Checking Equivalence of Quantum Circuits and States

George F. Viamontes,[*] Igor L. Markov,[§] and John P. Hayes[§]

[*]Lockheed Martin ATL
3 Executive Campus
Cherry Hill, NJ 08002
gviamont@atl.lmco.com

[§]University of Michigan
Advanced Computer Architecture Lab.
2260 Hayward St.
Ann Arbor, MI 48109-2121
{imarkov, jhayes}@eecs.umich.edu

Abstract

Among the post-CMOS technologies currently under investigation, quantum computing (QC) holds a special place. QC offers not only extremely small size and low power, but also exponential speed-ups for important simulation and optimization problems. It also poses new CAD problems that are similar to, but more challenging, than the related problems in classical (non-quantum) CAD, such as determining if two states or circuits are functionally equivalent. While differences in classical states are easy to detect, quantum states, which are represented by complex-valued vectors, exhibit subtle differences leading to several notions of equivalence. This provides flexibility in optimizing quantum circuits, but leads to difficult new equivalence-checking issues for simulation and synthesis. We identify several different equivalence-checking problems and present algorithms for practical benchmarks, including quantum communication and search circuits, which are shown to be very fast and robust for hundreds of qubits.

1 Introduction

Quantum computing (QC) is a recently discovered alternative to conventional computer technology that offers not only miniaturization, but massive performance speed-ups for certain tasks [11, 17, 10] and new levels of protection in secure communications [4, 5]. Practical implementations of these devices are quickly becoming a reality. This year, D-Wave Systems unveiled a 16-qubit quantum computer which can be fabricated using existing semiconductor technology [1]. D-Wave plans to release a 512- and a 1024-qubit computer in 2008. Large quantum computing systems may therefore require deeper consideration by the CAD community sooner than previously expected.

In a quantum computer, information is stored in particle states and processed using quantum-mechanical operations referred to as quantum gates. The analogue of the classical bit, the qubit, has two basic states denoted $|0\rangle$ and $|1\rangle$, but can also exist in a superposition of these states $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $|\alpha|^2 + |\beta|^2 = 1$. A composite system consisting of n such qubits requires 2^n parameters (amplitudes) α_j, which are complex numbers that can be indexed by n-bit binary numbers thus: $|\Phi\rangle = \Sigma_{j=1}^{2^n} \alpha_j |j\rangle$, where $\Sigma|\alpha_j|^2 = 1$. Like any complex number, each α_j can be written in the form $e^{i\theta_j}|\alpha_j|$, where $e^{i\theta_j}$ is the *phase* and $|\alpha_j|$ is the *magnitude* of α_j. Both θ_j and $|\alpha_j|$ are real numbers, and $i = \sqrt{-1}$. Quantum gates transform qubits by applying unitary matrices to them. Measurement of a quantum state produces classical bits with probabilities dependent on the magnitudes $|\alpha_i|$. Combining several gates, as in Figure 1, yields *quantum circuits* [13] which compactly describe more sophisticated transformations that implement quantum algorithms.

Based on the success of CAD for classical logic circuits, new algorithms have been proposed for synthesis and simulation of quantum circuits [3, 15, 18, 9, 21, 23]. In particular, Maslov et al. [12] describe what amounts to placement and physical synthesis for quantum circuits — "adapting the circuit to particulars of the physical environment which restricts/complicates the establishment of certain direct interactions between qubits." Traditionally, such transformations must be verified by equivalence-checking, but the quantum context is more difficult because qubits and quantum gates may differ by global and relative phase (defined below), yet be equivalent upon measurement [13]. To this end, our work is the first to develop techniques for quantum phase-equivalence checking.

