Equivalence Checking Paradigms in Quantum Circuit Design

A Case Study

Tom Peham
Chair for Design Automation,
Technical University of Munich,
Germany
tom.peham@tum.de

Lukas Burgholzer
Institute for Integrated Circuits,
Johannes Kepler University Linz,
Austria
lukas.burgolzer@jku.at

Robert Wille
Chair for Design Automation,
Technical University of Munich,
Germany
Software Competence Center
Hagenberg GmbH, Austria
robert.wille@tum.de

ABSTRACT

As state-of-the-art quantum computers are capable of running increasingly complex algorithms, the need for automated methods to design and test potential applications rises. Equivalence checking of quantum circuits is an important, yet hardly automated, task in the development of the quantum software stack. Recently, new methods have been proposed that tackle this problem from widely different perspectives. However, there is no established baseline on which to judge current and future progress in equivalence checking of quantum circuits. In order to close this gap, we conduct a detailed case study of two of the most promising equivalence checking methodologies—one based on decision diagrams and one based on the ZX-calculus—and compare their strengths and weaknesses.

1 INTRODUCTION

Quantum computing [1] has had a surge in research endeavors by academia and industry in recent years. While quantum computers have not reached a stage of practical usability yet, they promise to outperform classical computers in various important tasks, such as unstructured search, integer factorization, optimization problems, the simulation of molecules and more [2]–[4]. To keep pace with the rapid developments in quantum hardware, various tools have been developed that help in designing corresponding applications.

Initially, a quantum computation is described as a sequence of (high-level) quantum gates—somehow similar to a classical C program. However, just like assembly for a classical processor, the actual machine instructions that may be performed on a given quantum processor are generally restricted to a small (low-level) gate set and might only allow interactions between specific pairs of qubits. Therefore, in order to execute a given circuit on quantum hardware, it needs to be *compiled* to a representation that adheres to all constraints imposed by the targeted device [5]–[8]. Since quantum computers are heavily affected by noise and decoherence, it is paramount to optimize circuits as much as possible in order to maximize the expected fidelity when running the circuit [9]–[12].

Since the compiled quantum circuit might be altered drastically from its original high-level description, it is of utmost importance that the circuit to be executed on the hardware still implements the same functionality as originally intended. Verification of compilation results or, more generally *equivalence checking of quantum circuits*, turns out to be an extremely complex, even QMA-complete [13], task and is in dire need of automation. Although, various methods have been proposed [14]–[21] that tackle the equivalence checking problem from completely different perspectives, a baseline indicating which paradigm is suited best for which use-case is yet to be established.

In this work, we address this issue by first reviewing the quantum circuit equivalence checking problem and arising issues unique to the quantum domain. Then, we show how two of the most promising and publicly available equivalence checking paradigms—one based on quantum decision diagrams [21]–[26] and one based on the ZX-calculus [18], [27]–[29]—tackle the immense complexity. Based on that, we conduct a detailed case study in order to establish a baseline for the current state of the art in equivalence checking of quantum circuits considering a large range of benchmarks.

The remainder of this paper is structured as follows: Section 2 provides the necessary background for this work. Then, Section 3 describes the considered problem and the related work. Based on that, Section 4 and Section 5 review how decision diagrams as well as the ZX-calculus, are used to tackle the complexity of equivalence checking. Section 6 summarizes the results of the conducted case study and discusses the resulting consequences. Finally, Section 7 concludes this paper.

2 BACKGROUND

To keep this work self contained, the following sections provide a brief overview of quantum computing and quantum circuit compilation. We refer the interested reader to the provided references for a more thorough introduction.

2.1 Quantum Computing

In classical computing, information is encoded in classical bits that can be either 0 or 1. Analogously, in quantum computing, *quantum bits* (or *qubits* in short) are used which can be either in the $|0\rangle$ or $|1\rangle$ state (in Dirac notation). Contrary to the classical domain, qubits can also be in *superposition* of multiple states. Formally, the state $|\phi\rangle$ of a qubit is written as

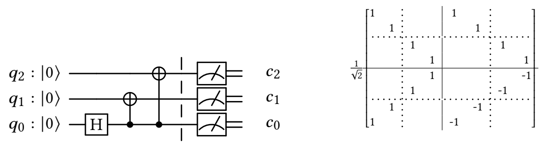
$$|\phi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle = \alpha_0 \begin{bmatrix}1\\0\end{bmatrix} + \alpha_1 \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}\alpha_0\\\alpha_1\end{bmatrix}$$

with *amplitutes* $\alpha_0, \alpha_1 \in \mathbb{C}, |\alpha_0|^2 + |\alpha_1|^2 = 1$.

The basis states of multi-qubit systems are obtained as the *tensor product* of single qubit states. So a basis state of a 3-qubit system would for example be written as $|1\rangle \otimes |1\rangle \otimes |0\rangle = |110\rangle =: |6\rangle$. In general, an n-qubit state $|\phi\rangle$ is described by a linear combination of basis vectors, i.e.,

$$\sum_{i=0}^{2^n-1} \alpha_i |i\rangle \text{ with } \sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1 \text{ and } \alpha_i \in \mathbb{C}.$$

Any operation manipulating the state of a quantum system must again yield a valid quantum state. As a consequence, any



(a) GHZ state preparation circuit G **(b) System matrix U of G**

Figure 1: GHZ state preparation

such operation U must be *unitary*, i.e., it must obey the equation $UU^\dagger = U^\dagger U = I$ where $U^\dagger$ is the *conjugate transpose* of U and I is the identity transformation.

