Improved DD-based Equivalence Checking of Quantum Circuits

Lukas Burgholzer Robert Wille
Institute for Integrated Circuits, Johannes Kepler University, 4040 Linz, Austria
lukas.burgholzer@jku.at robert.wille@jku.at
iic.jku.at/eda/research/quantum/

Abstract—Quantum computing is gaining considerable momentum through the recent progress in physical realizations of quantum computers. This led to rather sophisticated design flows in which the originally specified quantum functionality is compiled through different abstractions. This increasingly raises the question whether the respectively resulting quantum circuits indeed realize the originally intended function. Accordingly, efficient methods for equivalence checking are gaining importance. However, existing solutions still suffer from significant shortcomings such as their exponential worst case performance and an increased effort to obtain counterexamples in case of non-equivalence. In this work, we propose an improved DD-based equivalence checking approach which addresses these shortcomings. To this end, we utilize decision diagrams and exploit the fact that quantum operations are inherently reversible – allowing for dedicated strategies that keep the overhead moderate in many cases. Experimental results confirm that the proposed strategies lead to substantial speed-ups – allowing to perform equivalence checking of quantum circuits factors or even magnitudes faster than the state of the art.

I. Introduction

Quantum computing promises to significantly outperform classical computing in certain tasks, e.g., integer factorization [1], database search [2], quantum chemistry [3] and many more [4]–[6]. By utilizing quantum mechanical phenomena like superposition and entanglement of so-called quantum bits (qubits) [7], up to exponential speed-ups can be achieved. Recent advancements in the physical realization of quantum computers and the involvement of big players like Google, IBM or Intel [8]–[10] further increase the momentum behind quantum computing.

This led to rather sophisticated design flows in which the originally intended quantum functionality is realized and executed on an actual physical device. More precisely, a high-level description of the desired functionality (usually provided in terms of a *quantum algorithm* [11]–[13]) is first compiled to a sequence of low-level quantum operations (usually in terms of a *quantum circuit* composed of so-called *quantum gates*). Then, the gates of the resulting circuit often have to be further decomposed into elementary gates which are supported by the targeted architecture [14]–[16]. Finally, physical constraints on the allowed operations induced by the targeted architecture have to be accounted for in another mapping step [17]–[19]. In between, further optimizations can be applied to improve the costs or certain aspects of the final circuit to be executed [20]–[22].

However, while several methods for all these tasks are available in the meantime and some of them even have been integrated into comprehensive toolkits such as IBM's Qiskit [23] or Microsoft's QDK [24], this process eventually yields several abstractions of quantum algorithms and/or quantum circuits which significantly differ in their basis operations and structure but are still supposed to be functionally equivalent. Consequently, checking whether the original functionality is indeed maintained throughout all these different abstractions, becomes increasingly relevant in order to guarantee a consistent and error-free design flow. Although first approaches for this purpose (addressing *equivalence checking of quantum circuits*) have been proposed in the past, they often suffer from not being complete, from providing insufficient support of key features such as entanglement, or from the exponential worst case complexity as well as the non-trivial process of obtaining counterexamples in the case of non-equivalence (the state of the art is discussed in more detail later in Section III).

In this work, we propose an advanced approach for equivalence checking which addresses these shortcomings. To this end, we utilize *Decision Diagrams* (DDs) which already have been proven to provide a suitable means for representing and manipulating quantum circuits [25]–[28]. Moreover, we additionally exploit the fact that every quantum computation is inherently reversible. This allows to treat the two quantum circuits to be checked for equivalence in a fashion that keeps the corresponding representation close to the identity function. Since the identity function can be represented in terms of DDs with linear complexity, this directly tackles the exponential worst case complexity of current state-of-the-art solutions. Besides that, counterexamples in case of non-equivalence can be determined "for free" following the proposed idea. Experimental results show the potential of the proposed idea and confirm that this allows to perform equivalence checking of quantum circuits factors or even magnitudes faster than the state of the art.

The remainder of this paper is structured as follows: Section II covers the background on quantum circuits and decision diagrams. Section III reviews the state of the art in equivalence checking of quantum circuits followed by a description of the proposed idea. Based on that, several possible strategies implementing the proposed idea are provided in Section IV. Finally, Section V summarizes the obtained experimental results before the paper is concluded in Section VI.

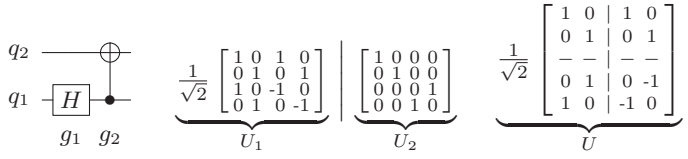
II. Background

In this section, we briefly review quantum circuits as well as decision diagrams. Note that the descriptions are kept brief. For a more detailed treatment, we refer to [7] and [27], respectively.