Consider two quantum states $|\psi\rangle = \Sigma_j \alpha_j |j\rangle$ and $|\varphi\rangle = \Sigma_j \beta_j |j\rangle$. They are obviously *equivalent* if $\alpha_j = \beta_j$ for all j. If $|\alpha_j| = |\beta_j|$ for all j but some α_j and β_j differ in phase, then $|\psi\rangle$ and $|\varphi\rangle$ are said to be *equivalent up to relative phase*. This implies that for each j there is a phase angle θ_j such that $\alpha_j = e^{i\theta_j}\beta_j$. In the special case where all the θ_j's have the same value θ, we can write $|\varphi\rangle = e^{i\theta}|\psi\rangle$, and the two states are said to be *equivalent up to global phase*. When the states are only equivalent up to relative phase, a unitary diagonal matrix can be used to transform one state into the other thus:

$$|\varphi\rangle = \mathrm{diag}(e^{i\theta_0}, e^{i\theta_1}, \ldots, e^{i\theta_{N-1}})\,|\psi\rangle\,. \qquad (1)$$

For example, $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ and $|\varphi\rangle = -\alpha|0\rangle - \beta|1\rangle$ are equivalent up to global phase, but $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ and $|\varphi\rangle = \alpha|0\rangle - \beta|1\rangle$ are equivalent up to relative phase.

When operators are applied, the probability amplitudes of a state $U|\psi\rangle$ will in general differ by more than relative phase from those of $U|\varphi\rangle$, but the measurement outcomes may be equivalent. One can consider a hierarchy in which exact equivalence implies global-phase equivalence, which implies relative-phase equivalence, which in turn implies measurement outcome equivalence. The equivalence checking problem is also extensible to quantum operators with applications to quantum-circuit synthesis and verification, which involves computer-aided generation of minimal

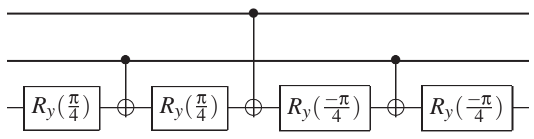


Figure 1: Margolus' circuit is equivalent up to relative phase to the Toffoli gate, which otherwise requires six *CNOT* and eight 1-qubit gates to implement.

quantum circuits with correct functionality. Extended notions of equivalence create several design opportunities. For example, the well-known three-qubit Toffoli gate can be implemented with fewer controlled-NOT (CNOT) and 1-qubit gates up to relative phase [3, 18] as shown in Figure 1. The relative-phase differences can be canceled out if every pair of these gates in the circuit is strategically placed [18]. Since circuit minimization is being pursued for a number of key quantum arithmetic circuits with many Toffoli gates, such as modular exponentiation [20, 8, 16, 15], this optimization could reduce the number of gates even further.

The inner product and matrix product may be used to determine such equivalences, but in this work, we present new decision-diagram (DD) algorithms to accomplish the task more efficiently. In particular, we make use of the quantum information decision diagram (QuIDD) [22, 21], a datastructure with unique properties that are exploited to solve this problem asymptotically faster in practical cases.

Empirical results confirm the algorithms' effectiveness and show that the improvements are more significant for the operators than for the states. Interestingly, solving the equivalence problems for the benchmarks considered requires significantly less time than creating the DD representations, which indicates that such problems can be reasonably solved in practice using quantum-circuit CAD tools.