Example 1. *Consider the* Hadamard transform $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$. *It can be easily checked by matrix multiplication that H is a unitary transformation. The Hadamard transform maps* Z-basis *states to* X-basis *states, i.e.,*

$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle =: |+\rangle\,,\; H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle =: |-\rangle\,.$$

An important unitary acting on two qubits is the controlled not *or CNOT gate. It is defined by the matrix* $\begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&1 \\ 0&0&1&0 \end{bmatrix}$ *and flips the second qubit (the* target*) when the first qubit (the* control*) is in state* $|1\rangle$.

A quantum computation is a unitary transformation acting on some initial state (usually the qubits are all prepared to be $|0\rangle$). Instead of writing the *system matrix* (i.e., the unitary describing the behaviour of the whole circuit) explicitly, a common way to describe the *unitary evolution* of a quantum system is through *quantum circuit* notation [1]. There, qubits are represented by wires and operations (called *gates*) are annotated as boxes and circles on the wires. The evolution of the initial state is read from left to right. Thus, a quantum circuit G is described as a sequence of gates $g_0 \ldots g_{m-1}$. Due to their unitary nature, quantum circuits are inherently reversible. More specifically, the inverse of a quantum circuit $G = g_0 \ldots g_{m-1}$ is obtained by inverting each gate and reversing the order of operations, i.e., $G^\dagger = g_{m-1}^\dagger \ldots g_0^\dagger$.

Example 2. *The circuit G in Fig. 1a represents a 3-qubit system. The box annotated with H is a Hadamard transform on qubit q_0 and the connected circles and dots are CNOT gates with q_0 as control and q_1 and q_2 as target qubit, respectively. The circuit maps $|000\rangle$ to $\frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle$, the well-known GHZ state [30]. The system matrix describing the unitary this circuit realizes is given in Fig. 1b.*

2.2 Quantum Circuit Compilation

Quantum algorithms are typically designed at a rather high abstraction level without considering specific hardware restrictions. In order to execute a conceptual quantum algorithm on an actual device, it has to be *compiled* to a representation that conforms to all restrictions imposed by the targeted device. Since quantum computers typically only support a limited gate set, every high-level operation has to be *decomposed* into that gate set [31]–[33]. In addition, many architectures (such as those based on superconducting qubits) restrict the pairs of qubits that operations may be applied to. Hence, it is necessary to *map* the decomposed circuit to the device such that it adheres to the device's coupling constraints [34]–[36]. In general, this is accomplished by establishing a mapping between the circuit's logical qubits and the device's physical qubits. Since it is generally not possible to determine a conforming mapping in a static fashion, SWAP gates are inserted into the circuit that allow to dynamically change the logical-to-physical qubit mapping.

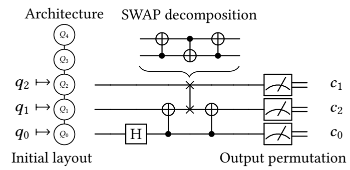


Figure 2: Compilation of GHZ state preparation circuit

Example 3. *Consider again the GHZ preparation circuit shown in Fig. 1a and assume it shall be mapped to the 5-qubit, linear architecture shown on the left-hand side of Fig. 2. Assume that, initially, logical qubit q_i is mapped to physical qubit Q_i for $0 \leq i \leq 2$. Then, the first two operations can be directly applied, while the last operation cannot — due to the fact that Q_0 and Q_2 are not directly connected on the architecture. Hence, a SWAP operation between Q_2 and Q_1 is introduced, which allows to execute the final gate. At the end of the circuit q_0 is measured on Q_0, q_1 on Q_2 and q_2 on Q_1.*

Eventually, compilation yields a new circuit that might look quite different to the original high-level description. It is essential for the successful execution of a quantum computation to verify that the compiled circuit still implements the same functionality as the original one. To this end, methods to check the equivalence of quantum circuits are necessary.

3 EQUIVALENCE CHECKING

In general, given two quantum circuits

$$G = g_0 \ldots g_{m-1} \text{ and } G' = g'_0 \ldots g'_{m'-1}$$

with corresponding system matrices

$$U = U_{m-1} \cdots U_0 \text{ and } U' = U'_{m'-1} \cdots U'_0,$$

the *equivalence checking problem for quantum circuits* asks whether

$$U = e^{i\theta} U' \text{ or, equivalently, } U^\dagger U' = e^{i\theta} I,$$

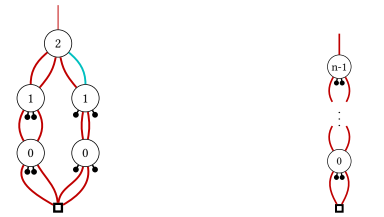
where $\theta \in (-\pi, \pi]$ denotes a physically unobservable global phase.

