Efficient Implementation of LIMDDs for Quantum Circuit Simulation

Lieuwe Vinkhuijzen[1], Thomas Grurl[2,3], Stefan Hillmich[3], Sebastiaan Brand[1], Robert Wille[4,5], and Alfons Laarman[1]

[1] Leiden University, The Netherlands
[2] Secure Information Systems, University of Applied Sciences Upper Austria, Austria
[3] Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
[4] Chair for Design Automation, Technical University of Munich, Germany
[5] Software Competence Center Hagenberg GmbH (SCCH), Austria

Abstract. Realizing the promised advantage of quantum computers over classical computers requires both physical devices and corresponding methods for the design, verification and analysis of quantum circuits. In this regard, decision diagrams have proven themselves to be an indispensable tool due to their capability to represent both quantum states and unitaries (circuits) compactly. Nonetheless, recent results show that decision diagrams can grow to exponential size even for the ubiquitous stabilizer states, which are generated by Clifford circuits. Since Clifford circuits can be efficiently simulated classically, this is surprising. Moreover, since Clifford circuits play a crucial role in many quantum computing applications, from networking, to error correction, this limitation forms a major obstacle for using decision diagrams for the design, verification and analysis of quantum circuits. The recently proposed *Local Invertible Map Decision Diagram* (LIMDD) solves this problem by combining the strengths of decision diagrams and the stabilizer formalism that enables efficient simulation of Clifford circuits. However, LIMDDs have only been introduced on paper thus far and have not been implemented yet—preventing an investigation of their practical capabilities through experiments. In this work, we present the first implementation of LIMDDs for quantum circuit simulation. A case study confirms the improved performance in both worlds for the Quantum Fourier Transform applied to a stabilizer state. The resulting package is available under a free license at `https://github.com/cda-tum/ddsim/tree/limdd`.

1 Introduction

Quantum computing is a new and drastically different computing paradigm promising to solve certain problems that are intractable for classical computers. Examples of such problems include unstructured search [1]–[3], integer factorization [4] and quantum chemistry [5]. This computational power is harnessed by using quantum mechanical effects such as *superposition*, where the system can be in a linear combination of multiple states, and *entanglement*, where operations on one part of the system can affect other parts as well. In the near term,

quantum computers classified as *Noisy Intermediate-Scale Quantum* (NISQ) devices are expected to both deliver empirical evidence of quantum advantage over classical computers as well as solve practical problems. However, the ability to build large quantum computers is not by itself sufficient if there are no means of harnessing their power: we also need tools for the design of quantum circuits, i.e., for simulation, compilation, and verification. The *classical simulation* of quantum computers in particular has been used in service of the verification of quantum algorithms [6]–[8], and is a way to quantify "the elusive boundary at which a quantum advantage may materialize" [9].

A major challenge in the classical design of quantum systems is that the memory requirements grow exponentially in the number of qubits. Contrary to the classical world, where representing a system state with m classical bits requires only a linear amount of memory, the state of an n-qubit quantum system is described by a vector of 2^n complex numbers. Current estimates indicate that at least hundreds of qubits are required to perform useful tasks on a quantum computer [10]. However, even current super-computing clusters can only handle systems with between 50 and 60 qubits represented as vectors [11]. Therefore, dedicated data structures and design methods which can tackle the exponential complexity of quantum computing need to be developed.

Given that merely representing a quantum state may require an exponential amount of memory with respect to the number of qubits, it comes as no surprise that conducting quantum circuit simulation, for example, is a hard problem. Even more dauntingly, verification of quantum circuits at its heart considers quantum operations and therefore has $2^n \times 2^n$ complexity when implemented naively. Fortunately, due to the characteristics of quantum computing, quantum circuit simulation can help to verify the equivalence of two circuits to a very high degree of confidence, despite the infinite number of possible input states. More precisely, providing an appropriate quantum state as input to the circuits under consideration and checking equivalence of the resulting states will provide a counterexample for non-equivalent circuits with a high probability [7]. Selecting basis states as input, i.e., states without superposition or entanglement, does not always suffice for this purpose, but random *stabilizer states*, introduced next, have been shown to do the trick [6]. This makes quantum circuit simulation a key component of design automation for quantum computing and hence the subject of the current paper.

Stabilizer states are ubiquitous in quantum computing. They are computed by so-called Clifford circuits, a subset of the universal quantum computing gate set [12], [13]. For example, stabilizer states include the Bell state and GHZ state. Further, Clifford circuits play an essential role in error correction [14], [15], entanglement distillation [16] and are used in one-way quantum computing [17]. Any n-qubit stabilizer state can be represented using memory in the order of $\mathcal{O}(n^2)$ and the non-universal fragment of Clifford circuits can be simulated efficiently by manipulating this representation [12], [13]. In fact, stabilizer states capture the essential symmetries of all universal quantum computing states, which is why they also play a key role in the reduction from verification to simulation

explained above. For these reasons, it can be argued that any practically efficient classical simulation of (universal) quantum computing should also support Clifford circuits and the stabilizer states they generate. The current work removes this limitation from existing (universal) simulation approaches based on decision diagrams.

Decision diagrams (DDs) are a tried-and-tested data structure in the world of classical design automation [18]–[22]. They have also shown promising results in quantum design automation [23]–[28]. Decision diagrams exploit redundancies in the state vector and operations matrix to enable a compact representation in many cases. Unfortunately, a state-of-the art quantum simulation method called Quantum Multi-valued Decision Diagram (QMDD) [29] does not efficiently represent stabilizer states [27], which poses a serious bottleneck to their adoption, as explained above, but also observed in practice [6]. The recently proposed *Local Invertible Map Decision Diagram* (LIMDD, [27]) addresses this shortcoming. LIMDDs efficiently represent stabilizer states, they simulate Clifford circuits in polynomial time and can efficiently apply many Clifford gates also to non-stabilizer states. However, LIMDDs lack an implementation to demonstrate that their asymptotic advantage also translates to practical use cases.

In this paper, we present an implementation of LIMDDs for universal simulation of quantum circuits (and thus design automation) based on the QMDD package [26], [28]. We adapt techniques that are tried and tested in the implementations of both classical and quantum decision diagram packages, and enrich them with special considerations to efficiently handle *Local Invertible Maps* (LIMs). For the first time, this leads to an implementation that realizes LIMDDs, and also demonstrates the potential of LIMDDs. In particular, we show their use for verification through circuit equivalence checking for a case study on the Quantum Fourier Transform (QFT, [30], [31]). The results confirm that the more complex LIMDD-based simulator surpasses a state-of-the-art decision-diagram-based simulator for larger instances. The resulting implementation is available at `https://github.com/cda-tum/ddsim/tree/limdd` under the MIT license.