A. Quantum Circuits

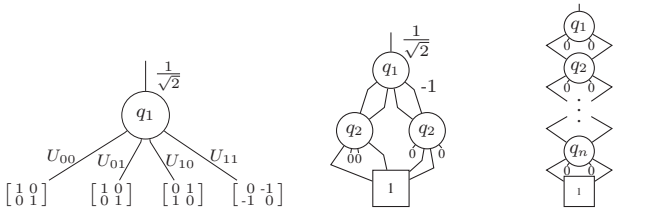
Quantum computing uses *qubits* instead of bits to describe the state of a quantum system. In addition to the basis states

978-1-7281-4123-7/20/$31.00



(a) $G = g_1 g_2$ (b) $U_1 \equiv g_1$ and $U_2 \equiv g_2$ (c) $U = U_2 U_1$

Figure 1: Quantum circuits and matrix representation



(a) First level (b) Complete DD (c) Identity DD

Figure 2: Decision diagrams

$|0\rangle$ and $|1\rangle$, qubits may also assume an arbitrary superposition $\alpha_0 |0\rangle + \alpha_1 |1\rangle$ with $\alpha_0, \alpha_1 \in \mathbb{C}$ and $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Accordingly, the state of an n-qubit system is described as $|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$ with $\sum_{x \in \{0,1\}^n} |\alpha_x|^2 = 1$, which is represented by a 2^n-dimensional *state vector* $[\alpha_0, \ldots, \alpha_{2^n-1}]^\top$. Modifying the state of a quantum systems can be conducted by applying *quantum operations*, which are represented by unitary[1] matrices $U \in \mathbb{C}^{2^n \times 2^n}$.

A sequence of quantum operations is usually described in terms of a *quantum circuit* G, which is comprised of a sequence of *quantum gates*, i.e., $G = g_1 \ldots g_m$. Each gate g_i represents a unitary matrix describing its functionality, i.e., $g_i \equiv U_i \in \mathbb{C}^{2^n \times 2^n}$. To obtain the unitary matrix describing the entire circuit, the individual gate matrices have to be multiplied with each other in reverse order, i.e., the functionality U of a circuit $G = g_1 \ldots g_m$ is obtained by $U = U_m \cdots U_1$ (where each U_i describes the functionality of a gate g_i).

Example 1. *Fig. 1a shows a quantum circuit $G = g_1 g_2$ comprised of $n = 2$ qubits and $m = 2$ gates. The functionality of the two gates g_1 and g_2 is defined by the unitary matrices U_1 and U_2, respectively, which are provided in Fig. 1b. The unitary matrix U describing the entire circuit G is obtained by $U = U_2 U_1$ – leading to a matrix as shown in Fig. 1c.*

Since quantum operations are inherently reversible, the inverse of a quantum circuit G can easily be determined by $G^{-1} = (g_1 \ldots g_m)^{-1} = g_m^{-1} \ldots g_1^{-1} \equiv U_1^{-1} \cdots U_m^{-1}$.

B. Decision Diagrams

Decision diagrams (DDs) have been proposed as a compact way of representing quantum functionality. This is accomplished by decomposing a given unitary matrix $U \in \mathbb{C}^{2^n \times 2^n}$ into equally sized sub-matrices $U_{ij} \in \mathbb{C}^{2^{n-1} \times 2^{n-1}}$, i.e., $U = \left[\begin{smallmatrix} U_{00} & U_{01} \\ U_{10} & U_{11} \end{smallmatrix}\right]$. This is recursively conducted until single complex numbers remain. Then, each decomposition step constitutes a level in the decision diagram and each (sub-)matrix is represented by a node with four successors for each of the corresponding sub-matrices[2]. A compact representation is eventually obtained when identical sub-matrices occur which are represented by the same node. By additionally introducing edge weights, sub-matrices that only differ by a scalar factor can also be shared – allowing for an even more compact representation.