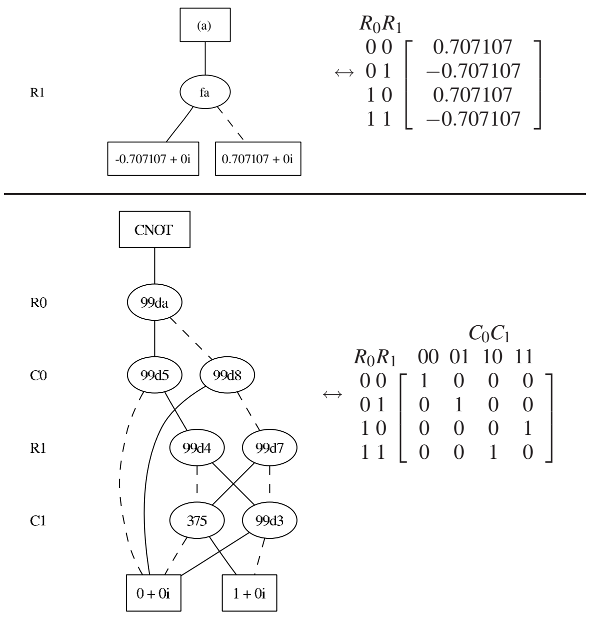
2 Background

The QuIDD is a variant of the reduced ordered binary decision diagram (ROBDD or BDD) datastructure [7] applied to quantum circuit simulation [22, 21]. Like other DD variants, it has all of the key properties of BDDs as well as a few other application-specific attributes (see Figure 2 for examples).

- It is a directed acyclic graph with internal nodes whose edges represent assignments to binary variables
- The leaf or terminal nodes refer to complex values
- Each path from the root to a terminal node is a functional mapping of row and column indices to complex-valued matrix elements ($f : \{0,1\}^n \to \mathbb{C}$)
- Nodes are unique and shared, meaning that nodes v and v' with isomorphic subgraphs do not exist
- Variables whose values do not affect the function output for a particular path (not in the *support*) are absent
- Binary row (R_i) and column (C_i) index variables have evaluation order $R_0 \prec C_0 \prec \ldots R_{n-1} \prec C_{n-1}$

Maintaining these properties makes many DDs such as QuIDDs canonical. Thus, exact equivalence checking may be performed in $O(1)$ time by comparing the root nodes, a technique which has been long exploited in the classical domain [19]. Quantum state and operator equivalence is less trivial as we show.

The algorithms which manipulate DDs are just as important as the properties of the DDs. In particular, the **Apply**



$$\leftrightarrow \begin{array}{c|c} R_0R_1 & \\ 0\,0 & 0.707107 \\ 0\,1 & -0.707107 \\ 1\,0 & 0.707107 \\ 1\,1 & -0.707107 \end{array}$$

$$\leftrightarrow \begin{array}{c|cccc} & & C_0C_1 & & \\ R_0R_1 & 00 & 01 & 10 & 11 \\ 0\,0 & 1 & 0 & 0 & 0 \\ 0\,1 & 0 & 1 & 0 & 0 \\ 1\,0 & 0 & 0 & 0 & 1 \\ 1\,1 & 0 & 0 & 1 & 0 \end{array}$$

Figure 2: Sample QuIDDs of (top) a 2-qubit equal superposition with relative phases and (bottom) the *CNOT* operator. Internal nodes (circles) have a unique hex ID depends on a variable to the left (dashed (solid) edge is 0 (1) assignment).

algorithm performs recursive traversals on DD operands to build new DDs using any desired unary or binary function [7]. Although originally intended for digital logic operations, **Apply** has been extended to linear-algebraic operations such as matrix addition and multiplication [2], as well as quantum-mechanical operations such as measurement and partial trace [22, 21]. The runtime and memory complexity of **Apply** is $O(|A||B|)$, where $|A|$ and $|B|$ are the sizes in number of nodes of the DDs A and B, respectively [7].[1] Thus, the complexity of DD-based algorithms is tied to the compression achieved by the datastructure.

3 Checking Equivalence up to Global Phase

This section describes algorithms that check global-phase equivalence of two quantum states or operators. The first two algorithms are known QuIDD-based linear-algebraic operations, while the remaining algorithms are new ones that exploit DD properties explicitly. The section concludes with experiments comparing all algorithms.