The remainder of this paper is structured as follows. Section 2 briefly reviews the necessary background on quantum computing and classical quantum circuit simulation. In Section 3, we briefly review existing decision diagrams for quantum computing and motivate the need for an efficient LIMDD implementation. Section 4 details the techniques used to enable efficient construction and manipulation of LIMDDs. In Section 5, we provide an experimental evaluation showcasing the performance of the proposed implementation. Finally, Section 6 concludes the paper.

2 Background

To keep this work self-contained, this section provides the necessary background on quantum computing as well as classical quantum circuit simulation.

2.1 Quantum States and Operations

The basic unit of information in quantum computing is the *quantum bit* or *qubit* [30]. A single-qubit quantum state $|\psi\rangle$ can be described by its amplitude vector $|\psi\rangle = \alpha_0 \cdot |0\rangle + \alpha_1 \cdot |1\rangle$ with complex amplitudes $\alpha_0, \alpha_1 \in \mathbb{C}$. Here, $|0\rangle = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right], |1\rangle = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$ denote the two basis states and are analogous to the basis states 0 and 1 in classical computing, in the sense that a classical register containing one bit can be either in state 0 or 1. An amplitude vector must meet the normalization constraint $|\alpha_0|^2 + |\alpha_1|^2 = 1$. If both amplitudes α_0 and α_1 are non-zero, the state is in *superposition*. For a state $|\phi\rangle$, we write $\langle\phi| = |\phi\rangle^\dagger$ to denote its conjugate transpose. Two quantum states $|\phi\rangle, |\psi\rangle$ are considered equal if the absolute value of their in-product equals one, i.e., $|\langle\phi| \cdot |\psi\rangle| = 1$, also written as $|\langle\phi|\psi\rangle| = 1$. We say that $|\phi\rangle, |\psi\rangle$ are approximately the same state if $|\langle\phi|\psi\rangle| \approx 1$. When measuring a state, the probability that a given basis state is the outcome is the squared magnitude of the amplitude of that basis state, i.e, for the state $\alpha_0 \cdot |0\rangle + \alpha_1 \cdot |1\rangle$, the probability of measuring the zero state is $|\alpha_0|^2$. Therefore, in a physical quantum computer, the individual amplitudes are fundamentally non-observable and information about the quantum state can only be extracted through destructive measurement, i.e., after measurement, superposition is destroyed.

A quantum register may consist of multiple qubits. A register $|\phi\rangle$ consisting of n qubits has 2^n basis states $|i\rangle$ with $i \in \{0,1\}^n$, each with a corresponding amplitude α_i, i.e., $|\phi\rangle = \sum_i \alpha_i |i\rangle$. Here a basis state $|i\rangle$ with i a bit string $b_1 b_2 \dots b_n$ is formed from the single qubit basis vectors above using tensor products $|b_1\rangle \otimes |b_2\rangle \otimes \cdots \otimes |b_n\rangle$, written shortly as $|b_1\rangle|b_2\rangle \dots |b_n\rangle = |b_1 b_2 \dots b_n\rangle = |i\rangle$. Here the *tensor product* of a $k \times \ell$ matrix $A = (a_{ij})_{ij}$ and an $n \times m$ matrices $B = (b_{ij})_{ij}$ is the $kn \times \ell m$ matrix $C = (a_{ij} b_{xj})_{ijxy}$. Alternatively, $|\phi\rangle$ can be understood as the pseudo-Boolean function $f\colon \{0,1\}^n \to \mathbb{C}$, defined as $f(i) = \langle i|\phi\rangle = \alpha_i$. The normalization constraint is generalized to $\sum_{0 \le i < 2^n} |\alpha_i|^2 = 1$. A quantum state $|\phi\rangle$ is *entangled* if it cannot be written as a tensor product of single qubit states, i.e, $|\phi\rangle = |\phi_1\rangle \otimes \cdots \otimes |\phi_n\rangle$.

Example 1. Consider the quantum state $1/\sqrt{2} \cdot (|00\rangle + |11\rangle)$, commonly known as the *Bell state* [30]. As a vector, it would be written as $1/\sqrt{2} \cdot \left[\begin{smallmatrix}1 & 0 & 0 & 1\end{smallmatrix}\right]^{\mathrm{T}}$. If this state is measured, we have equal probabilities of obtaining as outcome one of the basis states $|00\rangle$ and $|11\rangle$, and zero probability of seeing the states $|01\rangle$ and $|10\rangle$.

In addition to superposition, this quantum state shows *entanglement*. Measuring a value for one qubit of the Bell state would immediately fix the value of the other qubit corresponding to the measurement outcome, e.g., after measuring $q_1 = |0\rangle$ (or $q_1 = |1\rangle$) we immediately know that $q_0 = |0\rangle$ (or $q_0 = |1\rangle$).

Quantum states are manipulated through quantum gates. A quantum gate is any linear operator mapping quantum states to quantum states, i.e., a unitary matrix $U \in \mathbb{C}^{2^n \times 2^n}$, where n is the number of qubits. A quantum algorithm consists of a series of gates applied sequentially, e.g., $U = U_m \cdots U_1$ is an algorithm consisting of m gates, which first applies U_1, then U_2, up to U_m. Thus, U denotes the unitary matrix corresponding to applying all the gates. If a quantum state $|\phi\rangle$

serves as input to U, then the output is the quantum state $U \cdot |\phi\rangle$. We say that a quantum algorithm $U_1, \ldots, U_m$ is equivalent to another quantum algorithm $V_1, \ldots, V_\ell$ iff they effect the same unitary matrix, i.e., if $U_m \cdots U_1 = V_\ell \cdots V_1$.

Example 2. Three examples of common quantum gates are the single-qubit phase-shift operation S, the single-qubit Hadamard operation H, and the two-qubit controlled-NOT operation $CNOT$ (here shown with control on the first qubit, target on the second qubit).

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \qquad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The state from Example 1 can be created by starting the $|00\rangle$ basis state and then applying a Hadamard operation on the first qubit, followed by a controlled-NOT. Using the tensor product for parallel composition of gates as usual [30], this can be written as $CNOT \times (H \otimes \mathbb{I}) \times |00\rangle = |11\rangle$. Figure 1 shows another example of a 3-qubit quantum circuit built from these three gates generalized to multiple qubits. The circuit is to be read from left to right, so that the Hadamard gate is the first gate, applied to qubit q_2. The $\bullet$ denotes the control qubit of a $CNOT$ gate; the $\oplus$ denotes its target qubit.

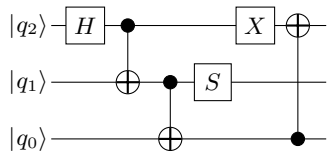


Fig. 1: A quantum circuit on three qubits, q_2–q_0.