Example 2. *Consider again the the matrix U as shown in Fig. 1c. Decomposing this matrix yields, in the first level, four sub-matrices as indicated in Fig. 2a. Since sub-matrices U_{00} and U_{01} are equal and sub-matrices U_{10} and U_{11} only differ by the scalar factor -1, the representation of these sub-matrices can be shared if corresponding edge weights are employed. This eventually leads to the decision diagram shown in Fig. 2b representing the entire matrix U.*

Using decision diagrams, multiplication of matrices is handled by recursively breaking the operation down into multiplications/additions of sub-matrices. This eventually allows to construct a decision diagram representing the functionality of a quantum circuit G by multiplying the DD-representations of its gates g_i (more precisely, the DD-representations of the respective matrices U_i of the gates g_i). This leads to *intermediate decision diagrams* after each multiplication until the entire functionality U of the circuit G is obtained. Keeping the average (or maximum) number of nodes required by the decision diagrams low is essential for a compact and efficient performance on quantum circuits. In this regard, the identity function

$$\mathbb{I} = \left(\begin{array}{cc|cc} 1 & & 0 & \\ & \ddots & & \\ 0 & & 1 & \\ \hline & 0 & & \begin{matrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{matrix} \end{array}\right)$$

constitutes the best case, because here *all* sub-matrices U_{00} and U_{11} are recursively identical and *all* sub-matrices U_{01} and U_{10} are solely composed of 0-entries – yielding the compact decision diagram as shown in Fig. 2c.

III. Equivalence Checking of Quantum Circuits

The problem of equivalence checking addresses the question whether two given circuits G and G' do realize the same function. Complete methods for equivalence checking either provide a corresponding proof for that or, in case both circuits G and G' are not equivalent, generate a counterexample showcasing the non-equivalence.

In this section, we briefly review and discuss existing methods for equivalence checking of quantum circuits. Based on that, the general idea of the proposed improved equivalence checking method is provided.

A. State of the Art

Equivalence checking of quantum circuits has intensely been considered in the past. Inspired by methods for equivalence checking of classical circuits, methods based on simulation [28], [29], re-writing [30], Boolean satisfiability [31], or decision diagrams [26]–[29], [32] have been proposed. However, approaches based on simulation and re-writing either suffer from the fact that they are not complete and/or may require the enumeration of an exponential number of stimuli or possibilities to check. Solvers for Boolean satisfiability may

[1] A matrix U is unitary iff $UU^\dagger = U^\dagger U = \mathbb{I}$, where $U^\dagger$ denotes the Hermitian conjugate of U. As a result, $U^{-1} = U^\dagger$.

[2] Note that sub-matrices only containing zero entries are represented as 0-stubs.

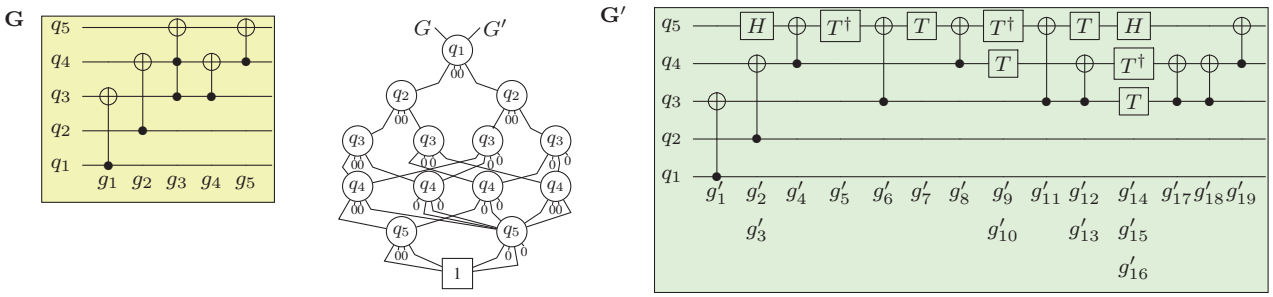


Figure 3: Two equivalent circuits G and G' with their decision diagram representation

cope with these problems but, in case of quantum circuits, face the problem that an infinite number of quantum states need to be encoded.[3] Because of that, approaches based on decision diagrams (also called *DD-based equivalence checking*) still constitute the state of the art for equivalence checking of quantum circuits.

Indeed, decision diagrams offer several benefits for this task: As reviewed in Section II-B, they provide a compact representation of the respective functionality realized by a quantum circuit. Moreover, most decision diagrams provide a canonical way of representing quantum functionality (usually with respect to a given variable order). Because of that, once the functionalities of different quantum circuits are represented as decision diagrams, checking their equivalence can be conducted in constant time by simply comparing the pointers to their root nodes.