3.1 Product-based Checks

Since the quantum-circuit formalism models an arbitrary quantum state $|\psi\rangle$ as a unit vector, the inner product $\langle\psi \mid \psi\rangle = 1$. In the case of a global-phase difference between two states $|\psi\rangle$ and $|\varphi\rangle$, the inner product is the global-phase factor, $\langle\varphi \mid \psi\rangle = e^{i\theta}\langle\psi \mid \psi\rangle = e^{i\theta}$. Since $|e^{i\theta}| = 1$ for any θ, checking if the complex modulus of the inner product is 1 suffices to check global-phase equivalence for states. A QuIDD-based implementation of the inner product is derived based on previously defined QuIDD operations [22]. Since the operations are based on **Apply**, computing the global-phase difference in this way requires $O(|A||B|)$ time

[1] The runtime and memory complexity of the unary version acting on one DD A is $O(|A|)$ [7].

and memory for two QuIDDs A and B with sizes in nodes $|A|$ and $|B|$, respectively.

The matrix product can be used for checking global-phase equivalence between operators. In particular, since all quantum operators are unitary, the adjoint of each operator is its inverse. Thus, if two operators U and V differ by a global phase, then $UV^\dagger = e^{i\theta}I$. As with the inner product, this operation is based on previously defined QuIDD operations including matrix multiplication. This technique therefore requires $O((|U||V|)^2)$ time and memory.

3.2 Node-Count Check

The previous algorithms merely translate linear-algebraic operations to QuIDDs, but exploiting the following QuIDD property leads to faster checks.

Lemma 1 *The QuIDD* $A' = \textbf{Apply}(A, c, *)$*, where* $c \in \mathbb{C}$ *and* $c \neq 0$*, is isomorphic to A, hence* $|A'| = |A|$.

Proof. In creating A', **Apply** expands all of the internal nodes of A since c is a scalar, and the new terminals are the terminals of A multiplied by c. All terminal values t_i of A are unique by definition of a QuIDD [22]. Thus, $ct_i \neq ct_j$ for all i, j such that $i \neq j$. As a result, the number of terminals in A' is the same as in A. □

Lemma 1 states that two QuIDD states or operators that differ by a non-zero scalar factor, such as global phase, have the same number of nodes. Thus, equal node counts in QuIDDs are a necessary but not sufficient condition for global-phase equivalence. To see why it is not sufficient, consider two state vectors $|\psi\rangle$ and $|\varphi\rangle$ with elements w_j and v_k, respectively, where $j, k = 0, 1, \ldots N-1$. If some $w_j = v_k = 0$ such that $j \neq k$, then $|\varphi\rangle \neq e^{i\theta}|\psi\rangle$. The QuIDD representations of these states can have the same node counts. Despite this drawback, this check requires only $O(1)$ time since **Apply** is easily augmented to sum the number of distinct nodes as a QuIDD is created.

3.3 Recursive Check

Lemma 1 implies that a QuIDD-based algorithm can implement a sufficient condition for global-phase equivalence by accounting for terminal value differences. The algorithm is a modified version of **Apply**. In each recursive step, the variable depended upon in a node in one QuIDD must be the same as the variable depended upon in the other QuIDD's node. If it is not, then the two QuIDDs are not isomorphic and hence not equivalent up to global phase (early termination). When terminal values are reached in both QuIDDs, the magnitude of the ratio is computed. If it is not equal to 1, then the difference is not a phase factor. Also, if the ratio is not equal to previously computed terminal ratios, the difference is not a global phase factor.

In the worst case, both QuIDDs are isomorphic and all nodes are visited. Thus, the overall runtime and memory complexity of **GPRC** for states or operators is $O(|A| + |B|)$. Also, the node-count check can be run before **GPRC** to quickly eliminate many nonequivalences.

3.4 Empirical Results

The first benchmark considered is the logic associated with a single iteration of Grover's quantum search algorithm [10], which is depicted in Figure 3. One iteration is sufficient

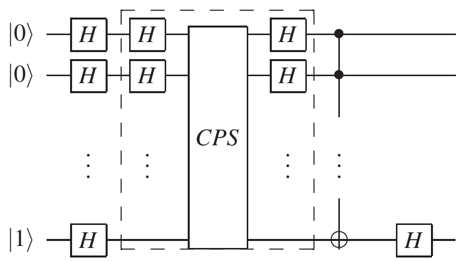


Figure 3: An iteration of Grover's algorithm, where the boxed part is the Grover iteration operator.

to test the effectiveness of the equivalence-checking algorithms since the state vector QuIDD remains isomorphic across all iterations [22].