An important, though non-universal, subset of quantum circuits are Clifford circuits [14], which consist only of the Clifford gates S, H and $CNOT$. Clifford circuits are ubiquitous in quantum computing because they represent the symmetries that occur in all quantum states albeit they cannot generate arbitrary transformations (hence they are non-universal). As a consequence, they play an essential role in error correction [14], [15], entanglement distillation [16] and are used in one-way quantum computing [17]. Clifford circuits are intimately related to the Pauli gate set:

$$\mathbb{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Example 3. The circuit shown in Figure 1 is a Clifford circuit, since it only consists of Clifford operations and gates that can be built from Clifford operations. The Pauli gates can be built from Clifford gates, namely as $Z = S^2$ and $X = H \times Z \times H$, and $Y = iXZ$; thus, Pauli gates are Clifford gates.

2.2 Classical Quantum Circuit Simulation

The classical simulation of a quantum circuit is the process of simulating a quantum circuit on a classical binary computer. It is an important task in the context of quantifying the capability of physical quantum computers and in the development of quantum algorithms.

Circuit simulation can be conducted in a straightforward fashion by repeated matrix-vector multiplication. The simulation starts with an initial state and applies the quantum gates one after the other. Each quantum operation is represented by a unitary matrix U_t of dimension $2^n \times 2^n$ and each quantum state by a unit vector $|\phi_t\rangle$ of dimension 2^n. The evolution of a state at time step t is then given by $|\phi_{t+1}\rangle = U_{t+1}|\phi_t\rangle$ (with $|\phi_0\rangle$ commonly being the vector representing the all-zero state, $|\phi_0\rangle = |0\dots0\rangle$).

Clifford circuits can be efficiently simulated on a classical computer [13], which is surprising given their importance, but not a contradiction given their non-universality. Starting from the all-zero state $|0^n\rangle$, Clifford circuits only yield so-called *stabilizer states*. An n-qubit stabilizer state $|\phi\rangle$ can be uniquely specified by the set of Pauli operators $G = \pm P_1 \otimes \cdots \otimes P_n$ with $P_i \in \{\mathbb{I}, X, Y, Z\}$ that stabilize it, i.e., which satisfy $G|\phi\rangle = |\phi\rangle$. This set forms an abelian group, and can always be described succinctly by a set of n generators $G_1, \dots, G_n \in \pm\{\mathbb{I}, X, Y, Z\}^n$. We can thus think of this generator set as an $n \times n$ matrix of local Pauli operators, where each row (a generator) also has an additional plus or minus sign as shown in Example 4. This characterization is the key to efficiently classical simulation of stabilizer states [12] since the Clifford gates can be applied directly to the generator set describing the state.

Example 4. The three-qubit Clifford circuit in Figure 1 can be simulated for time steps $t = 0, 1, 2$ using explicit vector representation as follows.

$$
\begin{aligned}
|\phi_0\rangle &= |000\rangle && \text{apply } H \text{ on } q_2 \rightarrow \\
|\phi_1\rangle &= \frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|001\rangle && \text{apply CNOT on control } q_1 \text{ and target } q_2 \rightarrow \\
|\phi_2\rangle &= \frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|011\rangle && (\text{which is } |0\rangle \otimes \text{Bell state})
\end{aligned}
$$

Alternatively, we may represent each $|\phi_t\rangle$ as a generator set $G(\phi_t) = \{G_1, G_2, G_3\}$.

$$
\begin{aligned}
G(\phi_0) &= \{\, Z\,\mathbb{I}\,\mathbb{I},\ \mathbb{I}\,Z\,\mathbb{I},\ \mathbb{I}\,\mathbb{I}\,Z \,\} && \text{apply } H \text{ on } q_2 \rightarrow \\
G(\phi_1) &= \{\, Z\,\mathbb{I}\,\mathbb{I},\ \mathbb{I}\,Z\,\mathbb{I},\ \mathbb{I}\,\mathbb{I}\,X \,\} && \text{apply CNOT on control } q_1 \text{ and target } q_2 \rightarrow \\
G(\phi_2) &= \{\, Z\,\mathbb{I}\,\mathbb{I},\ \mathbb{I}\,ZZ,\ \mathbb{I}\,XX \,\}
\end{aligned}
$$

We call the generator set representing a stabilizer state a stabilizer tableau. The *stabilizer formalism* stipulates how the tableau should be modified for different Clifford gates [12], [13] as exemplified in Example 4. It forms a non-universal, but classically tractable region of quantum computing, whereas decision diagrams considered in this work target universal quantum computing. Nonetheless, Clifford circuits and stabilizer states are important in many domains, as discussed in the introduction.

2.3 Verification of Quantum Circuits

A popular similarity metric for comparing two quantum circuits U, V is the *average fidelity*, $\mathcal{F}_{\text{avg}}(U,V) = \frac{1}{1+2^n}(1 + |\text{tr}(UV^\dagger)|^2)$, where $\text{tr}(M)$ denotes the trace of M. This metric has value 1 iff $U = V$, and has $\mathcal{F}_{\text{avg}}(U,V) < 1$ otherwise. Burgholzer et. al [6], building on a result from Kueng and Gross [32] showed how the average fidelity relates to inputs with random stabilizer states:

Theorem 1 (Burgholzer et. al [6]). *Suppose $|g\rangle$ is a random stabilizer state, and U, V are unitary matrices. Then there is the following relationship between the expectation value and the average fidelity:*

$$\mathbb{E}_{|g\rangle}[\langle g| V^\dagger \cdot U|g\rangle] \approx \mathcal{F}_{\text{avg}}(U,V) \tag{1}$$

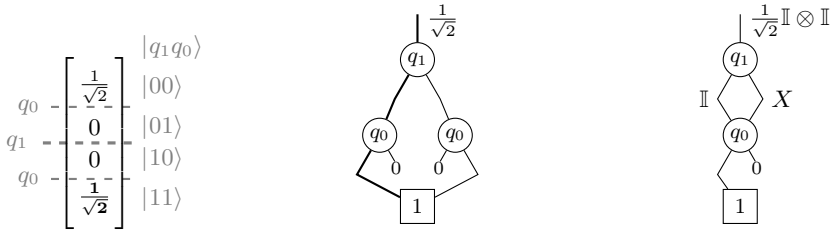
Consequently, the average fidelity $\mathcal{F}_{\text{avg}}(U,V)$ can be approximated by simulating the two circuits on several random stabilizer states. Put another way, this quantifies the statement that a random stabilizer state is to a quantum circuit what a random input is to a classical circuit. This motivates the approach of Burgholzer et al.: they repeatedly generate (two copies of) a random stabilizer state $|g\rangle$ as input, then they classically simulate the two circuits on this input, obtaining states $U|g\rangle, V|g\rangle$. Lastly, they compute the inner product of the output states $\langle g| V^\dagger \cdot U|g\rangle$; if the absolute magnitude of this number is smaller than 1, then $|g\rangle$ is a counterexample which certifies that $U \neq V$.