Example 3. *Consider the two quantum circuits G and G' as shown at the left-hand side and the right-hand side of Fig. 3. Creating decision diagrams for both circuits yields the representation as shown in the center of Fig. 3. As can be seen, the pointers to the root node(s) are equivalent for both circuits – proving that both, G and G', are functionally equivalent. During the construction, intermediate decision diagrams required a maximum of* 15 *nodes, while the average node count was* 12.2*.*

However, despite the benefits discussed above, DD-based equivalence checking of quantum circuits as conducted thus far still has significant shortcomings. In fact, representing the entire functionality of a quantum circuit still might be exponential in the worst case. Even if the representation of the overall functionality of a circuit might be compact, intermediate results may require significantly more space. As an example, the final decision diagram discussed in Example 3 and shown in Fig. 3 is composed of 13 nodes; however, intermediate decision diagrams generated during the construction required up to 15 nodes.[4]

Moreover, the generation of a counterexample (which is supposed to be provided in case two quantum circuits are not equivalent) require further (potentially) expensive manipulations of the decision diagrams. In fact, in order to generate such a counterexample, the "difference" between the two non-equivalent decision diagrams has to be determined. To this end, one decision diagram has to be inverted (i.e., the conjugated-transposed representation has to be generated) and multiplied with the other decision diagram – requiring non-trivial operations on (potentially) large representations.

B. General Idea

In this work, we propose an alternative and improved method for DD-based equivalence checking which addresses the problems of the current state of the art and, as confirmed by evaluations summarized in Section V, allows to substantially improve the process. To this end, we exploit the fact that every quantum operation U is inherently reversible. In the case that both considered circuits G and G' are functionally equivalent, this allows for the conclusion that $G \cdot G'^{-1} = \mathbb{I}$, i.e., that multiplying G with the inverse of G' yields the identity function $\mathbb{I}$. Since the identity function constitutes the best case for decision diagrams (as discussed in Section II-B and Fig. 2c), this offers significant potential.

Unfortunately, creating $G \cdot G'^{-1}$ in a naive fashion, i.e., by applying

$$\begin{aligned} G \cdot G'^{-1} &= (g_1 \dots g_m) \cdot (g'^{-1}_{m'} \dots g'^{-1}_1) \\ &\equiv (U_m \cdots U_1) \cdot (U'^{-1}_1 \cdots U'^{-1}_{m'}) \end{aligned}$$

hardly yields any benefits compared to the state of the art. This is, because even if the final decision diagram would represent the (very compact) identity function, (potentially large) decision diagrams representing G and G'^{-1} would still be generated as intermediate decision diagrams for the first m and last m' gates, respectively. Instead, the full potential of $G \cdot G'^{-1} = \mathbb{I}$ is utilized if the associativity of the respective multiplications is fully exploited. More precisely, if the respective gates of G and G'^{-1} are multiplied in a fashion so that their products frequently yield the identity, the entire equivalence checking process can be conducted with rather small (intermediate) decision diagrams only. This is illustrated by the following example.

Example 4. *Consider again the two circuits G and G' from Fig. 3 as discussed before in Example 3. Conducting the multiplications as sketched in Fig. 4 frequently yields to situations where the impact of a gate of circuit G (potentially increasing the size of the decision diagram) is reverted by multiplications with gates from G'^{-1} (potentially decreasing the size of the decision diagram back to the representation*

[3]Note that this problem has partially been addressed in [31] by employing a detailed structural analysis that restricts the number of possible states to a finite one. However, the resulting encoding does not support key features of quantum circuits such as entanglement and, hence, is not applicable for almost all relevant quantum circuits.

[4]Note that an increase by two nodes might not seem substantial. However, for realistic quantum circuits which obviously represent much more complex functionality than this example, the difference in the number of nodes between the final decision diagram and intermediate decision diagrams easily sums up to several orders of magnitudes.

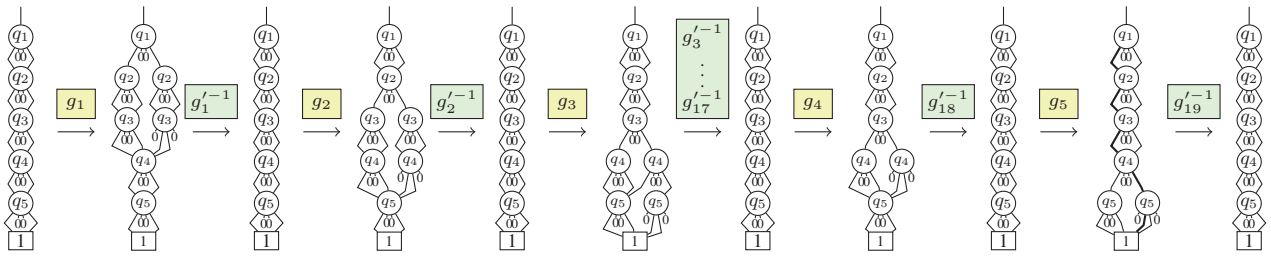


Figure 4: Illustration of the general idea: $G \cdot G'^{-1} = \mathbb{I}$ with G and G' from Fig. 3 using decision diagrams

of the identity function). By this, instead of 15 nodes (as discussed in Example 3), never more than 9 *nodes are required. Moreover, the average node count of intermediate decision diagrams drops from* 12.2 *to* 6.4*.*

