmatrix} + \begin{bmatrix} V_0' \\ V_1' \end{bmatrix} = \begin{bmatrix} V_0 + V_0' \\ V_1 + V_0' \end{bmatrix} \tag{9}$$

Based on Eq. (9), the QMDD matrix–vector multiplication algorithm computes the product of one level by recursively computing the products of a deeper level, and this continues until scalars are reached [29, 36, 41]. Since vertices are cached, it is expected that the same products appear multiple times. Therefore, by caching product results as well, the operation's exponential cost can be significantly reduced [29]. The procedure is described in Alg. 9.

Algorithm 9: QMDD Matrix-Vector Multiplication

Data: e_m, e_v the entries to the matrix and vector QMDDs, respectively, the terminal $\mathcal{T}$ and the current level l
Result: e, the entry to the vector QMDD representing the product $e_m \times e_v$
if *e_m or e_v is a zero edge* **then return** *a zero edge*
if $T(e_m)$ **then return** $(\mathcal{T}, w(e_m) \cdot w(e_v))$
if *the product is cached* **then return** *the cached result*
$r \leftarrow$ an array of 2 zero edges
for $i, j = 0, 1$ **do**
  $v \leftarrow E_j(e_v)$
  if $x(e_m) = l$ **then**
    $m \leftarrow E_{2i+j}(e_m)$
  else
    $m \leftarrow (v(e_m), 1)$, if $i = j$, otherwise a zero edge
  end
  $p \leftarrow$ multiply m, v using this procedure with level $l + 1$
  $r_i \leftarrow$ add r_i, p using Alg. 10 with level $l + 1$
end
$e \leftarrow$ an edge pointing to a (normalized) vertex created via Alg. 1 with variable l and r as edges
$w \leftarrow w(e)$ (after normalization)
$e \leftarrow (v(e), w \cdot w(e_m) \cdot w(e_v))$
Cache the product result e
return e

In Alg. 9, the vertex level l is required as a dependency injection in order to inform the algorithm of skipped variables [41]. The level relates to the variables of the two vertices to be multiplied together, and specifically, it is initialized as the one closest to the entry [41, 50]. So, in the variable order considered here, it is the minimum of $x(e_m), x(e_v)$. If during the recursion step, the variable of the matrix vertex is found to disagree with the current level, then that means a variable has been skipped. The procedure thus treats the matrix of the current level as an identity implicitly and defers operations on the true matrix to the next level [41]. In Alg. 9, the next level is $l + 1$, because variables increase the closer they are to the terminal. For the reverse order, it

would be $l-1$. Note that level disagreement can only occur for the matrix vertex; as specified in Sec. 3.3, trivialization, and by extension variable skipping, does not occur for vector QMDDs.

Vector addition emerges as a required subroutine for matrix–vector multiplication, a fact that can be noticed by both Eq. (9) and Alg. 9. The same recursive relationship can also be found in addition, and thus, QMDD vertex addition follows the same paradigm [29]. The routine is elucidated in Alg. 10 [29, 41, 50]. A slight generalization can make the addition algorithm work for tensors of any, but equal, rank. The level l is initialized in the same manner as in multiplication.

Algorithm 10: QMDD Addition

Data: e_0, e_1 the entries of the two QMDD representing tensors of the same rank, the terminal $\mathcal{T}$ and the current level l.
Result: e, the entry to the QMDD representing the sum $e_0 + e_1$
if e_0 *is a zero edge* **then return** e_1
if e_1 *is a zero edge* **then return** e_0
if $v(e_0) = v(e_1)$ **then return** $(v(e_0), w(e_0) + w(e_1))$
if *the sum is cached* **then return** *the cached result*
$\mathcal{R} \leftarrow$ the rank of the tensors
$r \leftarrow$ an empty array of size $2\mathcal{R}$
for $i \leftarrow 0$ **to** $2\mathcal{R} - 1$ **do**
  for $j = 0, 1$ **do**
    if $T(e_j) \vee x(e_j) > l$ **then**
      $p_j \leftarrow e_j$ if $i = 0 \vee i = 3$, otherwise a zero edge
    else
      $t \leftarrow E_i(e_j)$
      $p_j \leftarrow (v(t), w(t) \cdot w(e_j))$
    end
  end
  $r_i \leftarrow$ add p_0, p_1 using this procedure with level $l + 1$
end
$e \leftarrow$ an edge pointing to a (normalized) vertex created via Alg. 1 using l as variable and r as edges
Cache the sum result e
return e

5 Output extraction

Perhaps the most important point in circuit simulation is extracting the correct output. In the ideal scenario, the outcome of the simulation would reveal the theoretical amplitudes of each basis state of the resulting state vector. QMDDs hold such potential, as each unique edge path leads to one of the basis states, with the product of the weights along the way computing the amplitude [29, 52, 57].

Initial works [28, 29] proposed an output extraction scheme by which the final diagram was to be traversed according to a predefined set of r^2-selection variables. Essentially, which edge was to be followed from each vertex was known ahead of time. The reader is reminded here that r is the radix of the system, thus for binary QC

$r = 2$. This idea requires complexity cost linear to the size of the state vector diagram; however, it would output only a single basis state and a single amplitude, which for the general case of QC it would discard valuable information.

More recent works [12, 19, 52, 57] have articulated further refined approaches for output simulation. The two prominent methods can be found in the literature as *strong* and *weak simulation*.

5.1 Strong simulation

In [12], two algorithmic approaches can be recognized for "executing" the circuit diagram for a given initial state. *Explicit multiplication* introduces the concept of a vector QMDD by proposing the construction of the initial state as a matrix QMDD diagram, where the e_0, e_2 edges are utilized as normal and the e_1, e_3 ones point to null. Then, the circuit QMDD is to be multiplied to that initial vector QMDD, using a process similar to Alg. 9, giving the final state vector QMDD to be traversed in full in a Depth-First-Search (DFS) fashion. The second option, *implicit multiplication*, utilizes only the matrix QMDD representing the circuit. Its traversal from each vertex is performed according to the initial state in string form; specifically, the e_0, e_2 (e_1, e_3) edges are followed for variable initial value of $|0\rangle$ ($|1\rangle$).