We use this reduction from circuit verification to circuit simulation as the motivation for the setup in our case study in Section 5.

3 Motivation

Efficient classical simulation of quantum circuits is an important task for the development of both quantum circuits and their compilation toolchains [7], [9]. As reviewed above, this task is conceptually simple (matrix-vector multiplication), but practically hard due to the memory requirements of classical descriptions of quantum states, i.e., state vectors require an exponential amount of memory with respect to the number of qubits. However, certain families of quantum circuits, such as those consisting only of Clifford gates, can be simulated in polynomial time by the stabilizer formalism that exploits the strong algebraic structure present in stabilizer states [12], [13]. These techniques have the disadvantage that they do not encompass all of quantum computing, since they cannot produce all quantum gates, i.e., they are not *universal*. For general quantum circuits, i.e., those with no restrictions on the gate set, decision diagrams as reviewed later in this section are a promising data structure to drastically reduce memory requirements in many cases. However, the stabilizer formalism and decision diagrams have thus far excelled only in their respective areas. LIMDDs unite the capabilities of both worlds and thereby enable efficient representation of multiple classes of quantum states.

Although the verification of a given quantum circuit is likely more difficult than simulation of that circuit on a given input, Burgholzer et al. [6] recently



(a) Vector (b) QMDD (c) Pauli-LIMDD

Fig. 2: Different representations of a Bell state

showed that simulation can nevertheless be very useful for verification as explained in Section 2.3. In particular, they showed that a promising approach is to simulate the circuit on a certain state called a stabilizer state, and gave qualitative and quantitative analytical guarantees on the errors found by this method. Since LIMDDs excel on this family of quantum states, we adopted this approach for our case study in Section 5.

The remainder of this section first reviews the basics of QMDDs and LIMDDs, and then discusses their respective strengths (as far as they have been analytically investigated thus far). Based on this, we motivate the need for an implementation of LIMDDs.

3.1 Quantum Multiple-valued Decision Diagrams (QMDDs)

Representing quantum states and operations in a straightforward fashion as vectors and matrices requires an exponential amount of memory with respect to the number of qubits. Decision diagrams are an established data structure that approach this problem by providing a compact representation by exploiting redundancies in the data in many cases. There are multiple types of decision diagrams for the quantum domain [23]–[27]. We focus on *Quantum Multiple-valued Decision Diagrams* (QMDDs, [26], [33]) since they are the state of the art for decision diagram-based quantum circuit simulation.

Conceptually, the QMDD corresponding to the amplitude vector $|\phi\rangle \in \mathbb{C}^{2^n}$ can be built as follows. First, we repeatedly split the amplitude vector in two equal halves, until the individual amplitudes are reached, thus obtaining a binary tree (of height n), in which a node at height k represents a (sub-)vector of length 2^k. Next, whenever two nodes represent states $|\phi\rangle, |\psi\rangle$ satisfying $|\phi\rangle = \lambda \cdot |\psi\rangle$ for some $\lambda \in \mathbb{C}$, we merge these two nodes (discarding one of the subtrees) and place the factor λ on one of the two incoming edges. This *reduced* QMDD is now a directed acyclic graph (DAG) and no longer a tree. Thus, the way the QMDD exploits structure in the vector is by recognizing repeated sub-vectors (or more precisely: sub-vectors which are equal up to a complex constant), which is how it can avoid the exponential blowup in many cases. Note that this construction is explained for illustration purposes only as working with decision diagrams does

not at any point require explicitly storing the vector representing a quantum state. For a formal definition we refer to [29].

One can efficiently apply a given operation U to a state $|\phi\rangle$ when both are given as QMDDs. This is done by the MULTIPLY algorithm, which recursively traverses the decision diagrams of U and $|\phi\rangle$ and builds the DD corresponding to the state $|\psi\rangle = U \cdot |\phi\rangle$. We briefly sketch how this algorithm works. First, note that if $|\phi\rangle = |0\rangle \otimes |\phi_0\rangle + |1\rangle \otimes |\phi_1\rangle$, then the DD node representing $|\phi\rangle$ has two children, v_0 and v_1, which represent the amplitude vectors $|v_j\rangle = |\phi_j\rangle$ for $j = 0, 1$. Similarly, the matrix $U = \sum_{ij} |i\rangle\langle j| \otimes U_{ij}$ is represented by a QMDD node with four children v_{ij}, with each v_{ij} representing the submatrix U_{ij}, a quadrant of U. The algorithm first constructs decision diagrams for the states $U_{ij} \cdot |\phi_j\rangle$ using four recursive calls to MULTIPLY(U_{ij}, v_j). Next, it construct decision diagrams representing the states $|\psi_i\rangle = U_{i0}|\phi_0\rangle + U_{i1}|\phi_1\rangle$ for $i = 0, 1$ using two calls to a procedure ADD implementing addition on QMDDs. Last, it makes a node whose two children are $|\psi_0\rangle$ and $|\psi_1\rangle$, obtaining a node representing $|\psi\rangle = |0\rangle \otimes |\psi_0\rangle + |1\rangle \otimes |\psi_1\rangle = U \cdot |\phi\rangle$, as intended. We use dynamic programming to store the results of all intermediate, recursive calls to MULTIPLY; as is typically done to avoid the exponential-time behavior occurring when all paths in the DAG are considered. In light of this, we remark that in this work we consider matrices U representing a universal gate set that nonetheless each have a small number of nodes that scales as $\mathcal{O}(n)$.

Thus, while it is instructive to consider how a QMDD may be constructed from a given amplitude vector (in the way described above), our algorithms use a more efficient approach, never "expanding" the decision diagram to its amplitude vector, instead working directly on the DD representation of the vectors and matrices. Indeed, this is the primary strength of decision diagrams in general: that they can work on compressed data without decompressing it first.

Example 5. Consider the quantum state $1/\sqrt{2} \cdot (|00\rangle + |11\rangle)$ from Example 1. Figure 2a shows the corresponding vector, with superimposed information on the splitting by qubit (on the left) and the basis states to each amplitude (on the right). Figure 2b shows the same quantum state represented as QMDD.

Retrieving the amplitude of a given quantum state requires traversing the decision diagram and multiplying the edge weights along the way. For readability, edge weights of 1 are omitted; and edges with weight 0 are cut off and represented as stubs. The bolded path in Figure 2b represents the state $|00\rangle$, and following it gives $1/\sqrt{2} \cdot 1 \cdot 1 = 1/\sqrt{2}$.