In general, starting from a decision diagram representing the identity function, gates from G and G'^{-1} are applied successively either from the left (G) or from the right (G'^{-1}), i.e.

$$G \rightarrow \mathbb{I} \leftarrow G'^{-1}.$$

However, determining when to apply gates from G and when to apply gates from G'^{-1} is not always obvious. But whenever a "good" strategy for a selection of gates can be employed, equivalence checking of two equivalent quantum circuits can be conducted very efficiently and compactly using decision diagrams (corresponding strategies for this purpose will be presented and evaluated in Section IV and Section V, respectively).

Moreover, even if the considered circuits G and G' are *not* functionally equivalent, the observations from above still promise improvements compared to creating the complete decision diagrams for G and G'. In fact, the resulting decision diagram for $G \cdot G'^{-1}$ likely will be substantially smaller than the representation of the full functionalities of G and G'. Moreover, $G \cdot G'^{-1}$ inherently provides an efficient representation of all counterexamples "for free" (while state-of-the-art solutions have to explicitly create those using additional inversion and multiplication operations as discussed in Section III-A).

Example 5. *Consider again the circuits G and G' from Fig. 3 and assume that, e.g., due to a design error, gate g_{19}' is dropped. Applying the state-of-the-art approach for equivalence checking (taken from [27]) would yield two decision diagrams of size* 13 *and detect that they are not equivalent. Then, in order to generate counterexamples, the difference of both circuits has to be determined – requiring the inversion of G' and multiplication with G. In contrast, applying the general idea proposed above follows the same steps as illustrated in Fig. 4 up to the very last multiplication – which is dropped. As already discussed in Example 4, this yields decision diagrams not larger than* 9 *nodes and additionally provides the difference of both circuits as a result. From this, we can extract, e.g., the counterexample $|\phi\rangle = |00010\rangle$ (indicated by bold lines in the decision diagram before the last step in Fig. 4). Indeed, it holds that $G|\phi\rangle = |00011\rangle$, while $G'|\phi\rangle = |00010\rangle$.*

IV. Strategies for Advanced Equivalence Checking

Following the general ideas outlined above potentially allows to conduct *DD-based* equivalence checking of quantum circuits in a significantly more efficient fashion than before. However, to fully exploit that, a "good" strategy how to eventually conduct $G \rightarrow \mathbb{I} \leftarrow G'^{-1}$ (i.e., when to apply a gate from G and when to apply a gate from G'^{-1}) is essential. In this section, we propose several promising strategies and illustrate their application. The effect of those strategies and, by this, the efficiency of the proposed improved DD-based equivalence checking is afterwards evaluated in Section V.

A. Naive Strategy

The first strategy is motivated by the (rather naive) assumption that a given circuit G is checked against itself, i.e. $G \rightarrow \mathbb{I} \leftarrow G^{-1}$. Then, obviously the best possible strategy is to alternate between applications of gates from G and G^{-1} – yielding the identity function after each pair of operations. In case that $G \neq G'$ (and, w.l.o.g., assuming that $m < m'$, i.e., that G' has more gates), this strategy alternates between the gates from G and the gates from G'^{-1} until all gates of G have been applied. Afterwards, the remaining "left-over gates" from G'^{-1} are applied. This strategy supposedly works well if G and G'^{-1} are very similar, but obviously looses its benefits if both circuits significantly differ in their structure (in particular if one circuit has significantly more gates than the other, i.e., if $m \ll m'$).

Example 6. *Consider again the two circuits G and G' from Fig. 3 as discussed before in Example 3. Applying the naive strategy leads to an order of gate applications as shown in the first row of Fig. 5. During this process, the size of the (intermediate) decision diagrams never exceeds* 9 *nodes, while the average node count is* 7.1*.*

B. Proportional Strategy

Applying the naive strategy to circuits which, structurally, are significantly different obviously leads to an unbalance since a huge portion of "left-over" gates are solely applied – possibly neglecting the effect of staying close to the identity function. In order to avoid that, the proportional strategy aims for a balanced approach. To this end, first, the ratio with respect to the number of gates for both circuits is determined. Afterwards, the gates from G and the gates from G'^{-1} are proportionally applied according to this ratio.