So, in principle, checking the equivalence of two quantum circuits reduces to the construction and the comparison of the respective system matrices. While this is straight-forward conceptually, it quickly becomes a difficult task due to the exponential scaling of the involved matrices in the number of qubits. Equivalence checking of quantum circuits has even been shown to be QMA-complete [13].

One of the biggest, yet hardly talked about, practical issue when actually conducting equivalence checking concerns numerical inaccuracies. Because quantum gates are described by matrices over $\mathbb{C}$, they are hard to accurately represent in memory. Usually, these matrices are stored using floating point numbers which leads to imprecisions and rounding errors. Therefore, Comparing two matrices for exact equality becomes pointless in many practical cases. Instead, the Hilbert-Schmidt inner product can be used to quantify the similarity between two matrices. Let tr denote the trace of a matrix, i.e., the sum of its diagonal elements. Then, because $\mathrm{tr}(I) = 2^n$ for the identity transformation on n qubits, one can check whether $|\,\mathrm{tr}(U^\dagger U')| \approx 2^n$ in order to conclude the equivalence of both circuits up to a given tolerance.

Further considerations have to be made when comparing circuits which might have different initial layouts and output permutations. Compilation flows use a circuit's initial layout and output permutation as an additional degree of freedom for saving SWAP operations, as, e.g., illustrated in Example 3. Hence, in order to verify the equivalence of compilation flow results, any equivalence checking routine must be able to handle these kind of permutations.

In order to avoid the emergence of a verification gap as for classical systems, automated software solutions for equivalence checking



(a) System Matrix U **(b) n-qubit Identity**

Figure 3: Decision Diagrams

of quantum circuits have to be developed. To this end, various methods have been proposed [14]–[21]. However, most of them either only work on small circuits, lack publicly available implementations, or are based on paradigms established in classical computing that do not take the full picture of quantum computing into account. Few methods exist that approach equivalence checking entirely from the perspective of quantum computing [18]–[20]. Even these existing approaches view the equivalence checking problem from completely different perspectives and a baseline indicating which paradigm is suited best for which use-case is yet to be established.

In the following two sections, we review how two of the most promising and publicly available equivalence checking paradigms—one based on decision diagrams [21]–[26] and one based on the ZX-calculus [18], [27]–[29]—provide means to efficiently check the equivalence of quantum circuits.

4 DECISION DIAGRAMS

We have seen in the previous section that verification of quantum circuits by constructing their system matrices is infeasible in general due to the exponential growth of the matrices' dimensions with respect to the number of qubits. But it might actually not be necessary to explicitely represent every entry of the matrix in memory. *Decision Diagrams* [21]–[26] have proven effective in efficiently representing and manipulating quantum states and transformations in many cases. By exploiting *redundancies* in the vectors and matrices, it is often possible to significantly reduce the necessary memory, sometimes even exponentially.

Given a matrix U, the matrix is divided into equally-sized submatrices $U = \left[\begin{array}{c|c} U_{00} & U_{01} \\ \hline U_{10} & U_{11} \end{array}\right]$, where U_{ij} denotes the action of U given the considered qubit is mapped from j to i. Every (sub-)matrix is represented by a node in the decision diagram and an edge is created for each U_{ij} connecting its node to the node representing U. This procedure is then recursively applied to each of the matrices U_{ij} until only complex numbers remain, thus building up the decision diagram. If any two sub-matrices are identical up to a constant factor their decision diagrams can be identified with each other and, therefore, do not have to be represented twice. The factors are stored as edge-weights in the diagram. This *sharing* of structure is what allows compact representations of many quantum circuits.

Example 4. *Consider again the system matrix shown in Fig. 1b. We can see that $U_{00} = U_{01}$ and $U_{10} = (-1)U_{11}$.The decision diagram for the system matrix is given in Fig. 3a. To this end, we adopt the decision diagram visualization method proposed in [37], where thickness and color of an edge represent the edge weight's magnitude and phase, respectively. Obviously, the decision diagram representation is much more compact than the whole matrix.*

4.1 Equivalence Checking using Decision Diagrams

Decision diagrams are predestined for verification, because they are canonical (with respect to a particular variable order and normalization criterion), i.e., there are no two different decision diagrams for the same functionality. Once the decision diagrams for both circuits G and G' in question are constructed, it suffices to compare their root pointers and the corresponding top edge weight [23]. While this is true in theory, the diagrams might not be exactly identical due to numerical impreci7sions (as discussed in Section 3). Thus, further, potentially expensive, operations might be necessary to decide the equivalence of both circuits. Furthermore, the resulting decision diagrams are still exponentially large in the worst case.