While QMDDs enable compact representation of quantum states in many cases, Vinkhuijzen et al. [27] showed they can become exponentially sized for stabilizer states, which can be efficiently simulated classically using the stabilizer formalism as discussed in Section 2.1. These states are the intermediate states of circuits consisting of only Clifford gates, thus preventing QMDDs from simulating such circuits efficiently.

3.2 Local Invertible Map Decision Diagrams (LIMDDs)

LIMDDs remove the limitation of QMDDs that cannot efficiently represent every stabilizer state. They can represent each stabilizer state in polynomial space by not just merging nodes that are equivalent up to a scalar, but also those equivalent up to a LIM transformation while retaining universality. A LIM is similar to the stabilizer generator except that it includes an arbitrary scalar.

Definition 1 (Local Invertible Map (LIM), adapted from [27]). *An n-qubit* Local Invertible Map *(LIM) is an operator P of the form $P = \lambda P_n \otimes \cdots \otimes P_1$, where the matrices $P_i \in \{ \mathbb{I}, X, Y, Z \}$ are local Pauli matrices and $\lambda \in \mathbb{C} \setminus \{0\}$. An isomorphism between two n-qubit quantum states $|\varphi\rangle, |\psi\rangle$ is a LIM P such that $P|\varphi\rangle = |\psi\rangle$. We then say that $|\varphi\rangle$ is isomorphic to $|\psi\rangle$, denoted $|\varphi\rangle \simeq |\psi\rangle$. Note that isomorphism is an equivalence relation.*

In fact, LIMDDs can efficiently apply most Clifford gates to any quantum state (i.e., even to non-stabilizer states), without increasing the size of the diagram by more than a factor two. LIMDDs extend QMDDs by annotating an edge not only with a complex-valued weight, but also with a series of local Pauli gates represented by the LIMs. This allows LIMDDs to represent all states at least as succinctly as QMDDs and stabilizer tableaus. Additionally, LIMDDs can efficiently represent states which cannot be represented efficiently with either the stabilizer formalism or QMDDs [27], for instance $|T\rangle \otimes |G\rangle$, where $|T\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{i\pi/4}|1\rangle)$ and $|G\rangle$ is a stabilizer state.

The interpretation of a LIMDD is similar to that of a QMDD. Each node still corresponds to a complex vector and, when following an edge, the vector given by the child node is still multiplied by the weight on the followed edge. For LIMDDs, rather than only multiplying the vector with a complex scalar, it is now additionally multiplied by the tensor product of the single-qubit gates on the incoming edge. This is illustrated in the following example.

Example 6. Consider again the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ from Example 1. Figure 2c shows the LIMDD of this state. Note that it uses one node less than the corresponding QMDD since it only requires one node q_0 due to the X operation annotated to the left out-edge of q_1. The node labelled q_0 represents the vector $|0\rangle$. From the root, following the left edge from node q_1 gives the vector $1/\sqrt{2} \cdot \mathbb{I}|0\rangle$, while following the right edge gives $1/\sqrt{2} \cdot X|0\rangle = 1/\sqrt{2} \cdot |1\rangle$. Note that correspondence is shown for illustration purposes only as working with decision diagrams does not require explicitly storing the vector at any point.

3.3 The Need for a LIMDD Implementation

As discussed, stabilizer states and Clifford circuits are ubiquitous in many quantum computing algorithms. Moreover, in the context of verifying quantum circuits, stabilizer states serve as good candidates for counterexamples. Therefore, we stand to profit twice from the exponential advantage that LIMDDs promise over existing decision diagrams: first, since stabilizers capture the symmetries

present in all quantum states, LIMDDs likely improve universal simulation; second, when we want to verify a quantum circuit by reduction to simulation with a random stabilizer state, the Pauli-LIMDD is guaranteed to efficiently represent at least the initial state, whereas the QMDD is likely exponential [27, Appendix B].

However, the asymptotic advantage of LIMDDs comes with a price. They require both additional memory for bookkeeping the LIMs on edges and additional time for calculating canonical form of nodes. To the best of our knowledge, it is still unknown how this affects the memory and time use in practice because so far an implementation is absent. Existing implementations of decision diagrams in the classical domain [34]–[38] and the quantum domain [23]–[28] have shown that translating the concept of a decision diagram into an efficient and usable program or library is far from trivial. LIMDDs are no exception to this rule and come with new challenges regarding the handling of the LIMs in the nodes and edges of the decision diagram.

4 Implementation of LIMDDs

As discussed in the previous sections, LIMDDs scale exponentially better in many cases compared to QMDDs. However, this advantage comes with an increased overhead to keep track of the local invertible maps annotated to nodes and edges in the decision diagram. Efficient management of this additional information is paramount to implement LIMDDs efficiently. Further crucial ingredients for efficient decision diagram implementations are canonicity and dynamic programming. Canonicity ensures that the diagram is never larger than necessary and uniquely represents a quantum state (in QMDDs) or Boolean function (in BDDs). Dynamic programming ensures that manipulation operations, such as gate applications or measurements, take polynomial time in the size of the diagram representing the state. To ensure canonicity, the implementation must put nodes in canonical form, as worked out in [27], and also store them in a corresponding table. To implement dynamic programming, LIMs must be normalized and stored in caches.

This section discusses both established techniques in developing implementations of decision diagrams in Section 4.1 as well as new approaches to efficiently handle the LIMs annotated to the edges of LIMDDs in Section 4.2. The established and correspondingly adopted techniques include dynamically-sized unique tables, garbage collection, compute tables, as well as indirect storing of complex numbers [28]. While the aforementioned techniques lay a solid foundation for the LIMDDs, they are not sufficient. Efficient approaches to store and manipulate the annotated local invertible maps are required to exploit the potential of LIMDDs.

4.1 Established Techniques

Implementations of various types of (predominantly classical) decision diagrams have been proposed in the last decades [23]–[27], [34]–[38]. Over this time, a lot

of effort has been put into translating abstract concepts of decision diagrams into concrete instructions that run efficiently on classical computers. Multiple parts and "tricks" of these implementations are reusable for the LIMDD implementation as well. The following list provides a brief description of the most important building blocks:

Unique Tables store the nodes of the decision diagram and enable efficient detection of redundant nodes. These tables are commonly implemented as hash maps storing the nodes with two levels of indirection: the first level gives the qubit (or variable) and the second level is the hash value of the node, which is recursively calculated from the weights of the out-edges and hashes of the respective successors. The subsequent *strong canonical* identifier [39] of a node is the pointer into the memory of the unique tables, enabling access via constant-time de-referencing of the pointer.

When a new node is created, the unique tables are checked for already existing equivalent nodes. If an equivalent node exists, this node is re-used, otherwise the new node is stored in the unique tables.