Considering that modern QMDD work assumes the existence of a vector diagram, as detailed previously in this work, the approaches in [12] can be essentially merged into one. The initialization of the circuit can always be decomposed into elementary gates [32], meaning that the initial state preparation can be incorporated at the start of the circuit definition, leaving the initial vector simply implementing the ground state $|0\rangle^{\otimes n}$ (for circuit width n). As such, the final state vector QMDD can be traversed in DFS fashion for the full state vector, following [12].

Algorithm 11: Strong Simulation of QMDDs

```
Data: The state vector QMDD entry e
Result: A table L mapping each basis state of the statevector to its amplitude
Initialize empty table L
stack ← {e, w(e), ''}
while stack is not empty do
    {edge, weight, state} ← Pop the top of stack
    if T(edge) then
        L[state] ← weight
    else
        for i = 1, 0 do                          /* Preorder inclusion into stack */
            if w(E_i(edge)) = 0 then continue
            w' ← weight · w(E_i(edge))
            s' ← Place 'i' to state in position x(edge)
            Push {E_i(edge), w', s'} to stack
        end
    end
end
return L
```

$$QFT_n = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix}$$

Fig. 9 The Quantum Fourier Transform matrix for n qubits ($N = 2^n$). Each element of the matrix is a power of $\omega_N = e^{\frac{2\pi i}{N}}$. It is clear that the matrix contains no zeros. Such a matrix would generate no zero edges, making all traversals have exponential length

Thus strong simulation can be defined algorithmically as in Alg. 11. Traversing the diagram fully results in (ideally[3]) no loss of information. Each path yields a basis state with its theoretical probability amplitude as edge weight product. However, as was implied in Sec. 2.2, the state vector grows exponentially fast on the number of considered qubits. This is a fact grounded in theory and consequently unavoidable in general, which makes the vector traversal also exponential in cost. A small remedy to the exponential overhead is to completely skip zero edges during traversal, which is akin to not returning basis states with amplitude 0. Such a "sacrifice" would be acceptable on the user side, as a zero amplitude basis state could be inferred by its absence in the output. Still, many useful quantum operations exhibit no zero elements, and thus no zero edges as QMDDs, rendering this trick inapplicable. An important example is the Quantum Fourier Transform (QFT), shown in Fig. 9. Another remedy is to lazily return basis state-amplitude pairs, leveraging natively supported asynchronous generator tools found in modern programming languages, such as Python and JavaScript. This is possible because, due to DFS traversal, each edge path is traversed fully before moving on to the next. Therefore, each basis state-amplitude pair is reached in a sequential manner, allowing for an iteration-based representation of the state vector that would not commit exponential resources in-memory by default.

5.2 Weak simulation

Real quantum computers do not return the full state vector. Measurement induces collapse on the state of the machine right after the quantum algorithm. This leads to a single basis state, out of the probable ones, being returned, collapsed upon at random according to the probability distribution formed by the internal state vector of the system [32]. In order to counteract this stochastic behavior, many quantum algorithms are designed to be repeated multiple times to generate a statistical approximation of the output, leading to the so-called *shot-based measurement*.

Weak simulation arose as an idea, inspired by shot-based measurement, to counteract the exponential cost of strong simulation [19, 52, 57]. Since, after construction, the amplitudes are scattered inside the vector QMDD representation as edge weights, should they be mapped to selection probabilities for each vertex, then the resulting probability distribution could be sampled in linear time on the number of vertices,

[3] Up to precision errors relating to complex number representation, see Sec. 3.4

leading to the measurement of a single basis state out of the possible ones of the final state vector.

Without loss of generality, consider the case where the topmost qubit q_0 is to be measured. Its probability of collapse toward $|i\rangle$ ($i = 0, 1$) is the sum of all the squared magnitudes of the amplitudes relating to the sub-area of the state vector where $q_0 = |i\rangle$ [57]. Concretely, as in Eq. (10),

$$P(q_0 = |i\rangle) = \sum_{j \in i\{0,1\}^{n-1}} |\alpha_j|^2 = \sum_{j \in i0\{0,1\}^{n-2}} |\alpha_j|^2 + \sum_{j \in i1\{0,1\}^{n-2}} |\alpha_j|^2 \quad i = 0, 1 \tag{10}$$

Equation (10) reveals a recursive relationship among the collapse probabilities of q_0 and the collapse probabilities of following variables. This is evident in the rightmost part of the equation, where the left and right sub-sums compute the collapse probability of the next qubit in line (q_1) toward $|0\rangle$ and $|1\rangle$, respectively (barring q_0 as the most significant digit, which is fixed to $|i\rangle$). Therefore, the final sum of Eq. (10) computes $P(q_1 = |0\rangle | q_0 = |i\rangle) + P(q_1 = |1\rangle | q_0 = |i\rangle)$, which can be interpreted as invoking Eq. (10) for q_1 for the sub-area of the state vector where $q_0 = |i\rangle$.

The above relationship can be exploited algorithmically to stochastically perform single-shot measurement on the vector QMDD [19, 52, 57]. For this to be possible, vector QMDD vertices must be augmented with an additional attribute, the property **Pselect** (p_{select}), defined as $p_{select} = 1$ for terminal nodes and as in Eq. (11) for every other vertex.

$$p_{select} = p_{select;0} \cdot |w(E_0(e))|^2 + p_{select;1} \cdot |w(E_1(e))|^2 \tag{11}$$

In Eq. (11), e is the entry edge of the vertex for the variable in question, and $p_{select;0}$, $p_{select;1}$ the selection probabilities of $v(E_0(e))$, $v(E_1(e))$, respectively. By defining the selection probability of the terminal as 1 (since it is unique), the calculation of p_{select} can be performed inside Alg. 1, after normalization.