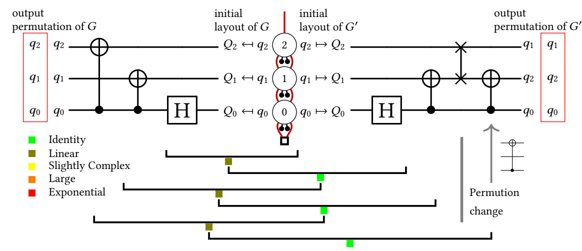


Figure 4: Verifying equivalence with decision diagrams

If G and G' are equivalent, then it holds that $G'G^\dagger = I$, i.e., concatenating one circuit with the inverse of the other implements the identity. Since the identity has a perfectly compact representation as a decision diagram, being linear in the number of qubits (as shown in Fig. 3b), the decision diagram for the combined circuit $G'G^\dagger$ can be constructed instead. However, building up the decision diagram of $G'G^\dagger$ sequentially from left to right might still result in an exponentially large decision diagram, since eventually the whole decision diagram for G' is constructed in the middle of the computation. The solution is to start constructing the functionality of the combined circuit from the "middle" and alternating between applications of $G^\dagger$ and G', such that the decision diagram being constructed remains as close to the identity as possible [20]. The strategy when to choose gates from which circuit is dictated by an *oracle*. If more information about the relation between G and G' is known, a more sophisticated oracle can be employed, e.g., for verifying the results of compilation flows [38]. This method also makes it easier to check equivalence of circuits up to some precision using the inner product $\mathrm{tr}(U^\dagger U')$, since the product $U^\dagger U'$ is inherently constructed during the equivalence check—saving a potentially expensive decision diagram multiplication.

As discussed in Section 3, a compiled circuit might act on different qubits than the original circuit due to the logical-to-physical qubit mapping. This can be accounted for by tracking the permutation of each circuit's qubits throughout the equivalence check and applying all operators according to that permutation. During this process, any SWAP operation can be translated to a change of the corresponding permutation. To maximize this potential, deconstructed SWAP operations (as in Fig. 2) are reconstructed. In the end, the tracked permutation is compared to the expected one and SWAP operations are executed to correct any potential mismatch. In this fashion, circuits with permuted inputs and/or outputs can be verified using the same methodology. The initial layout and the output permutation need to be known a-priori in order to properly check the equivalence of circuits.

Example 5. *Consider the circuits G and G' shown in Fig. 1a and Fig. 2, respectively. Fig. 4 shows an example of how the two circuits are verified using the decision diagram-based approach described above. Note that the decomposed SWAP has been reconstructed in G'.*

The equivalence checking process starts off with the identity diagram (shown in the middle of Fig. 4). Then, gates are applied in an

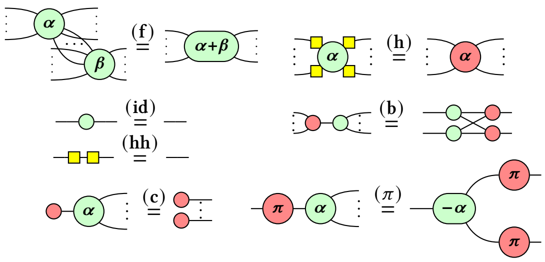
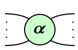


Figure 5: Axioms of the ZX-calculus

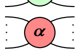
alternating fashion from $G^\dagger$ and G'. First, the Hadamard from $G^\dagger$ is applied to the decision diagram. After applying the corresponding Hadamard from G' the diagram is reduced back to the identity. Then, the CNOT gates on both sides are applied and the diagram is again back to the identity. After applying the last operation from the left, instead of applying the SWAP gate to the decision diagram, the tracked permutation of G' is updated. Because of this the last CNOT gate on the right-hand side is applied to qubits Q_2 and Q_0 instead of Q_1 and Q_0—again yielding the identity. In the end, the tracked permutation is compared to the expected one. Because they are identical, no corrections have to be made. Since the final decision diagram resembles the identity, it can be concluded that the circuits are equivalent.

5 ZX-CALCULUS

The ZX-calculus [18], [27]–[29] is a graphical notation for quantum circuits equipped with a powerful set of rewrite rules that enable diagrammatic reasoning about quantum computing. A ZX-diagram is made up of colored nodes (called *spiders*) that are connected by wires. Each spider can either be green (Z-spider) or red (X-spider) and is attributed a scalar phase which is omitted if the phase is 0. Any quantum circuit can be interpreted as a ZX-diagram (but not the other way around). ZX-diagrams have the following interpretation as transformations of qubits:



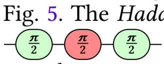
$$= |0 \dots 0\rangle \langle 0 \dots 0| + e^{i\alpha} |1 \dots 1\rangle \langle 1 \dots 1|$$



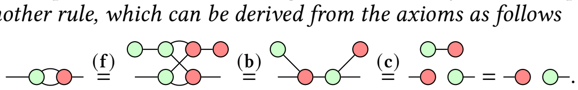
$$= |+ \cdots +\rangle \langle + \cdots +| + e^{i\alpha} |- \cdots -\rangle \langle - \cdots -|$$

Spiders without inputs are called *states*, whereas spiders with no outputs are called *effects*. Even though wires connected to spiders can be thought of as inputs and outputs the "only topology matters" paradigm of the ZX-calculus makes this distinction redundant.

ZX-diagrams can be composed just like quantum circuits. Horizontal composition is achieved by connecting the outputs of one diagram to the input of another. Vertical composition is achieved by simply "stacking" two diagrams on top of each other. Additionally, a ZX-diagram can carry a global phase that is annotated along the diagram. Since global phases are negligible in most cases, they are frequently omitted from ZX-diagrams and equations in the ZX-calculus usually hold up to a global phase.