Compute Tables cache results of operations to implement dynamic programming, avoiding repetitions of the same calculation. Intuitively, the more compact the decision diagram, the more paths (from root to leaf) traverse through the same nodes in the decision diagram. We can avoid processing exponentially many paths by hashing the operands of the recursive operation that traverses these paths in the diagram. The compute tables are implemented as individual hash tables for different operations, where the hash is calculated from the operand nodes.

Additionally, the compute tables are a key concept that enables efficient operations on decision diagrams by enabling dynamic programming. Without them, operations such as multiplication during circuit simulation would always be exponential, since no previous result could be re-used.

Handling of Complex Numbers requires special consideration to ensure that the limited accuracy of floating point numbers does not lead to wrong results. Two key aspects of these considerations are the introduction of a tolerance in the comparison of the components of the complex numbers and storing the components in a dedicated table to exploit the memory address as strong canonical form (with constant-time dereferencing).

Garbage Collection is frequently run to remove entries of the aforementioned tables that are not needed anymore, e.g., after each applied operation in quantum circuit simulation. For each node, a reference count is used to keep track of its state. Upon removal of nodes, garbage collection is also run on the other tables, such as the compute tables, so that no invalid pointers remain in memory—preventing an inconsistent state between the tables and subsequent illegal memory accesses.

The established techniques described above are used in existing packages of decision diagrams for quantum computing. However, LIMDDs require additional functionality to manage the LIMs which are annotated to edges and nodes.

The next section describes the techniques employed to efficiently integrate the information on local invertible maps into the decision diagrams.

4.2 Implementing Local Invertible Maps

Efficient handling of the LIMs annotated to the nodes and edges in the LIMDDs is *the* requirement to actually realize the exponential advantage over QMDDs for certain quantum circuit simulations. Recall that a LIM consists of a complex factor and a Pauli operator $P = P_1 \otimes \cdots \otimes P_n$ with $P_i \in \{\mathbb{I}, X, Y, Z\}$ denoting a local Pauli operator. We call the latter a *Pauli string*.

The proposed LIMDD implementation still provides a strong canonical form for the nodes to ensure canonicity. We implement the canonical form presented in [27], with minor changes described below, which entails, among others, finding the lexicographically minimal Pauli strings for both of a node's outgoing edges, and possibly swapping the two children. The LIMs are stored in a new table to enable constant-time decisions whether two LIMs are equal. Additionally, because the all-identity operator occurs very frequently, we "hardcode" this as null pointer, to prevent many lookups to the LIM table.

In line with existing work [13], we represent a Pauli operator using two bits, so that the operators $\mathbb{I}, X, Y, Z$ are represented by $00, 01, 11, 10$, respectively, and a Pauli string of n operators is stored using $2n + 2$ bits, using 2 extra bits to store a scalar factor in $\{\pm 1, \pm i\}$. This enables efficient multiplication of Pauli operators, namely, the product of two LIMs is obtained by XORing their respective bit strings. For QMDDs, the diagram can be traversed by following edges, which is accomplished simply by dereferencing a pointer and multiplying the weight of the considered edge. For LIMDDs, following an edge is slightly more involved, since the local invertible map can affect each level downwards. To keep track of the LIMs to be applied and to avoid creating a new decision diagram for each followed edge, we keep auxillary information about the current LIM during each step of the traversal.

We now briefly list the biggest changes that are required to turn a QMDD package into a LIMDD package.

Putting nodes in canonical form ensures canonicity, which keeps the diagram as small as possible, by allowing nodes representing redundant subvectors to be merged. We use the canonicity scheme for LIMs as proposed in [27]. In this scheme, a node v always has the identity LIM $\mathbb{I}_2^{\otimes n}$ on its 0-edge and a LIM P on its 1-edge, such that P is the lexicographically minimal LIM possible, in the sense that using any smaller LIM results in a node v' which is not Pauli-isomorphic to v. Since the LIM P depends only on the state vector that the node represents, and is minimal in a precise way, this makes the node canonical. Consequently, the diagram will merge two nodes whenever they represent two Pauli-isomorphic subvectors. This minimal P is found by first finding the stabilizer tableaux (see Section 2.2) of v's two children states, which requires time $\mathcal{O}(n^3)$. This approach amortizes the cost of computing canonical LIMs over the entire DD structure; in other words, to construct

a canonical LIM for node v, we only need to inspect the stored stabilizer generator sets of v's children (and not their descendants). This step, of constructing these groups, presents the biggest added computational overhead of all changes, namely the time required for making a new node increases from $\mathcal{O}(1)$ to $\mathcal{O}(n^3)$ because of the need to find stabilizer groups. Still, as shown in [27], this overhead enables a asymptotically exponential advantage.

Edge Weight Normalization is part of making the node canonical. We employ the normalization scheme from [40], which differs from the one proposed in [27]. Namely, when choosing the weights α_0, α_1 on the out-edges of a node, we require that $|\alpha_0|^2 + |\alpha_1|^2 = 1$, as opposed to requiring the left-most non-zero edge weight to be normalized to 1 (meaning that α_0 has to be either 0 or 1). This allows us to better take advantage of the existing cache for complex numbers and faster sampling from the decision diagram. Given such numbers α_0, α_1, we have a choice between α_1 and $-\alpha_1$, and we choose the one having nonnegative imaginary part; ties are broken by choosing the one with nonnegative real part. If we choose $-\alpha_1$, then we correct for this by multiplying the LIM on the incoming edge by $Z \otimes \mathbb{I}^{\otimes n-1}$.

The **Operation Cache** for LIMDDs allows us to improve the caching of the ADD operation results to potentially be more succinct and achieve more cache hits. Specifically, we get a cache hit on input $A|v\rangle + B|w\rangle$ whenever a previous call to ADD had input $C|v\rangle + D|w\rangle$ satisfying $A^{-1}B|w\rangle = C^{-1}D|w\rangle$. We implement this using the caching algorithm from [27], namely, on input $A|v\rangle + B|w\rangle$, if the result is edge $E|r\rangle$, then we add CACHE$[F, v, w] := E|r\rangle$, where F is a canonically chosen LIM determined by A, B, and w.

The **LIM Table** stores the LIMs on the edges and the LIMs which generate a state's stabilizer group, so that common LIMs are shared in the LIMDD, thus reducing the total memory footprint. Multiple LIMDD sub-routines make use of a state's stabilizer group, e.g., for finding canonical edge labels. We choose to construct a set of LIMs which generate a state's stabilizer group as soon as that state's node is created, using the algorithm by [27], and we store these LIMs in the LIM Table. LIMs that are no longer required are identified via reference counting and removed during garbage collection.