Armed with selection probabilities for every vertex, measurement can now be performed stochastically on q_0 by selecting between its children vertices randomly, according to the weighted probabilities shown in Eq. (12), where ρ is the entry of the entire diagram [57].

$$P(q_0 = |i\rangle) = |w(\rho)|^2 \cdot |w(E_i(\rho))|^2 \cdot p_{select;i} \quad i = 0, 1 \tag{12}$$

An alternative to Eq. (12) that avoids incorporating the weight of the entry ρ is shown in Eq. (13), leveraging the fact that it must be $P(q_0 = |0\rangle) + P(q_0 = |1\rangle) = 1$.

$$P(q_0 = |i\rangle) = \frac{|w(E_i(\rho))|^2 \cdot p_{select;i}}{|w(E_0(\rho))|^2 \cdot p_{select;0} + |w(E_1(\rho))|^2 \cdot p_{select;1}} \quad i = 0, 1 \tag{13}$$

After measurement of q_0, a real quantum computer would partially collapse its internal state vector. Mathematically, this means that, if, for example, q_0 was measured on $|0\rangle$, now basis states with $q_0 = |1\rangle$ have exactly zero probability of occurring. Also, the amplitudes of all other qubits are scaled by the inverse root of the probability of

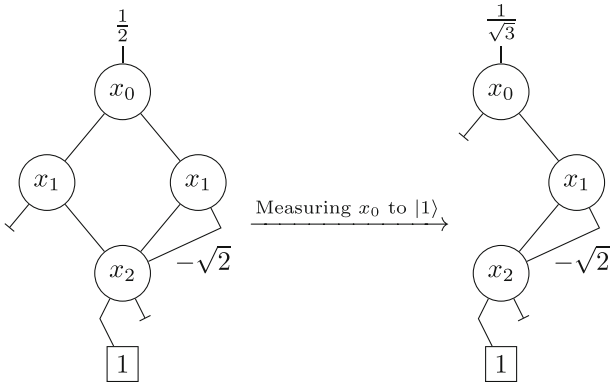


Fig. 10 An example of measurement collapse implemented on a vector QMDD. This specific example was inspired by [57] and recreated here. Measuring x_0 to $|0\rangle$ has probability $P_0 = (\frac{1}{2})^2 \cdot 1^2 \cdot 1 = \frac{1}{4}$, while measuring it to $|1\rangle$ has $P_1 = (\frac{1}{2})^2 \cdot 1^2 \cdot 3 = \frac{3}{4}$. After measuring to $|1\rangle$, the edge e_0 of vertex x_0 now leads to the impossible scenario $q_0 = |0\rangle$, fixed to 0 amplitude and, by extension, probability. Every other weight should be scaled by $\sqrt{\frac{1}{P_1}}$, to account for the received outcome. This can easily be done by only scaling the entry weight, as it represents the global amplitude factor by definition

the received scenario. This can be modeled on the vector QMDD, efficiently and in real time, by replacing the edge e_{1-i} of the measured vertex with a zero edge, and dividing the weight of the entry ρ by $\sqrt{P(q_0 = |i\rangle)}$ ($|i\rangle$ the output received) [19, 52, 57]. An example is pictured in Fig. 10.

This entire scheme can be extended for measuring multiple qubits [19, 52, 57]. By reordering the variables of the diagram or applying SWAP operations on it, all vertex levels relating to qubits-to-be-measured can be moved to the top side of the diagram. Then, each measured vertex scales the entry edge appropriately and leaves only a single nonzero edge to follow for the next measurement, achieving measurement complexity cost linear to the number of desired qubits [19, 57]. Measuring all the qubits is even simpler, as no reordering is required. A schematic of the described procedure can thus be identified in Alg. 12.

The function detailed in Alg. 12 performs single-shot measurement. As previously mentioned, usually multi-shot measurement is desired. For completely unitary circuits, the vector QMDD could be recycled for multiple invocations of Alg. 12 by not performing collapse after each selection [19]. However, for circuits with, for example, intermediate measurements, the dynamic alteration of the diagram is required for the expected execution of the described algorithm, which implies the reconstruction of the entire vector QMDD for each simulation shot [50]. Still, performing the collapse describes the full reality and faithfully emulates real quantum processors, as repeated measurements on the same qubit (without circuit reconstruction) should yield the same outcome every time [32].

As a final note, normalizing vector vertices by NR4, as previously defined in Sec. 3.2, can further improve performance [19, 52]. Since now each edge yields the probability to be followed via the square of its weight, the weights themselves can be

Algorithm 12: Weak Simulation of QMDDs (single-shot)

Data: The amount of levels m to measure, and the entry e of the state vector QMDD, already reordered so that the m qubits to measure are on the top side
Result: A randomly-chosen basis state s
$d \leftarrow e$
$s \leftarrow$ "$00\cdots0$" `/* The size of the circuit width */`
while $m > 0 \vee \neg T(d)$ **do**
  $o_0 \leftarrow |w(E_0(d))|^2 \cdot p_{select;0}$
  $o_1 \leftarrow |w(E_1(d))|^2 \cdot p_{select;1}$
  $P_0 \leftarrow \frac{o_0}{o_0+o_1}$
  $b \leftarrow$ select 0 with probability P_0, else 1
  if $b = 1$ **then** Place $'1'$ on s in position $x(d)$
  (Optional) divide $w(e)$ by $\sqrt{P_0}$ if $b = 0$ and $\sqrt{1-P_0}$ if $b = 1$, and set $E_{1-b}(d)$ to a zero-edge
  $d \leftarrow E_b(d)$
  $m \leftarrow m - 1$
end
return s

utilized as selection probabilities for each vertex. This avoids the extra p_{select} attributes at the cost of a slightly more complex normalization scheme.

6 QOLE: an open-source package

An implementation based on the concepts detailed in this paper is offered as a free and open-source package for JavaScript under the Mozilla Public License version 2.0 by the authors and can be found in https://github.com/asimakiskydros/QOLE. The package, dubbed the *Quantum Operations Lazy Evaluator* (shorthand QOLE, read as *Cole*), is a Software Development Kit aimed at facilitating the simulations of reversible and quantum circuits directly on the browser.

QOLE mimics the simple and declarative syntax of Qiskit [21] to ensure easy, quick usage, but also interchangeability between them. The state vector output is accessed as a Generator iterable, in line with previously detailed tactics. Backend details are hidden behind the interface, but should the experienced user be interested in experimenting with them, they are readily accessible through the package itself, thus also facilitating potential QMDD benchmarking alongside software development.