Figure 4a shows the runtime results for the inner product and **GPRC** algorithms (no results are given for the node-count check algorithm since it runs in $O(1)$ time). The results confirm the asymptotic complexity differences between the algorithms. The number of nodes in the QuIDD state vector after a Grover iteration is $O(n)$ [22], which is confirmed in Figure 4b. As a result, the runtime complexity of the inner product should be $O(n^2)$, which is confirmed by a regression plot within 1% error. By contrast, the runtime complexity of the **GPRC** algorithm should be $O(n)$, which is also confirmed by another regression plot within 1% error. The matrix product and **GPRC** exhibited the same asymptotic behavior for checking the Grover operator.

The next benchmark compares states in Shor's integer factorization algorithm [17]. Specifically, we consider the states created by the modular exponentiation sub-circuit that represent all possible combinations of x and $f(x,N) = a^x modN$, where N is the integer to be factored [17] (see Figure 5). Each of the $O(2^n)$ paths to a non-0 terminal represents a binary value for x and $f(x,N)$. Thus, this benchmark tests performance with exponentially-growing QuIDDs.

We applied the inner product and **GPRC** algorithms to this benchmark for values of N that ranged from 12 to 19 qubits in size, and each N was composed of two non-trivial prime factors.[2] Interestingly, both algorithms exhibited nearly the same performance with runtimes under 10 seconds for the 19-qubit value. However, experiments in which the first, middle, and last qubits are acted upon by Hadamard gates that destroy global phase equivalence, **GPRC** outperforms the inner product by factors ranging from 1.5x to 10x. This improvement comes from the node count early termination condition.

In almost every case, both algorithms represent far less than 1% of the total runtime. Thus, checking for global-phase equivalence among QuIDD states appears to be an easily achievable task once the representations are created. An interesting side note is that some modular exponentiation QuIDD states with more qubits can have more exploitable structure than those with fewer qubits. For instance, the $N = 387929$ (19 qubits) QuIDD has fewer than half the nodes of the $N = 163507$ (18 qubits) QuIDD.

Table 1 contains results for the matrix product and **GPRC** algorithm checking the inverse Quantum Fourier

[2]Such integers are likely to be the ones input to Shor's algorithm since they play a key role in modern public key cryptography [17].

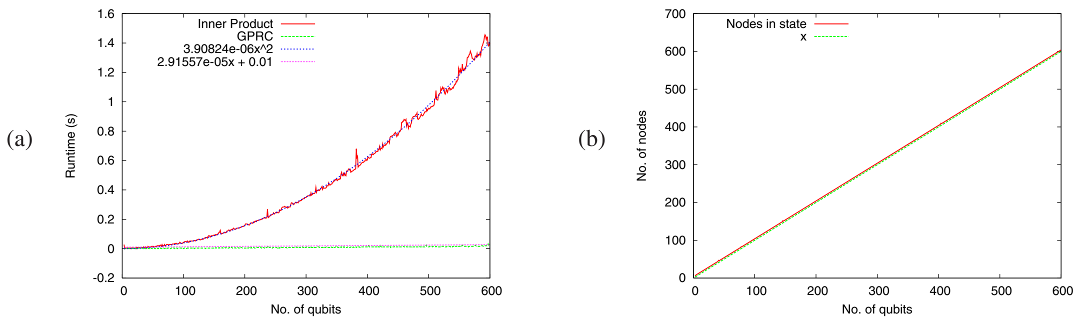


Figure 4: (a) Runtime results and regressions for the inner product and **GPRC** on checking global-phase equivalence of states generated by a Grover iteration. (b) Size in node count and regression of the QuIDD state vector.