We focus on the efficient storing and manipulation of LIMDDs for quantum states and continue to use QMDDs from the original package [28] to represent quantum operations as explained in Section 3.2, so that simulation can still be conducted in a fashion similar to [33]. This does not present a limitation of the approach, since QMDDs efficiently represent all gates considered in this work. Namely, we only use single-qubit gates with arbitrary controls, and these for gates the size of the diagram scales as $\mathcal{O}(n)$. Therefore, storage of the quantum state, rather than the matrix, remains the main memory bottleneck in common simulation scenarios.

To ensure the validity of the LIMDD implementation, we performed extensive tests on more than 1700 quantum circuits of varying sizes, parameters and complexity, ensuring that the intermediate state after every gate is the same as that found by the QMDD.

The next sections provides a case study that compares LIMDDs against QMDDs based on the Quantum Fourier Transform, which is an important building block of many quantum algorithms.

5 Case Study

In this section, we provide the results obtained by an experimental case study of the implementation presented in this paper. To this end, we created a complete, open-source LIMDD package in C++ available at `https://github.com/cda-tum/ddsim/tree/limdd`, based on an existing open-source implementation for QMDDs provided in [28], [41]. The motivation of the case study is an investigation on the extend that the theoretically proven advantage over QMDDs applies in an actual implementation.

In order to demonstrate the efficacy of the resulting implementation and thereby for the first time empirically comparing LIMDDs and QMDDs, we conducted quantum circuit simulation of a circuit which implements the Quantum Fourier Transform (QFT). The QFT is a common subroutine which is used by many quantum algorithms (notably order-finding in Shor's algorithm, phase estimation, and solving the hidden subgroup problem [30]); thus, verifying the correctness of this circuit is a useful step in the compilation toolchain of many quantum algorithms. We consider QFT circuits for various numbers of qubits, $n = 3 \dots 24$. We simulate the QFT on n qubits using a random n-qubit stabilizer state as the input. This stabilizer state is prepared by prepending a random Clifford circuit with $10 \cdot n$ gates, and then simulating from the initial state $|0\rangle^{\otimes n}$; the output of this circuit is a random stabilizer state.

The evaluation was conducted for QMDDs and LIMDDs on a server running GNU/Linux and GCC-10.3.0 with an AMD Ryzen 9 3950X running at 3.5 GHz and 128 GiB memory.

The results in Figure 3 show that LIMDDs outperform QMDDs on large instances (from $n = 19$ qubits and up), whereas QMDDs outperform LIMDDs on small instances (up to and including 18 qubits). This is most pronounced at 24 qubits, where LIMDDs are about five times faster. These results are expected: LIMDDs are proven to be asymptotically faster, but this comes at the price of adding a lot of computational overhead in the handling of the LIMs on the edges, as explained in Section 4. The data show that this overhead pays off in the long run where the asymptotically better performance becomes realized in practice. In the graph, this translates into the fact that the LIMDD line is less steep than the QMDD line. The LIMDD is still small (it has $\mathcal{O}(n)$ nodes) when it finishes preparing a random stabilizer state and starts simulating the QFT, whereas the QMDD is already very large (it has $2^{\mathcal{O}(n)}$ nodes) at this point. At the end of the QFT, both types of decision diagrams are almost fully populated (i.e., almost of maximum size), since the state after the QFT does not possess much redundancy to be exploited. Generally, applying a gate to a small decision diagram is more efficient than applying the same gate to a large decision diagram, so especially the first few gates of the QFT can be applied quickly by LIMDDs,

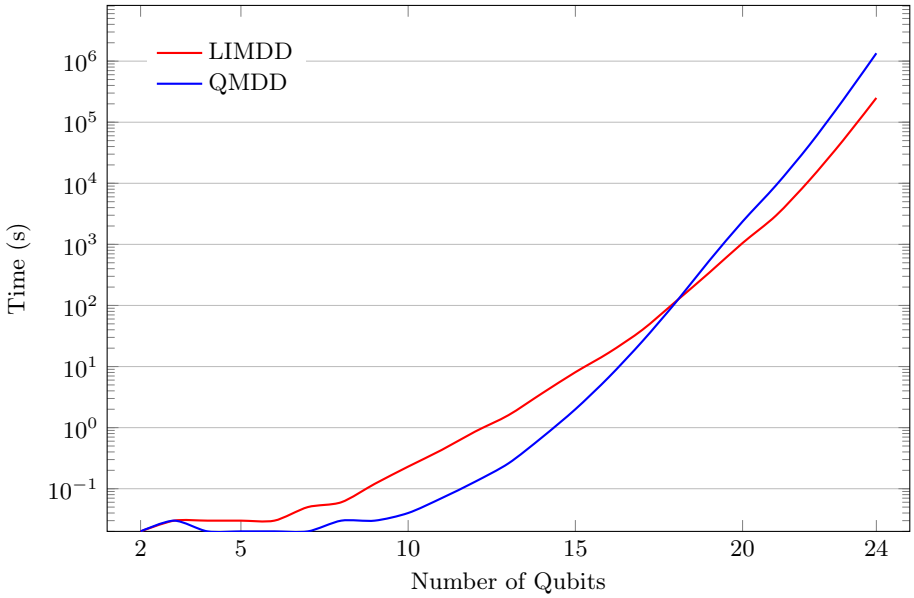


Fig. 3: Quantum Fourier Transform Simulation on Random Stabilizer States

which eventually leads to the better runtime. In summary, while the additional overhead of LIMDDs outweights the lower complexity for small circuits, they can demonstrate their advantage as the circuit size increases.

6 Conclusions

In this paper, we presented the first implementation of *Local Invertible Map Decision Diagrams* (LIMDDs). The implementation includes techniques adapted from other decision diagram packages (both classical and quantum) that are tried and tested, as well as new considerations to efficiently handle *Local Invertible Maps* (LIMs). By this, we enable the potential of LIMDDs to be realized in practice. A case study confirm that LIMDDs provide an advantage for the classical simulation of quantum circuits that exceed a certain complexity, as shown by the Quantum Fourier Transform. The resulting open-source C++ implementation is available under the MIT license via `https://github.com/cda-tum/ddsim/tree/limdd`.

Acknowledgment

This work received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 101001318) and was part of the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus.

References

[1] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Symp. on Theory of Computing*, 1996, pp. 212–219.

[2] A. Montanaro, "Quantum-walk speedup of backtracking algorithms," *Theory of Computing*, vol. 14, no. 1, pp. 1–24, 2018.

[3] A. Ambainis, A. Gilyén, S. Jeffery, and M. Kokainis, "Quadratic speedup for finding marked vertices by quantum walks," in *Symp. on Theory of Computing*, 2020, pp. 412–424.

[4] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Jour. of Comp.*, vol. 26, no. 5, pp. 1484–1509, 1997.