The package serves as a simplified starting point, designed to aid the understanding of researchers and practitioners interested in decision diagram-based simulation of quantum circuits. It also provides an accessible option for browser-based quantum computation tools. Indeed, the authors of this work plan to integrate the presented package into the open-source designer QuaCiDe [24], an online educational platform for quantum computing. As such, the scope of QOLE remains modest. For readers interested in high-performance alternatives, this work refers to some state-of-the-art simulators, such as DDSIM of the Munich Quantum Toolkit (MQT) [50], tensor diagram-based approaches such as TDD [20] and FTDD [53], as well as FlatDD, which employs hybrid, adaptive simulation techniques with heavy parallelization [22].

7 Extensions to the QMDD package

The application of sophisticated DD structures for the classical simulation of QC holds incredible promise. The QMDD package as presented in this work achieves a good scaling for important quantum algorithms in the ideal, noise-free scenario, facilitating their study. Extensions to the package have been proposed that include realistic noise effects into the structure, or lead to further compactness of the representation.

Real quantum computers are inherently noisy [32]. Noise effects on quantum states can occur via gate application errors, commonly called *depolarization error*. This occurs due to the ever-so imperfect operation of quantum machines and leads the state to an unexpected outcome. Another source are coherence errors, rooted in quantum theory. Those are the *amplitude damping error*, by which qubits tend to decay to the ground state ($|0\rangle$), and *phase flip errors*, by which interaction with the environment flips the phase of the qubit state. All such errors occur stochastically. Incorporating noise effects into classical simulation can be highly beneficial for practical research. Ideas for introducing noise into DD-based simulation have been developed. Using the idealized DD structure, the work in [17] approximates noise effects by randomly applying additional "unwanted" operations on the vector DD that simulate each noise case (e.g., $\{I, X, Y, Z\}$ for depolarization, Z for phase flip) with predefined probabilities. Multiple such "muddled" versions of the original vector diagram are then executed in parallel, and the average of their outcomes sufficiently approximates the noisy reality. An exact and deterministic approach has also been studied [15, 16], by which a state vector diagram is mapped to a diagram implementing its density operator (see Eq. (14), left), and noise effects on it are induced via matrix diagrams implementing Kraus operators. For example, the amplitude damping error with probability p is represented by the matrices shown in Fig. 14, right.

$$\rho = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \begin{bmatrix} \alpha^* & \beta^* \end{bmatrix} = \begin{bmatrix} |\alpha|^2 & \alpha\beta^* \\ \beta\alpha^* & |\beta|^2 \end{bmatrix} \qquad (E_0, E_1) = \left(\begin{bmatrix} 1 & 0 \\ 0 & \sqrt{1-p} \end{bmatrix}, \begin{bmatrix} 0 & \sqrt{p} \\ 0 & 0 \end{bmatrix} \right) \tag{14}$$

In Sec. 4, it was quickly mentioned that the QMDD structure can still lead to exponential representations. Maintaining compactness is perhaps the most crucial aspect of DD-based simulation. The work in [18] considers a dynamic behavior by which total accuracy is sacrificed for returns in increased efficiency. The approximation scheme centers around the fidelity of the representation (its similarity to its approximated version), employing two complementary heuristics; the first aims to maintain the size of the diagram under a user-defined threshold, to the potential detriment of the fidelity, while the second keeps the fidelity of the state above a user-defined threshold. Given these metrics, vertices contributing minimally to the overall fidelity are pruned, leading to a tunable accuracy-efficiency trade-off.

Further compactness requires reconsideration of major parts of the QMDD prism. It is a well-known fact that stabilizer circuits, those that are composed only by gates found in the Clifford set and measurements of Pauli group operators, can be perfectly simulated classically in polynomial time on the number of qubits [13]. However, as was previously mentioned in Sec. 4, a type of stabilizer state called a cluster state is known to require exponential QMDD representation [48]. For this reason, the works

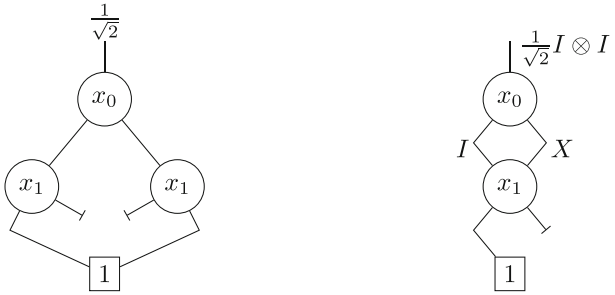


Fig. 11 A representation comparison between the QMDD structure (**a**) and the LIMDD structure utilizing the Pauli set $\{X, Y, Z, I\}$ (**b**) for the first Bell state $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. The case is of small scale, but even here the enhanced succinctness of LIMDDs over QMDDs can be noticed. This specific example was inspired by [49] and recreated here

[48, 49] develop a generalization of QMDDs called Local Invertible Map Decision Diagrams (LIMDDs). LIMDDs lead to identical vertices for states that are equivalent up to a local invertible map (a scaled tensor product train of 2×2 matrices from a selected group), as opposed to considering only scalar isomorphisms. A small comparison example for the first Bell state is shown in Fig. 11. Analysis reveals that the LIMDD extension succeeds in generating polynomially sized representations for circuits requiring exponential QMDD diagrams, and the overall LIMDD simulation achieves a best-case exponential speedup and a worst-case polynomial slow-down when compared to equivalent QMDDs.

Another important DD-based simulation approach is that of Tensor Decision Diagrams (TDD), studied in [20, 53]. The concept of a tensor generalizes those of scalars, vectors and matrices according to the number of required indices (rank) for their description (0-rank, 1-rank and 2-rank tensors respectively). Viewing all elements as tensors, a quantum circuit essentially forms a tensor network, an undirected graph connecting tensors together according to the indices they share. The TDD formalism exploits the Boole-Shannon expansion of tensors to form them as compact and canonical edge-weighted DAGs, similar to QMDDs, where each vertex level represents an index. Mapping all participant tensors of a circuit into TDDs, the contraction of adjacent tensors (according to the tensor network formed by the circuit schematic) in any order leads to the final circuit TDD representation. Efficient algorithms for TDD construction, addition and contraction have been developed, making use of recursion and memoization in a manner akin to QMDDs, while optimization strategies for rank simplification and optimal contraction orders have also been studied [53]. Experimental evaluation shows that TDDs have at least comparable compactness capabilities to QMDDs and in many cases lead to denser representations, while also achieving in general a significant speedup over them [20, 53].