The power of ZX-diagrams becomes evident when adding rewrite rules to the language. The axioms of the ZX-calculus are given in Fig. 5. The *Hadamard box*  is a notation for the ZX-diagram  and represents the Hadamard-gate. For an in-depth introduction to the ZX-calculus, we direct the reader to [27], [39].

Example 6. *To give a feel for how to work with ZX-diagrams, we are going to prove the well-known equivalence of a SWAP with 3 CNOT operations (as shown in Fig. 2). For this, we first need to prove another rule, which can be derived from the axioms as follows*

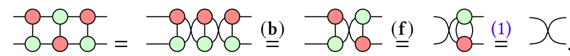


(1)

(a) ZX-diagram of Circuit G (b) ZX-diagram of Circuit G'

Figure 6: ZX-diagrams of GHZ state preparation circuits

With this we can proceed with

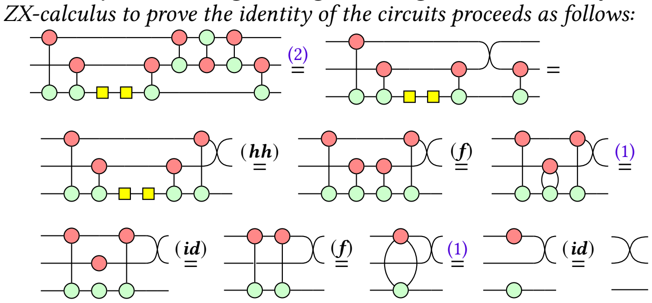


(2)

5.1 Equivalence Checking using the ZX-Calculus

The ZX-calculus has proven useful as an intermediate language when compiling and optimizing quantum circuits [18], [28], [40]. But it can also be used to verify the equality of two quantum circuits, either by rewriting the diagram of both circuits into one another (as in Ex. 6) or, similarly to the approach described in the previous sections, by inverting one diagram, composing the diagrams and simplifying as much as possible. If the composed diagram simplifies to a diagram composed only of bare wires, it is either the identity or contains swaps, i.e., resembles a permutation. As with decision diagrams, the permutation of the wires can be checked against the expected permutation. If they match, the circuits are equivalent.

Example 7. *Consider again the circuits G from Fig. 1a and G' from Fig. 2. Their respective ZX-diagrams are shown in Fig. 6a and Fig. 6b. Since all phases in all spiders are 0, the inverse of each diagram is obtained by just reversing the diagram. Using the rewrite rules of the ZX-calculus to prove the identity of the circuits proceeds as follows:*



The diagram contains a SWAP which permutes qubit Q_1 and Q_2. Since this is what we expect from the output permutation shown in Fig. 2 it can be concluded that the circuits are equivalent.

This example shows that the ZX-calculus can not only show the equivalence of circuits but that it can also provide a proof certificate in the form of the order of rewrite rules that are applied to derive the identity. A natural question to ask is whether the ZX-calculus is powerful enough to derive the identity for any pair of functionally equivalent circuits. The good news is that the ruleset provided in this paper is complete for circuits solely composed of Clifford gates [41]. The bad news is that, in order to achieve completeness for universal quantum computing, the ruleset has to be extended with a rule involving complicated iterated trigonometric functions [42], which makes it difficult to apply in automated reasoning.

Another important property of rewriting systems, such as the ZX-calculus, is the existence of normal forms. Normal forms are needed to determine whether a rewrite procedure terminates. Again, the basic ZX-calculus described here does not have a simple notion of a normal form. For some derivations (like Eq. (1)), the complexity of a diagram has to increase in order to eventually simplify. In [29] the authors define a normal form for ZX-diagrams called *reduced gadget form* which are based on *graph-like diagrams* from [28] . With the addition of several rewriting rules, graph-like diagrams provide a formalism that can be used to automatically simplify diagrams by simply repeatedly applying these rules until a (non-unique) reduced gadget form has been obtained. As discussed above the ZX-calculus as presented here is not complete for universal quantum computing. It is therefore not guaranteed that this algorithm can reduce all circuits in an equivalence checking problem to the identity.

Of course such a simplification procedure is subject to any numerical problems that arise from working with finite precision representations of complex numbers due to rounding errors. Due to these numerical problems, the simplification procedure might not be able to derive the identity. But because the number of spiders are non-increasing during the equivalence checking procedure, the size of the diagram does not blow up. Therefore, the size of the diagram — in terms of the number of spiders — is bounded by the initial ZX-diagram representation of the quantum circuit. Especially, ZX-diagrams are not as sensitive to the structure of the underlying system matrix as decision diagrams. It is also not expected that the run time of the equivalence check increases because it only depends on the number of rewrites that can be applied. Due to the inability to derive the identity diagram, the simplification procedure actually terminates prematurely.