Example 7. *Consider again the two circuits G and G' from Fig. 3 as discussed before in Example 3. The ratio between their gate counts is* $5 : 19 \approx 1 : 4$*. Hence, applying the*

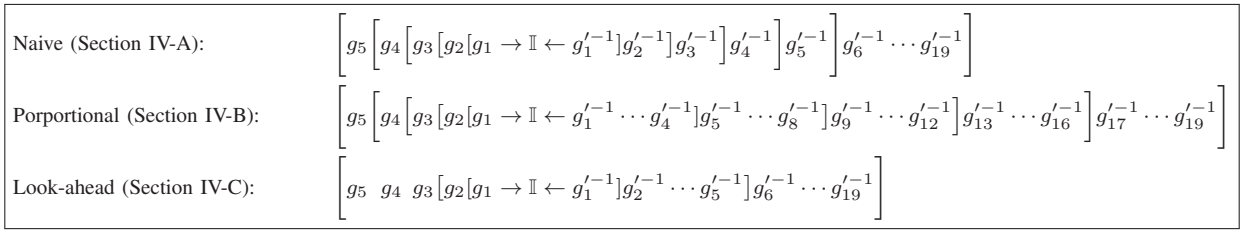


Figure 5: Illustrations of the proposed strategies (for the circuits G and G'^{-1} from Fig. 3)

proportional strategy leads to an order of gate applications as shown in the second row of Fig. 5. During this process, the size of the (intermediate) decision diagrams never exceeds 9 *nodes, while the average node count is* 6.5*.*

C. Look-ahead Strategy

Despite strategies that motivate themselves through the structure of the given circuit, also schemes based on the actual size of the (intermediate) DDs may provide a good indication of how to proceed. Recall that the general aim is to stay as close as possible to the identity function (leading to the smallest possible DD). Hence, the decision to either apply a gate from G or a gate from G'^{-1} can be based on which case actually leads to a smaller DD. This is conducted by the look-ahead scheme. While this potentially doubles the number of multiplications to be performed (since both alternatives have to be checked out), it may lead to smaller decision diagrams and, by this, a more efficient equivalence check.

Example 8. *Consider again the two circuits G and G' from Fig. 3 as discussed before in Example 3. Applying the look-ahead strategy leads to an order of gate applications as shown in the third row of Fig. 5. During this process, the size of the (intermediate) decision diagrams never exceeds* 9 *nodes, while the average node count is* 7.0*.*

Even for the small example showcased throughout this section, all proposed strategies perform significantly better in terms of maximum as well as average DD size when compared to the state-of-the-art approach.

V. Experimental Results

The proposed idea for improved DD-based equivalence checking of quantum circuits together with the strategies introduced in the previous section have been implemented in C++ on top of the DD-package provided in [33]. To evaluate the potential of the resulting solution, circuits taken from [34], [35] as well as own implementations of quantum algorithms (such as Shor's algorithm, Grover's Search, Deutsch Algorithm, etc.) have been used as benchmarks. For each given high-level circuit description G, a corresponding low-level circuit description G' has been generated – forming proper equivalence checking instances. Furthermore, in order to also evaluate the performance on non-equivalent circuits, errors have been randomly injected into the circuits. All evaluations have been conducted on a 4 GHz machine with 32 GiB of main memory running Ubuntu 16.04 using a hard timeout of one hour (3600 s). Afterwards, we compared the respectively obtained results to the state-of-the-art equivalence checking approach (taken from [27]).

Table I provides a selection of the respectively obtained results[5]. The first four columns list the name of the benchmark, the number n of qubits, as well as the numbers of gates of both circuits G and G'. The remaining columns show the peak node count (*Nodes*) reported by the DD-package and the runtime t (in CPU seconds) for the state-of-the-art approach as well as the corresponding strategies. The table rows are separated into two groups – showing the results obtained for the equivalent and for the non-equivalent circuits.

As expected, the naive strategy performs rather poorly – particular in cases where one circuit has significantly more gates than the other. In contrast, the proportional and the look-ahead strategy rather consistently allow for much smaller intermediate decision diagrams and, hence, for significantly better performance. This is perfectly in line with the expectations discussed in Section IV. However, most importantly, *all* strategies (even the naive one) show substantially better performance than the state-of-the-art approach. In the majority of cases, equivalence checking can be conducted factors or even magnitudes faster. Moreover, in many cases, the proposed solution even allows to successfully complete the check in seconds or just a few minutes, while the state-of-the-art solution ran into a timeout of one hour.