8 Conclusions and future work

A mini-review on Quantum Multiple-Valued Decision Diagrams was realized in this work. The scope was strictly over the employment of QMDDs in binary reversible and quantum circuit simulation. Different procedural details and optimization techniques were compiled together from all over the literature, and a free and open-source package implementing the entire scheme in JavaScript was presented.

In the future, the presented implementation will be further refined, with the goal of incorporating it in the web designer QuaCiDe, as previously stated. Recognizing both the advantages and limitations of the exact integer-based representation of complex numbers showcased in Sec. 3.4, it is planned to make its application a toggle-able or even dynamic choice, opting for floating point-based representations when the integer bitwidths become unacceptably large. Relaxing the complex number representation also allows enriching the considered gate-set with more important gates, such as the rotational gates $R_x(\theta)$, $R_y(\theta)$, $R_z(\theta)$ that find enormous application in the current era of Noisy Intermediate-Scale Quantum computers, and especially in Quantum Machine Learning. These gates can be approximated by the considered gate-set, since it is universal, but they should be immediately obtainable from the simulator, both for accessibility and performance reasons (their approximation by the available gates adds unnecessary computational overhead than simply defining them as gates themselves). Further down the line, extensions incorporating noise effects and favoring further compactness could also be considered in the implementation.

Appendix A Complex operations in the integer regime

In the following proofs, $Re(\cdot)$, $Im(\cdot)$ refer to the real and imaginary part of a complex number, respectively, and, using the integer formulation adopted in this paper, they are defined as follows:

$$Re(z_j) = \frac{A_j + \frac{B_J}{\sqrt{2}}}{E_j}, \quad Im(z_j) = \frac{C_j + \frac{D_j}{\sqrt{2}}}{E_j} \tag{A1}$$

Complex addition

Proof From the mathematical definition:

$$\begin{aligned} z_1 + z_2 &= (Re(z_1) + Re(z_2)) + (Im(z_1) + Im(z_2))i \\ &\xrightarrow{(A1)} z_1 + z_2 = \frac{A_1 + \frac{B_1}{\sqrt{2}}}{E_1} + \frac{A_2 + \frac{B_2}{\sqrt{2}}}{E_2} + \frac{C_1 + \frac{D_1}{\sqrt{2}}}{E_1} i + \frac{C_2 + \frac{D_2}{\sqrt{2}}}{E_2} i \\ &= \frac{E_2 A_1 + E_2 \frac{B_1}{\sqrt{2}} + E_1 A_2 + E_1 \frac{B_2}{\sqrt{2}} + E_2 C_1 i + E_2 \frac{D_1}{\sqrt{2}} i + E_1 C_2 i + E_1 \frac{D_2}{\sqrt{2}} i}{E_1 E_2} \end{aligned}$$

$$= \frac{(A_1E_2 + A_2E_1) + \frac{(B_1E_2+B_2E_1)}{\sqrt{2}} + (C_1E_2 + C_2E_1)i + \frac{(D_1E_2+D_2E_1)i}{\sqrt{2}}}{E_1E_2}$$

$$\Rightarrow$$

$$A_s = A_1E_2 + A_2E_1, \quad B_s = B_1E_2 + B_2E_1, \quad C_s = C_1E_2 + C_2E_1$$
$$D_s = D_1E_2 + D_2E_1, \quad E_s = E_1E_2$$

□

Complex multiplication

Proof From the mathematical definition:

$$z_1z_2 = Re(z_1)Re(z_2) - Im(z_1)Im(z_2) + (Re(z_1)Im(z_2) + Re(z_2)Im(z_1))i$$

$$\xrightarrow{(A1)} z_1z_2 = \frac{A_1A_2 + \frac{A_1B_2+A_2B_1}{\sqrt{2}} + \frac{B_1B_2}{2}}{E_1E_2} - \frac{C_1C_2 + \frac{C_1D_2+C_2D_1}{\sqrt{2}} + \frac{D_1D_2}{2}}{E_1E_2}$$
$$+ \frac{A_1C_2 + \frac{A_1D_2+B_1C_2}{\sqrt{2}} + \frac{B_1D_2}{2}}{E_1E_2}i + \frac{A_2C_1 + \frac{A_2D_1+B_2C_1}{\sqrt{2}} + \frac{B_2D_1}{2}}{E_1E_2}i$$
$$= \frac{A_1A_2 + \frac{B_1B_2}{2} - C_1C_2 - \frac{D_1D_2}{2}}{E_1E_2} + \frac{A_1B_2 + A_2B_1 - C_1D_2 - C_2D_1}{\sqrt{2}E_1E_2}$$
$$+ \frac{A_1C_2 + A_2C_1 + \frac{B_1D_2}{2} + \frac{B_2D_1}{2}}{E_1E_2}i + \frac{A_1D_2 + B_1C_2 + A_2D_1 + B_2C_1}{\sqrt{2}E_1E_2}i$$
$$= \frac{(2A_1A_2 - 2C_1C_2 + B_1B_2 - D_1D_2) + \frac{2(A_1B_2+A_2B_1-C_1D_2-C_2D_1)}{\sqrt{2}}}{2E_1E_2}$$
$$+ \frac{(2A_1C_2 + 2A_2C_1 + B_1D_2 + B_2D_1)i + \frac{2(A_1D_2+B_1C_2+A_2D_1+B_2C_1)}{\sqrt{2}}i}{2E_1E_2}$$

$$\Rightarrow$$

$$A_p = 2(A_1A_2 - C_1C_2) + B_1B_2 - D_1D_2$$
$$B_p = 2(A_1B_2 + A_2B_1 - C_1D_2 - C_2D_1)$$
$$C_p = 2(A_1C_2 + A_2C_1) + B_1D_2 + B_2D_1$$
$$D_p = 2(A_1D_2 + B_1C_2 + A_2D_1 + B_2C_1)$$
$$E_P = 2E_1E_2$$