[5] B. P. Lanyon, J. D. Whitfield, G. G. Gillett, *et al.*, "Towards quantum chemistry on a quantum computer," *Nature Chemistry*, vol. 2, no. 2, p. 106, 2010.

[6] L. Burgholzer, R. Kueng, and R. Wille, "Random stimuli generation for the verification of quantum circuits," in *Asia and South Pacific Design Automation Conf.*, New York, NY, USA: Association for Computing Machinery, 2021, pp. 767–772.

[7] L. Burgholzer and R. Wille, "Advanced equivalence checking for quantum circuits," *IEEE Trans. on CAD of Integr. Circ. and Sys.*, vol. 40, no. 9, pp. 1810–1824, 2021.

[8] L. Burgholzer, R. Raymond, and R. Wille, "Verifying results of the IBM Qiskit quantum circuit compilation flow," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE, 2020, pp. 356–365.

[9] J. Carette, G. Ortiz, and A. Sabry, "Symbolic execution of hadamard-toffoli quantum circuits," in *Proceedings of the 2023 ACM SIGPLAN International Workshop on Partial Evaluation and Program Manipulation*, 2023, pp. 14–26.

[10] G. G. Guerreschi and A. Y. Matsuura, "Qaoa for max-cut requires hundreds of qubits for quantum speed-up," *Scientific Reports*, vol. 9, no. 1, p. 6903, 2019.

[11] T. Jones, A. Brown, I. Bush, and S. C. Benjamin, "Quest and high performance simulation of quantum computers," *Scientific Reports*, vol. 9, no. 1, p. 10 736, 2019.

[12] D. Gottesman, "Stabilizer codes and quantum error correction," 1997. arXiv: `quant-ph/9705052`.

[13] S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits," *Phys. Rev. A*, vol. 70, no. 5, p. 052 328, 2004.

[14] D. Gottesman, "Theory of fault-tolerant quantum computation," *Phys. Rev. A*, vol. 57, pp. 127–137, 1 1998.

[15] D. Gottesman, *Stabilizer codes and quantum error correction*. California Institute of Technology, 1997.

[16] C. H. Bennett, H. J. Bernstein, S. Popescu, and B. Schumacher, "Concentrating partial entanglement by local operations," *Phys. Rev. A*, vol. 53, 4 1996.

[17] D. Browne and H. Briegel, "One-way quantum computation," *Quantum information: From foundations to quantum technology applications*, pp. 449–473, 2016.

[18] T. van Dijk, R. Wille, and R. Meolic, "Tagged BDDs: Combining reduction rules from different decision diagram types," in *Formal Methods in CAD*, 2017.

[19] S. Minato, "Zero-suppressed BDDs for set manipulation in combinational problems," in *Design Automation Conf.*, 1993, pp. 272–277.

[20] R. E. Bryant, "Symbolic manipulation of Boolean functions using a graphical representation," in *Design Automation Conf.*, 1985, pp. 688–694.

[21] R. E. Bryant and Y.-A. Chen, "Verification of arithmetic circuits with binary moment diagrams," in *Design Automation Conf.*, 1995, pp. 535–541.

[22] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski, "Efficient representation and manipulation of switching functions based on Ordered Kronecker Functional Decision Diagrams," in *Design Automation Conf.*, 1994, pp. 415–419.

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

[24] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, and S.-Y. Kuo, "An XQDD-based verification method for quantum circuits," *IEICE Trans. Fundamentals*, vol. 91-A, no. 2, pp. 584–594, 2008.

[25] G. F. Viamontes, I. L. Markov, and J. P. Hayes, *Quantum Circuit Simulation*. Springer, 2009.

[26] P. Niemann, R. Wille, D. M. Miller, M. A. Thornton, and R. Drechsler, "QMDDs: Efficient quantum function representation and manipulation," *IEEE Trans. on CAD of Integr. Circ. and Sys.*, vol. 35, no. 1, pp. 86–99, 2016, Implementation available via `http://www.informatik.uni-bremen.de/agra/eng/qmdd.php`.

[27] L. Vinkhuijzen, T. Coopmans, D. Elkouss, V. Dunjko, and A. Laarman, "LIMDD: A decision diagram for simulation of quantum computing including stabilizer states," *CoRR*, vol. abs/2108.00931, 2021. arXiv: `2108.00931`.

[28] A. Zulehner, S. Hillmich, and R. Wille, "How to efficiently handle complex values? Implementing decision diagrams for quantum computing," in *Int'l Conf. on CAD*, 2019, pp. 1–7.

[29] D. M. Miller and M. A. Thornton, "QMDD: A decision diagram structure for reversible and quantum circuits," in *36th International Symposium on Multiple-Valued Logic (ISMVL'06)*, IEEE, 2006, pp. 30–30.

[30] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information (10th Anniversary edition)*. Cambridge University Press, 2016.

[31] R. Jozsa, "Quantum algorithms and the fourier transform," *Royal Society of London. Series A*, vol. 454, no. 1969, pp. 323–337, 1998.

[32] R. Kueng and D. Gross, "Qubit stabilizer states are complex projective 3-designs," *arXiv preprint arXiv:1510.02767*, 2015.

[33] A. Zulehner and R. Wille, "Advanced simulation of quantum computations," *IEEE Trans. on CAD of Integr. Circ. and Sys.*, vol. 38, no. 5, pp. 848–859, 2019.

[34] F. Somenzi, *CUDD: CU decision diagram package release 3.0.0*, `http://vlsi.colorado.edu/~fabio/`, 2015.

[35] T. Van Dijk, A. Laarman, and J. Van De Pol, "Multi-core BDD operations for symbolic reachability," *Electronic Notes in Theoretical Computer Science*, vol. 296, pp. 127–143, 2013.

[36] G. Lv, Y. Chen, Y. Feng, Q. Chen, and K. Su, "A succinct and efficient implementation of a 2^{32} BDD package," in *Int'l Symp. on Theoretical Aspects of Software Engineering*, 2012, pp. 241–244.

[37] M. Herbstritt, *wld: A C++ library for decision diagrams*, `https://ira.informatik.uni-freiburg.de/software/wld/`, 2004.

[38] D. E. Knuth, *The art of computer programming: Binary decision diagrams*, `https://www-cs-faculty.stanford.edu/~knuth/programs.html`, 2011.

[39] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Design Automation Conf.*, 1990, pp. 40–45.

[40] S. Hillmich, I. L. Markov, and R. Wille, "Just like the real thing: Fast weak simulation of quantum computation," in *Design Automation Conf.*, IEEE, 2020, pp. 1–6.

[41] R. Wille, S. Hillmich, and L. Burgholzer, "JKQ: JKU tools for quantum computing," in *Int'l Conf. on CAD*, 2020, 154:1–154:5.