VI. Conclusions

In this paper, we proposed an improved DD-based approach for equivalence checking of quantum circuits. To this end, we exploited the fact that quantum circuits are inherently reversible – allowing for strategies which, together with decision diagrams, keep the overhead of the problem moderate in many cases. Experimental results have shown that this leads to speed-ups of several factors or even magnitudes. By this, we were able to show that – in contrast to conventional circuitry – quantum circuits can be verified rather efficiently whenever a "good" strategy for exploiting the underlying reversibility is available. This finding also provides interesting connections for future work on quantum circuit verification. An implementation of the proposed scheme is publicly available at iic.jku.at/eda/research/quantum_verification/.

Acknowledgements

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

[5]Due to page limitations, not all results could have been listed. However, all results as well as the implementation of the proposed approach are available at iic.jku.at/eda/research/quantum_verification/.

Table I: Experimental results

				Ref. (Section III)		Proposed Strategies (Section IV)					
						Naive		Proportional		Look-ahead	
Benchmark	n	$\|G\|$	$\|G'\|$	*Nodes*	t [s]	*Nodes*	t [s]	*Nodes*	t [s]	*Nodes*	t [s]
Equivalence											
$shor\_33\_2$	26	189979	449131	–	> 3600	–	> 3600	–	> 3600	65008	**322.83**
qft_16	16	136	616	–	> 3600	–	> 3600	15006	**0.07**	12276	**0.02**
$hwb6\_301$	46	159	1377	–	> 3600	–	> 3600	40107	**0.36**	70426	**3.81**
bw_291	87	307	2533	–	> 3600	–	> 3600	403396	**19.27**	169994	**22.48**
$grover\_6$	9	1042	1622	–	> 3600	55135	**14.16**	35201	**0.52**	18287	**0.03**
$inst4/4/100/0/10$	16	95	459	–	> 3600	30015	**0.17**	20292	**0.03**	25034	**0.05**
$urf6\_160$	15	10740	161100	1495110	810.92	1646515	1017.36	25001	**1.70**	40077	**1.95**
$ham15\_298$	45	153	699	3506681	439.07	1998361	**288.27**	40370	**0.33**	91857	**129.16**
$rd84\_253$	12	111	121645	440074	169.30	475024	**123.95**	295008	**63.52**	270085	**36.92**
$urf1\_149$	9	11554	173310	485035	69.34	500052	69.60	12892	**1.53**	25375	**1.39**
$5xp1\_194$	17	85	48485	360122	33.33	180080	**10.52**	240009	**16.31**	19005	**0.45**
$hwb5\_300$	28	88	746	339390	10.90	235935	**4.58**	18415	**0.07**	50335	**3.48**
$rd84\_313$	34	104	804	263763	9.68	130198	**2.73**	25075	**0.15**	20001	**0.11**
$mod5adder\_306$	32	96	782	288862	8.44	175256	**4.79**	15533	**0.06**	458317	17.47
$hwb7\_61$	7	236	28820	115109	4.12	115128	**3.94**	100002	**2.69**	115001	**3.60**
$grover\_5$	9	830	1262	25000	0.07	30009	0.13	30020	0.13	21236	**0.03**
$deutsch - 64bal$	64	190	314	27362	0.06	25061	**0.05**	25094	**0.05**	25164	0.07
Non-Equivalence											
$shor\_33\_2$	26	189979	444647	–	> 3600	–	> 3600	–	> 3600	35036	**6.27**
$urf1\_149$	9	11554	169785	–	> 3600	–	> 3600	–	> 3600	35019	**1.78**
qft_16	16	136	612	–	> 3600	–	> 3600	142017	**41.92**	–	> 3600
$hwb6\_301$	46	159	1357	–	> 3600	–	> 3600	1299050	**105.48**	79235	**2.61**
$inst4/4/100/0/10$	16	95	456	–	> 3600	30030	**0.19**	20714	**0.02**	23049	**0.04**
$ham15\_298$	45	153	693	3518246	584.18	2047513	**403.42**	70110	**0.89**	30852	**0.42**
$hwb5\_300$	28	88	740	311314	12.41	240930	**10.02**	51982	**0.53**	57836	**0.77**
$rd84\_313$	34	104	795	269111	10.86	231531	**10.22**	100522	**1.65**	95224	**5.96**
$mod5adder\_306$	32	96	774	275817	9.47	255232	11.33	45551	**0.45**	136907	26.01
$grover\_5$	9	830	1256	32268	0.17	35000	0.22	35026	0.34	45114	28.94
$deutsch - 64bal$	64	190	311	27836	0.06	30172	0.08	25095	**0.06**	25196	0.08

n: Number of qubits $|G|$: Gate count of G $|G'|$: Gate count of G' *Nodes*: Peak node count t: Runtime in seconds

REFERENCES

[1] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, 1997.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. of the ACM*, 1996, pp. 212–219.