□

Complex division

Proof From the mathematical definition:

$$\frac{z_1}{z_2} = \frac{(Re(z_1) + Im(z_1)i)(Re(z_2) - Im(z_2)i)}{Re^2(z_2) + Im^2(z_2)} \tag{A2}$$

Calculating the denominator:

$$Re^2(z_2) + Im^2(z_2) \overset{(A1)}{=} \frac{A_2^2 + \frac{B_2^2}{2} + \frac{2A_2B_2}{\sqrt{2}} + C_2^2 + \frac{D_2^2}{2} + \frac{2C_2D_2}{\sqrt{2}}}{E_2^2}$$
$$= \frac{A_2^2 + C_2^2 + \frac{B_2^2+D_2^2}{2} + \sqrt{2}(A_2B_2 + C_2D_2)}{E_2^2} = \frac{DEN}{E_2^2}$$
$$\Rightarrow 2DEN = 2A_2^2 + 2C_2^2 + B_2^2 + D_2^2 + 2\sqrt{2}(A_2B_2 + C_2D_2)$$
$$= \alpha + 2\sqrt{2}\beta = DEN' \Rightarrow DEN'^* = \alpha - 2\sqrt{2}\beta \tag{A3}$$

Calculating the numerator:

$$(Re(z_1) + Im(z_1)i)(Re(z_2) - Im(z_2)i)$$
$$= Re(z_1)Re(z_2) + Im(z_1)Im(z_2) - Re(z1)Im(z_2)i + Re(z_2)Im(z_1)i$$
$$\overset{(A1)}{=} \frac{(A_1A_2 + \frac{A_1B_2+A_2B_1}{\sqrt{2}} + \frac{B_1B_2}{2}) - (A_1C_2 + \frac{A_1D_2+B_1C_2}{\sqrt{2}} + \frac{B_1D_2}{2})i}{E_1E_2}$$
$$+ \frac{(A_2C_1 + \frac{A_2D_1+B_2C_1}{\sqrt{2}} + \frac{B_2D_1}{2})i + (C_1C_2 + \frac{C_1D_2+C_2D_1}{\sqrt{2}} + \frac{D_1D_2}{2})}{E_1E_2}$$
$$= \frac{(A_1A_2 + C_1C_2 + \frac{B_1B_2+D_1D_2}{2}) + \frac{(A_1B_2+A_2B_1+C_1D_2+C_2D_1)}{\sqrt{2}}}{E_1E_2}$$
$$+ \frac{(A_2C_1 - A_1C_2 + \frac{B_2D_1-B_1D_2}{2})i + \frac{(A_2D_1+B_2C_1-A_1D_2-B_1C_2)i}{\sqrt{2}}}{E_1E_2}$$
$$= \frac{1}{E_1E_2}(\gamma' + \delta'\frac{1}{\sqrt{2}} + \epsilon'i + \zeta'\frac{i}{\sqrt{2}}) = \frac{NUM}{E_1E_2}$$
$$\Rightarrow 2NUM = \gamma + \delta\frac{1}{\sqrt{2}} + \epsilon i + \zeta\frac{i}{\sqrt{2}} \tag{A4}$$

Combining everything:

$$(A2) \overset{(A3)}{\underset{(A4)}{\Longrightarrow}} \frac{z_1}{z_2} = \frac{E_2}{E_1} \times \frac{2NUM}{2DEN} = \frac{E_2}{E_1} \times \frac{2NUM}{DEN'} = \frac{E_2}{E_1} \times \frac{2NUM \times DEN'^*}{DEN' \times DEN'^*}$$
$$= \frac{E_2}{E_1} \times \frac{(\alpha - 2\sqrt{2}\beta)(\gamma + \delta\frac{1}{\sqrt{2}} + \epsilon i + \zeta\frac{i}{\sqrt{2}})}{\alpha^2 - 8\beta^2}$$
$$= \frac{E_2}{E_1} \times \frac{\alpha\gamma + \frac{\alpha\delta}{\sqrt{2}} + \alpha\epsilon i + \frac{\alpha\zeta i}{\sqrt{2}} - 2\sqrt{2}\beta\gamma - 2\beta\delta - 2\sqrt{2}\beta\epsilon i - 2\beta\zeta i}{\alpha^2 - 8\beta^2}$$
$$= \frac{E_2}{E_1} \times \frac{(\alpha\gamma - 2\beta\delta) + \frac{1}{\sqrt{2}}(\alpha\delta - 4\beta\gamma) + i(\alpha\epsilon - 2\beta\zeta) + \frac{i}{\sqrt{2}}(\alpha\zeta - 4\beta\epsilon)}{\alpha^2 - 8\beta^2}$$
$$\Rightarrow$$
$$A_q = E_2(\alpha\gamma - 2\beta\delta), \quad \alpha = 2A_2^2 + 2C_2^2 + B_2^2 + D_2^2, \quad \beta = A_2B_2 + C_2D_2,$$
$$B_q = E_2(\alpha\delta - 4\beta\gamma), \quad \gamma = 2A_1A_2 + 2C_1C_2 + B_1B_2 + D_1D_2,$$
$$C_q = E_2(\alpha\epsilon - 2\beta\zeta), \quad \delta = 2(A_1B_2 + A_2B_1 + C_1D_2 + C_2D_1),$$

$$D_q = E_2(\alpha\zeta - 4\beta\epsilon), \quad \epsilon = 2A_2C_1 - 2A_1C_2 + B_2D_1 - B_1D_2,$$
$$E_q = E_1(\alpha^2 - 8\beta^2), \quad \zeta = 2(A_2D_1 + B_2C_1 - A_1D_2 - B_1C_2),$$

□

Author Contributions Konstantinos Prousalis had the idea for the article and guided its realization. Asimakis Kydros performed the literature search, carried out the study, and produced the relevant software. Nikos Konofaos critically revised the work.