No. of Qubits	Matrix Product		GPRC	
	Time (s)	Mem (MB)	Time (s)	Mem (MB)
5	2.53	1.41	0.064	0.25
6	22.55	6.90	0.24	0.66
7	271.62	46.14	0.98	2.03
8	3637.14	306.69	4.97	7.02
9	22717	1800.42	17.19	26.48

Table 1: Performance results for the matrix product and **GPRC** algorithms on checking global-phase equivalence of the QFT operator.

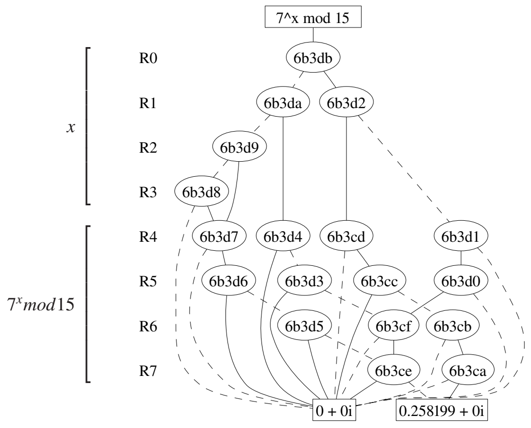


Figure 5: A QuIDD state combining x and $7^x mod 15$ in binary. The first qubit of each partition is least-significant.

Transform (QFT) operator. The inverse QFT is a key operator in Shor's algorithm [17], and it has been previously shown that its n-qubit QuIDD representation grows as $O(2^{2n})$ [22]. In this case, the asymptotic differences in the matrix product and **GPRC** are very noticeable. Also, the memory usage indicates that the matrix product may need asymptotically more intermediate memory despite operating on QuIDDs with the same number of nodes as **GPRC**.

4 Checking Equivalence up to Relative Phase

The relative-phase checking problem can also be solved in many ways. The first three algorithms are adapted from linear algebra to QuIDDs, while the last two exploit DD properties directly, offering asymptotic improvements.

4.1 Modulus and Products

Consider two state vectors $|\psi\rangle$ and $|\varphi\rangle$ that are equivalent up to relative phase and have complex-valued elements w_j and v_k, respectively, where $j,k = 0,1,\ldots,N-1$. Computing $|\varphi'\rangle = \Sigma_{i=0}^{N-1}|v_j||j\rangle$ and $|\psi'\rangle = \Sigma_{k=0}^{N-1}|w_k||k\rangle = \Sigma_{k=0}^{N-1}|e^{i\theta_k}v_k||k\rangle$ sets each phase factor to a 1, allowing the inner product to be applied as in Subsection 3.1. The complex modulus operations are computed as $C = \textbf{Apply}(A,|\cdot|)$ and $D = \textbf{Apply}(B,|\cdot|)$ with runtime and memory complexity $O(|A|+|B|)$, which is dominated by the $O(|A||B|)$ inner-product complexity.

For operator equivalence up to relative phase, two cases are considered, namely the diagonal relative-phase matrix appearing on the left or right side of one of the operators. Consider two operators U and V with elements $u_{j,k}$ and $v_{j,k}$, respectively, where $j,k = 0,\ldots N-1$. The two cases in which the relative-phase factors appear on either side of V are described as $u_{j,k} = e^{i\theta_j}v_{j,k}$ (left side) and $u_{j,k} = e^{i\theta_k}v_{j,k}$ (right side). In either case, the the matrix product check discussed in Subsection 3.1 may be extended by computing the complex modulus without increasing the overall complexity. Note that neither this algorithm nor the modulus and inner product algorithm calculate the relative-phase factors.

4.2 Element-wise Division

Given the states discussed in Subsection 4.1, $w_k = e^{i\theta_k}v_k$, the operation w_k/v_j for each $j = k$ is a relative-phase factor, $e^{i\theta_k}$. The condition $|w_k/v_j| = 1$ is used to check if each division yields a relative phase. If this condition is satisfied for all divisions, the states are equivalent up to relative phase.