6 CASE STUDY

Both methods presented in the previous sections are implemented and publicly available as Python libraries called QCEC (which is part of the JKQ toolset [43]) and pyzx [44]. Using either method merely requires a few lines of code. Even though both methods are presented as out-of-the-box solutions, some precautions still have to made to allow for a fair comparison. To this end, we first describe the experimental setup and, afterwards, provide a detailed discussion on the obtained results.

6.1 Experimental Setup

While there is no explicit configuration for pyzx, QCEC has different methods with their respective parameters based on [20], [38], [45]. For all the evaluations, we compare the equivalence checking routine of pyzx with the combined approach as presented in [20]. In pyzx, the ZX-diagrams of the circuits are combined as discussed above, transformed into a graph-like diagram and then simplified as much as possible using the local complementation and pivoting rules. For QCEC, we run the equivalence checking routine described in Section 4.1 in parallel with a sequence of 16 simulation runs. If the simulations manage to prove non-equivalence of the circuits, the equivalence checking routine is terminated early.

In order to compare both methods, various benchmarks have been considered. QCEC has been previously evaluated on a benchmark set of reversible circuits (from [46]) which are mapped to suitable quantum architectures. We also use these in our evaluation as well as a selection of common quantum circuits.

All benchmarks are provided in the form of QASM [47] files, which serves as a common language for both tools. Because pyzx does not natively support all gates of the QASM standard (especially, no multi-controlled Toffoli gates) the circuits need to be compiled to a gate set that pyzx can work with. All circuits have been compiled using *qiskit-terra* 0.18.3 with the default optimization level ($O1$).

We distinguish two use-cases: The first is concerned with verifying the compilation result of a high-level circuit. To this end, the circuits are compiled to the 65-qubit IBM Manhattan architecture with a gate set comprised of arbitrary single qubit rotations and the CNOT gate. The second use-case is about verifying the equivalence of two different implementations of the same functionality—an original circuit and an optimized version.

For each use-case we consider three configurations. First, two circuits that are indeed equivalent are used as input. Then, two instances are created where errors are injected into one of the circuits—one with a random gate removed and one where the control and target of one CNOT gate has been swapped. In the following, we summarize the results of our evaluations by means of a representative subset of benchmarks[1]. The obtained results are shown in Table 1.

All computations were conducted on a 4.2 GHz Intel i7-7700K machine running Ubuntu 18.04 and 32 GiB main memory. Each benchmark was run with a hard timeout of 1 h for each method.

6.2 Discussion

Both methods managed to prove the correct result for all considered circuits where a result is obtained within the given time frame. As discussed before, this is not guaranteed by theory for the ZX-calculus. In the considered examples it works out because a lot of the non-Clifford phases cancel in the rewriting procedure because we are dealing with circuits that are supposedly each others inverses. On the other hand, the question of completeness for the decision diagram based approach is trivial. Decision diagrams are a canonical representation of a matrix. Thus, if the combined circuit $G^\dagger G'$ has the identity system matrix, the decision diagram for $G^\dagger G'$ has to be the identity decision diagram as well.

For the set of reversible benchmarks, the two methods finished within 10 s of each other for 82 % of benchmark instances. The remaining reversible benchmarks and circuits containing large reversible parts in their high-level description (such as Grover's algorithm and the Quantum Random Walk) favor the decision diagram-based approach. These circuits can be *exactly* compiled to polynomially-sized quantum circuits comprised only of Clifford+T gates, i.e., circuits only using Hadamard (H), Phase (S), CNOT (CX), and T gates. As a consequence, the respective functionalities (i.e., the system matrices) possess lots of structure that can be exploited by decision diagrams and, additionally, only feature a very limited set of complex numbers which limits the effect of numerical instabilities. In contrast, the ZX-calculus based approach does not benefit from this structure very much. The simplification approach from [29] separates the ZX-diagram into Clifford phases and so-called phase gadgets that introduce non-Clifford phases into the diagrams. Since the circuits in question contain a large number of non-Clifford gates, there is no apparent benefit for ZX-diagrams.

For circuits containing no or smaller reversible parts (such as the QFT or Quantum Phase Estimation), the ZX-calculus approach fairs much better in comparison to decision diagrams. The main obstacle in these cases is that the considered algorithms feature many rotation gates with arbitrarily small rotation angles. Due to numerical instabilities and rounding errors, it might happen that two decision diagram nodes that should be identical in theory, differ by a small margin in practice. As a consequence, inherent redundancies in the underlying representations cannot be captured accurately anymore. Thus, while the resulting decision diagram is very close to the identity with respect to the Hilbert-Schmidt norm, it might grow exponentially large in the worst case. In contrast, ZX-diagrams do not seem to be susceptible to such exponential growth under numerical errors in general.

The above observations are similar in the case of non-equivalent instances. Although runtimes for both methods are generally lower, the relative performances are still similar. Since the resulting decision diagram is almost guaranteed to not be very close to the

[1]The full benchmark set is publicly available at https://github.com/cda-tum/qcec.