Funding No funding was received for conducting this study.

Data Availability No new data were generated in the course of this study.

Code Availability The code developed as part of this study is freely available as open-source software.

Declarations

Conflict of interest The authors declare no conflict of interest.

Ethics Approval and Consent to Participate Ethical approval was not required for this study, as no experiments involving living beings were conducted. All authors have consented to participate in this work.

Consent for Publication All authors have reviewed and approved the manuscript for publication.

Materials Availability This study did not involve the use of physical materials.

Open Access This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by-nc-nd/4.0/.

References

1. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. **17**, 525–532 (1973)
2. Briegel, H.J., Browne, D.E., Dür, W., Raussendorf, R., Nest, M.: Measurement-based quantum computation. Nat. Phys. **5**, 19–26 (2009)
3. Burgholzer, L., Wille, R.: Advanced equivalence checking for quantum circuits. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **40**(9), 1810–1824 (2021)
4. Chen, T.-F., Chen, Y.-F., Jiang, J.-H.R., Jobranová, S., Lengál, O.: Accelerating quantum circuit simulation with symbolic execution and loop summarization. In Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design, ICCAD '24, New York, NY, USA, (2025). Association for Computing Machinery
5. De Vos, A.: Reversible Computing: Fundamentals, Quantum Computing, and Applications. Wiley, 11 (2010)
6. Dirac, P.A.M.: A new notation for quantum mechanics. Math. Proc. Camb. Philos. Soc. **35**(3), 416–418 (1939)

7. Feinstein, D.Y., Thornton, M.A.: On the Guidance of Reversible Logic Synthesis by Dynamic Variable Reordering. In: 2009 39th International Symposium on Multiple-Valued Logic, pages 132–138. IEEE, (2009)
8. Feinstein, D.Y., Thornton, M.A.: On the Skipped Variables of Quantum Multiple-Valued Decision Diagrams. In: 2011 41st IEEE International Symposium on Multiple-Valued Logic, pp 164–169. IEEE, (2011)
9. Feinstein, D.Y., Thornton, M.A.: Reversible logic synthesis based on decision diagram variable ordering. J. Mult. -Val. Log. Soft Comput. **19**(4), 325–339 (2012)
10. Feinstein, D.Y., Thornton, M.A.: Quantum multiple-valued decision diagrams containing skipped variables. J. Mult. -Val. Log. Soft Comput. **24**(1–4), 93–108 (2014)
11. Feinstein, D.Y., Thornton, M.A., Miller, D.M.: Minimization of quantum multiple-valued decision diagrams using data structure metrics. J Mult. -Val. Log. Soft Comput. **15**(4), 361–377 (2009)
12. Goodman, D., Thornton, M., Feinstein, D., Miller, M.: Quantum Logic Circuit Simulation Based on the QMDD data structure. Master's thesis, Southern Methodist University, (2007)
13. Gottesman, D.: The Heisenberg Representation of Quantum Computers, (1998)
14. Grover, L.K.: A Fast Quantum Mechanical Algorithm for Database Search, (1996)
15. Grurl, T., Fuß, J., Wille, R.: Considering decoherence errors in the simulation of quantum circuits using decision diagrams. In: Proceedings of the 39th International Conference on Computer-Aided Design, ICCAD '20, New York, NY, USA, 2020. Association for Computing Machinery
16. Grurl, T., Fuß, J., Wille, R.: Noise-aware quantum circuit simulation with decision diagrams. Trans. Comp. Aided Des. Integ. Cir. Sys. **42**(3), 860–873 (2023)
17. Grurl, T., Kueng, R., FuB, J., Wille, R.: Stochastic quantum circuit simulation using decision diagrams. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 194–199, (2021)
18. Hillmich, S., Kueng, R., Markov, I., Wille, R.: As accurate as needed, as efficient as possible: Approximations in dd-based quantum circuit simulation. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 188–193, (2021)
19. Hillmich, S., Markov, I.L., Wille, R.: Just like the real thing: fast weak simulation of quantum computation. In: 2020 57th ACM/IEEE Design Automation Conference (DAC), pages 1–6, (2020)
20. Hong, X., Zhou, X., Li, S., Feng, Y., Ying, M.: A tensor network based decision diagram for representation of quantum circuits. ACM Trans. Des. Autom. Electron. Syst. **27**, 1 (2021)
21. Javadi-Abhari, A., Treinish, M., Krsulich, K., Wood, C.J., Lishman, J., Gacon, J., Martiel, S., Nation, P.D., Bishop, L.S., Cross, A.W., Johnson, B.R., Gambetta, J.M.: Quantum computing with Qiskit, (2024)
22. Jiang, S., Fu, R., Burgholzer, L., Wille, R., Ho, T.-Y., Huang, T.-W.: Flatdd: A high-performance quantum circuit simulator using decision diagram and flat array. In: Proceedings of the 53rd International Conference on Parallel Processing, ICPP '24, pp. 388 399, New York, NY, USA, (2024). Association for Computing Machinery
23. Knuth, D.E.: The Art of Computer Programming, Vol. 2: Seminumerical Algorithms. 3rd ed. Addison Wesley, (1997)
24. Kydros, A., Prousalis, K., Konofaos, N.: Quacide: A general purpose quantum circuit design and simulation interface. In: Proceedings of Recent Advances in Quantum Computing and Technology, ReAQCT '24, page 51 55, New York, NY, USA, (2024). Association for Computing Machinery
25. Landauer, R.: Irreversibility and heat generation in the computing process. IBM J. Res. Dev. **5**(3), 183–191 (1961)
26. Li, Y., Miao, H.: Quantum Multiple-valued Decision Diagrams with Linear Transformations, (2022)
27. Mato, K., Hillmich, S., Wille, R.: Mixed-dimensional qudit state preparation using edge-weighted decision diagrams. In: Proceedings of the 61st ACM/IEEE Design Automation Conference, DAC '24, New York, NY, USA, (2024). Association for Computing Machinery
28. Miller, D.M., Thornton, M.A.: QMDD: A decision diagram structure for reversible and quantum circuits. In: 36th International Symposium on Multiple-Valued Logic (ISMVL'06), pp. 30–30. IEEE, (2006)
29. Miller, D.M., Thornton, M.A., Goodman, D.: A decision diagram package for reversible and quantum circuit simulation. In: 2006 IEEE International Conference on Evolutionary Computation, pp. 2428–2435. IEEE, (2006)
30. Miller, D.M., Feinstein, D.Y., Thornton, M.A.: QMDD minimization using sifting for variable reordering. J. Mult. -Val. Log. Soft Comput. **13**(4–6), 537–552 (2007)