[3] P. J. J. O'Malley et al., "Scalable Quantum Simulation of Molecular Energies," *Phys. Rev. X*, vol. 6, no. 3, p. 031007, 2016.

[4] P. J. Coles et al., "Quantum Algorithm Implementations for Beginners," *arXiv:1804.03719 [quant-ph]*, 2018.

[5] A. Montanaro, "Quantum algorithms: An overview," *npj Quantum Inf*, vol. 2, no. 1, p. 15023, 2016.

[6] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[7] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[8] J. Kelly, "A Preview of Bristlecone, Google's New Quantum Processor," ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html, 2018.

[9] J. Hsu, "CES 2018: Intel's 49-Qubit Chip Shoots for Quantum Supremacy," spectrum.ieee.org/tech-talk/computing/hardware/intels-49qubit-chip-aims-for-quantum-supremacy, 2018.

[10] IBM Q team, "IBM Q," research.ibm.com/ibm-q/.

[11] K. M. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, and M. Roetteler, "Q#: Enabling scalable quantum computing and development with a high-level domain-specific language," *Proc. of the RWDSL*, 2018.

[12] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron, "Quipper: A Scalable Quantum Programming Language," *SIGPLAN Not.*, vol. 48, no. 6, p. 333, 2013.

[13] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open Quantum Assembly Language," *arXiv:1707.03429 [quant-ph]*, 2017.

[14] R. Wille, M. Soeken, C. Otterstedt, and R. Drechsler, "Improving the mapping of reversible circuits to quantum circuits using multiple target lines," in *Asia and South Pacific Design Automation Conf.*, 2013, pp. 145–150.

[15] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 32, no. 6, pp. 818–830, 2013.

[16] D. M. Miller, R. Wille, and Z. Sasanian, "Elementary Quantum Gate Realizations for Multiple-Control Toffoli Gates," in *Int'l Symp. on Multi-Valued Logic*, 2011, pp. 288–293.

[17] M. Saeedi, R. Wille, and R. Drechsler, "Synthesis of Quantum Circuits for Linear Nearest Neighbor Architectures," *Quantum Inf Process*, vol. 10, no. 3, pp. 355–377, 2011.

[18] G. Li, Y. Ding, and Y. Xie, "Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices," in *ASPLOS*, 2019, pp. 1001–1014.

[19] A. Zulehner, A. Paler, and R. Wille, "An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2019.

[20] Z. Sasanian and D. M. Miller, "Reversible and Quantum Circuit Optimization: A Functional Approach," in *Int'l Conf. of Reversible Computation*, R. Glück and T. Yokoyama, Eds., 2013, pp. 112–124.

[21] 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, pp. 228–243.

[22] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, and D. Maslov, "Automated optimization of large quantum circuits with continuous parameters," *npj Quantum Inf*, vol. 4, no. 1, p. 23, 2018.

[23] IBM Q team, "Qiskit: An Open-source Framework for Quantum Computing," 2019.

[24] "Microsoft QDK," microsoft.com/en-us/quantum/development-kit.

[25] G. F. Viamontes, M. Rajagopalan, I. L. Markov, and J. P. Hayes, "Gate-level simulation of quantum circuits," in *Asia and South Pacific Design Automation Conf.*, 2003.

[26] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, and S.-Y. Kuo, "An XQDD-Based Verification Method for Quantum Circuits," in *IEICE Trans. Fundamentals*, 2008, pp. 584–594.

[27] 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 Integrated Circuits and Systems*, vol. 35, no. 1, pp. 86–99, 2016.

[28] A. Zulehner and R. Wille, "Advanced Simulation of Quantum Computations," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 848–859, 2019.

[29] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Checking equivalence of quantum circuits and states," in *Int'l Conf. on CAD*, 2007, pp. 69–74.

[30] S. Yamashita and I. L. Markov, "Fast equivalence-checking for quantum circuits," in *Int'l Symp. on Nanoscale Architectures*, 2010, pp. 23–28.

[31] R. Wille, N. Przigoda, and R. Drechsler, "A compact and efficient SAT encoding for quantum circuits," in *IEEE AFRICON*, 2013.

[32] P. Niemann, R. Wille, and R. Drechsler, "Equivalence Checking in Multi-level Quantum Systems," in *Int'l Conf. of Reversible Computation*, 2014, pp. 201–215.

[33] 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.

[34] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: An Online Resource for Reversible Functions and Reversible Circuits," in *Int'l Symp. on Multi-Valued Logic*, 2008, pp. 220–225.

[35] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, "Characterizing Quantum Supremacy in Near-Term Devices," *Nature Phys*, vol. 14, no. 6, pp. 595–600, 2018.