Table 1: Benchmarks

Benchmark				Equivalent		1 Gate Missing		Flipped CNOT	
Name	n	$\|G\|$	$\|G'\|$	t_{pyzx}[s]	t_{qcec}[s]	t_{pyzx}[s]	t_{qcec}[s]	t_{pyzx}[s]	t_{qcec}[s]
Compiled Circuits									
Grover	6	1606	2803	3.4	3.40	2.14	0.04	3.17	0.04
Grover	7	4732	8476	25.58	0.30	6.75	0.14	24.60	0.14
Grover	8	12482	22860	198.50	0.91	34.02	0.42	117.76	0.39
QFT	23	1311	3741	3.16	2.00	1.46	>3600	2.60	902.99
QFT	38	3591	10449	14.09	>3600	6.61	>3600	17.55	>3600
Random-Walk	7	6523	8955	166.46	0.24	13.89	0.14	55.44	0.16
Random-Walk	8	14084	19755	1475.85	0.57	672.92	0.33	754.95	0.31
Random-Walk	9	29325	41942	>3600	1.31	>3600	0.59	>3600	0.50
QPE-Exact	22	1217	3006	2.73	0.10	1.47	>3600	2.26	0.82
QPE-Exact	39	3823	11552	8.53	>3600	5.47	>3600	9.42	>3600
GHZ	65	130	493	0.06	<0.01	0.06	<0.01	0.59	0.01
Graph State	62	403	2041	0.36	0.17	0.43	0.17	0.35	0.17
Optimized Circuits									
urf2-154	20	52532	44615	2956.58	129.72	90.72	0.27	188.03	0.27
plus63mod4096-163	53	94520	82195	>3600	201.20	378.70	0.655	>3600	0.67
example2-231	53	22016	19411	475.92	18.02	83.51	0.16	602.70	0.14
Grover	8	12479	12287	174.815	0.04	53.85	0.24	137.66	0.04
Grover	9	37193	36881	1521.54	0.14	519.65	129.56	1491.29	0.17
Grover	10	104977	104501	>3600	0.42	2131.22	> 3600	>3600	41.24
QFT	32	2544	2482	1.43	0.04	1.49	3.57	1.89	14.53
QFT	43	4601	4502	2.86	10.837	2.86	17.78	2.81	1.02
QFT	44	4818	4702	3.01	>3600	3.05	1.27	2.90	1.21
Random-Walk	7	6523	5875	150.661	0.02	19.75	0.10	19.75	0.02
Random-Walk	8	14084	12802	938.19	0.04	89.79	0.10	696.34	0.11
Random-Walk	9	29325	26769	>3600	0.09	>3600	0.11	1914.26	0.11

identity during the equivalence check, the alternating scheme discussed cannot be as efficient as in the equivalent case. Due to this, QCEC resorts to simulations of the circuit with random inputs which, as shown in [20], are expected to show the non-equivalence within a few simulations. Yet, the complexity of decision diagram-based simulation is still exponential in the worst case. The rewriting approach of the ZX-calculus is less volatile to errors in the circuit. During the equivalence check, the combined circuit diagram is simplified as much as possible until no more rules can be applied. Depending on the severity and kind of error, the procedure stops sooner or later. This is not a proof of non-equivalence, but as we see from our evaluations, it gives a strong indication.

7 CONCLUSION

In this work, we examined the effectiveness of decision diagrams and the ZX-calculus for the equivalence checking of quantum circuits. While they show similar performance in many cases, they differ in key areas. Decision diagrams show significant benefits for circuits containing large reversible parts, such as oracles or adders. The sensibility of decision diagrams to numerical imprecision makes them hard to use on quantum algorithms that cannot be exactly represented using floating points, such as algorithms relying on arbitrary rotation angles, due to the potential blow-up of the intermediate representation. The ZX-calculus based equivalence checking procedure is less sensitive to this and is useful in showing equivalence in these cases. However, the ZX-calculus tends to be more suitable for verifying smaller building blocks than whole quantum algorithms due to the large number of involved gates. In conclusion, we can see that decision diagrams and the ZX-calculus can serve as complementary approaches for the equivalence checking problem.

Acknowledgements

This work received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 101001318), was part of the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus, and has been supported by the BMWK on the basis of a decision by the German Bundestag through project QuaST.