31. Miller, D.M., Feinstein, D.Y., Thornton, M.A.: Variable reordering and sifting for QMDD. In: 37th International Symposium on Multiple-Valued Logic (ISMVL'07), pp. 10–10. IEEE (2007)
32. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press (2010)
33. Niemann, P., Wille, R., Drechsler, R.: On the "Q" in QMDDs: efficient representation of quantum functionality in the QMDD data-structure, pp. 125–140. Springer Berlin Heidelberg (2013)
34. Niemann, P., Wille, R., Drechsler, R.: Efficient synthesis of quantum circuits implementing clifford group operations. In: 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 483–488. IEEE (2014)
35. Niemann, P., Wille, R., Drechsler, R.: Equivalence Checking in Multi-level Quantum Systems, pp. 201–215. Springer International Publishing (2014)
36. Niemann, P., Wille, R., Miller, D.M., Thornton, M.A., Drechsler, R.: QMDDs: efficient quantum function representation and manipulation. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **35**(1), 86–99 (2016)
37. Niemann, P., Zulehner, A., Wille, R., Drechsler, R.: Efficient Construction of QMDDs for Irreversible, Reversible, and Quantum Functions, pp. 214–231. Springer International Publishing (2017)
38. Raussendorf, R., Briegel, H.J.: A one-way quantum computer. Phys. Rev. Lett. **86**, 5188–5191 (2001)
39. Saeed, S.M., Cui, X., Zulehner, A., Wille, R., Drechsler, R., Wu, K., Karri, R.: IC/IP piracy assessment of reversible logic. In: Proceedings of the International Conference on Computer-Aided Design, ICCAD '18, pp. 1–8. ACM (2018)
40. Saeed, S.M., Zulehner, A., Wille, R.R., Drechsler, and Ramesh K (2019) Reversible circuits: IC/IP piracy attacks and countermeasures. IEEE Trans. Very Large Scale Integr. VLSI Syst. **27**(11), 2523–2535
41. Sander, A., Florea, I.-A., Burgholzer, L., Wille, R.: Stripping Quantum Decision Diagrams of their Identity, (2024)
42. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997)
43. Smith, K.N., Thornton, M.A.: Quantum logic synthesis with formal verification. In: 2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 73–76. IEEE (2019)
44. Soeken, M., Wille, R., Hilken, C., Przigoda, N., Drechsler, R.: Synthesis of reversible circuits with minimal lines for large functions. In: 17th Asia and South Pacific Design Automation Conference, pp. 85–92. IEEE (2012)
45. Stankovic, R.S., Miller, D.M.: Using QMDD in numerical methods for solving linear differential equations via Walsh functions. In: 2015 IEEE International Symposium on Multiple-Valued Logic, pp. 182–188. IEEE (2015)
46. Stankovic, S., Astola, J., Miller, D.M., Stankovic, R.S.: Heterogeneous decision diagrams for applications in harmonic analysis on finite non-abelian groups. In: 2010 40th IEEE International Symposium on Multiple-Valued Logic, pp. 307–312. IEEE (2010)
47. Tsai, Y.-H., Jiang, J.-H.R., Jhang, C.-S.: Scaling Up Accurate Quantum Circuit Simulation to a New Level, Bit-Slicing the Hilbert Space (2020)
48. Vinkhuijzen, L., Coopmans, T., Elkouss, D., Dunjko, V., Laarman, A.: LIMDD: a decision diagram for simulation of quantum computing including stabilizer states. Quantum **7**, 1108 (2023)
49. Vinkhuijzen, L., Grurl, T., Hillmich, S., Brand, S., Wille, R., Laarman, A.: Efficient Implementation of LIMDDs for Quantum Circuit Simulation, pp. 3–21. Springer, Cham, 05 (2023)
50. Wille, R., Berent, L., Forster, T., Kunasaikaran, J., Mato, K., Peham, T., Quetschlich, N., Rovara, D., Sander, A., Schmid, L., Schoenberger, D., Stade, Y., Burgholzer, L.: The MQT handbook: a summary of design automation tools and software for quantum computing. In: IEEE International Conference on Quantum Software (QSW) (2024)
51. Wille, R., Grosse, D., Miller, D.M., Drechsler, R.: Equivalence checking of reversible circuits. J. Mult. -Val. Log. Soft Comput. **19**(4), 361–378 (2012)
52. Wille, R., Hillmich, S., Burgholzer, L.: Decision Diagrams for Quantum Computing, pp. 1–23. Springer International Publishing, Cham (2023)
53. Zhang, Q., Saligane, M., Kim, H.-S., Blaauw, D., Tzimpragos, G., Sylvester, D.: Quantum Circuit Simulation with Fast Tensor Decision Diagram. In 2024 25th International Symposium on Quality Electronic Design (ISQED), pp. 1–8. IEEE (2024)

54. Zulehner, A., Hillmich, S., Wille, R.: How to efficiently handle complex values? implementing decision diagrams for quantum computing. In: 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–7 (2019)
55. Zulehner, A., Niemann, P., Drechsler, R., Wille, R.: Accuracy and compactness in decision diagrams for quantum computation. In: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 280–283 (2019)
56. Zulehner, A., Rani, P.M.N., Datta, K., Sengupta, I., Wille, R.: Generalizing the concept of scalable reversible circuit synthesis for multiple-valued logic. In: 2018 IEEE 48th International Symposium on Multiple-Valued Logic (ISMVL), pp. 115–120. IEEE (2018)
57. Zulehner, A., Wille, R.: Advanced Simulation of Quantum Computations (2018)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.