REFERENCES

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.
[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," *Proc. of the ACM*, 1996.
[3] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, 1997.
[4] M. Cerezo *et al.*, *Variational quantum algorithms*, 2020. arXiv: 2012.09265.
[5] S. Sivarajah *et al.*, "T|ket>: A retargetable compiler for NISQ devices," *Quantum Sci. Technol.*, 2020.
[6] M. Amy and V. Gheorghiu, "Staq—A full-stack quantum processing toolkit," *Quantum Sci. Technol.*, 2020.
[7] K. N. Smith and M. A. Thornton, "A quantum computational compiler and design tool for technology-specific targets," in *Int'l Symp. on Computer Architecture*, 2019.
[8] T. Häner *et al.*, "A software methodology for compiling quantum programs," *Quantum Sci. Technol.*, 2018.
[9] W. Hattori and S. Yamashita, "Quantum circuit optimization by changing the gate order for 2D nearest neighbor architectures," in *Int'l Conf. of Reversible Computation*, 2018.
[10] M. Soeken *et al.*, "Window optimization of reversible and quantum circuits," in *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2010.
[11] T. Itoko *et al.*, "Optimization of quantum circuit mapping using gate transformation and commutation," *Integration*, 2020.
[12] Y. Nam *et al.*, "Automated optimization of large quantum circuits with continuous parameters," *npj Quantum Inf*, 2018.
[13] D. Janzing *et al.*, ""Non-identity check" is QMA-complete," *Int. J. Quantum Inform.*, 2005.
[14] S. Yamashita and I. L. Markov, "Fast equivalence-checking for quantum circuits," in *Int'l Symp. on Nanoscale Architectures*, 2010.
[15] L. Berent *et al.*, *Towards a SAT encoding for quantum circuits: A journey from classical circuits to Clifford circuits and beyond*, 2022. arXiv: 2203.00698.
[16] X. Hong *et al.*, *Approximate equivalence checking of noisy quantum circuits*, 2021. arXiv: 2103.11595.
[17] G. F. Viamontes *et al.*, "Checking equivalence of quantum circuits and states," in *Int'l Conf. on CAD*, 2007.
[18] A. Cowtan *et al.*, *A generic compilation strategy for the unitary coupled cluster ansatz*, 2020. arXiv: 2007.10515.
[19] M. Amy, "Towards large-scale functional verification of universal quantum circuits," in *International Conference on Quantum Physics and Logic*, 2019.
[20] L. Burgholzer and R. Wille, "Advanced equivalence checking for quantum circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2021.
[21] S.-A. Wang *et al.*, "An XQDD-based verification method for quantum circuits," in *IEICE Trans. Fundamentals*, 2008.
[22] G. F. Viamontes *et al.*, "Gate-level simulation of quantum circuits," in *Asia and South Pacific Design Automation Conf.*, 2003.
[23] P. Niemann *et al.*, "QMDDs: Efficient quantum function representation and manipulation," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2016.
[24] D. Miller and M. Thornton, "QMDD: A decision diagram structure for reversible and quantum circuits," in *Int'l Symp. on Multi-Valued Logic*, 2006.
[25] A. Zulehner and R. Wille, "Advanced simulation of quantum computations," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2019.
[26] A. Zulehner *et al.*, "How to efficiently handle complex values? Implementing decision diagrams for quantum computing," in *Int'l Conf. on CAD*, 2019.
[27] J. van de Wetering, *ZX-calculus for the working quantum computer scientist*, 2020. arXiv: 2012.13966.
[28] R. Duncan *et al.*, *Graph-theoretic simplification of quantum circuits with the ZX-calculus*, 2019. arXiv: 1902.03178.
[29] A. Kissinger and J. van de Wetering, "Reducing T-count with the ZX-calculus," *Phys. Rev. A*, 2020.
[30] D. M. Greenberger *et al.*, *Going beyond Bell's theorem*, 2007. arXiv: 0712.0921.
[31] G. Vidal and C. M. Dawson, "Universal quantum circuit for two-qubit transformations with three controlled-NOT gates," *Phys. Rev. A*, 2004.
[32] A. Barenco *et al.*, "Elementary gates for quantum computation," *Phys. Rev. A*, 1995.
[33] D. Maslov, "On the advantages of using relative phase Toffolis with an application to multiple control Toffoli optimization," *Phys. Rev. A*, 2016.
[34] R. Wille *et al.*, "Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations," in *Design Automation Conf.*, 2019.
[35] P. Murali *et al.*, "Noise-adaptive compiler mappings for Noisy Intermediate-Scale Quantum computers," in *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2019.
[36] G. Li *et al.*, "Tackling the qubit mapping problem for NISQ-era quantum devices," in *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2019.
[37] R. Wille *et al.*, "Visualizing decision diagrams for quantum computing," in *Design, Automation and Test in Europe*, 2021.
[38] L. Burgholzer *et al.*, "Verifying results of the IBM Qiskit quantum circuit compilation flow," in *Int'l Conf. on Quantum Computing and Engineering*, 2020.
[39] B. Coecke and A. Kissinger, "Picturing quantum processes," in *Diagrammatic Representation and Inference*, 2018.
[40] M. Backens *et al.*, "There and back again: A circuit extraction tale," *Quantum*, 2021.
[41] M. Backens, *The ZX-calculus is complete for stabilizer quantum mechanics*, 2013. arXiv: 1307.7025.
[42] R. Vilmart, *A near-optimal axiomatisation of ZX-calculus for pure qubit quantum mechanics*, 2018. arXiv: 1812.09114.
[43] R. Wille *et al.*, "JKQ: JKU tools for quantum computing," in *Int'l Conf. on CAD*, 2020.
[44] A. Kissinger and J. van de Wetering, "PyZX: Large scale automated diagrammatic reasoning," presented at the Quantum Physics and Logic, 2019.
[45] L. Burgholzer *et al.*, "Random stimuli generation for the verification of quantum circuits," in *Asia and South Pacific Design Automation Conf.*, 2021.
[46] R. Wille *et al.*, "RevLib: An online resource for reversible functions and reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, 2008.
[47] A. W. Cross *et al.*, *OpenQASM 3: A broader and deeper quantum assembly language*, 2021. arXiv: 2104.14722.