The QuIDD implementation for states is simply $C = \textbf{Apply}(A,B,/)$, where **Apply** is augmented to avoid division by 0 and instead return 1 when two terminal values being compared equal 0, and return 0 otherwise. **Apply** can be further augmented to terminate early when $|w_j/v_i| \neq 1$. C is a QuIDD vector containing the relative-phase factors. If C contains a terminal value of 0, then A and B do not differ by relative phase. Since a call to **Apply** implements this algorithm, the runtime and memory complexity are $O(|A||B|)$.

Element-wise division for operators is more complicated. For QuIDD operators U and V, $W = \textbf{Apply}(U,V,/)$ is a QuIDD matrix with the relative-phase factor $e^{i\theta_j}$ along row j in the case of phases appearing on the left side and along column j in the case of phases appearing on the right side. In the first case, all rows of W are identical, meaning that the support of W does not contain any row variables. Similarly, in the second case the support of W does not contain any column variables. A complication arises when 0 values appear in either operator. In such cases, the support of W may contain both variable types, but the operators may in fact be equivalent up to relative phase. We implement an algorithm which accounts for these cases by recursively marking valid 0 entries with a sentinel value. Since the algorithm is a variant of **Apply**, the runtime and memory complexity are $O(|U||V|)$.

4.3 Non-0 Terminal Merge

A necessary condition for relative-phase equivalence is that zero-valued elements of each state vector appear in the same locations, as expressed by the following lemma.

Lemma 2 *A necessary but not sufficient condition for two states* $|\varphi\rangle = \Sigma_{j=0}^{N-1} v_j |j\rangle$ *and* $|\psi\rangle = \Sigma_{k=0}^{N-1} w_k |k\rangle$ *to be equivalent up to relative phase is that* $\forall v_j = w_k = 0,\ j = k$.

Proof. If $|\psi\rangle = |\varphi\rangle$ up to relative phase, $|\psi\rangle = \Sigma_{k=0}^{N-1} e^{i\theta_k} v_k |k\rangle$. Since $e^{i\theta_k} \neq 0$ for any θ, if any $w_k = 0$, then $v_j = 0$ must also be true where $j = k$. A counter-example proving insufficiency is $|\psi\rangle = (0, 1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})^T$ and $|\varphi\rangle = (0, 1/2, 1/\sqrt{2}, 1/2)^T$. □

QuIDD canonicity may now be exploited. Let A and B be the QuIDD representations of the states $|\psi\rangle$ and $|\varphi\rangle$, respectively. First compute $C = \textbf{Apply}(A, \lceil|\cdot|\rceil)$ and $D = \textbf{Apply}(B, \lceil|\cdot|\rceil)$, which converts every non-zero terminal value of A and B into a 1. Since C and D have only two terminal values, 0 and 1, checking if $C = D$ satisfies Lemma 2. Canonicity ensures this check requires $O(1)$ time and memory. The overall runtime and memory complexity of this algorithm is $O(|A| + |B|)$ due to the unary **Apply** operations. This algorithm also applies to operators since Lemma 2 also applies to $u_{j,k} = e^{i\theta_j} v_{j,k}$ (phases on the left) and $u_{j,k} = e^{i\theta_k} v_{j,k}$ (phases on the right) for operators U and V.

4.4 Modulus and DD Compare

A variant of the algorithm presented in Subsection 4.1, which also exploits canonicity, provides an asymptotic improvement for checking a necessary and sufficient condition for relative-phase equivalence of states and operators. As in Subsection 4.1, compute $C = \textbf{Apply}(A, |\cdot|)$ and $D = \textbf{Apply}(B, |\cdot|)$. If A and B are equivalent up to relative phase, then $C = D$ since each phase factor becomes a 1. This check requires $O(1)$ time and memory due to canonicity. Thus, the runtime and memory complexity is dominated by the unary **Apply** operations, giving $O(|A| + |B|)$.

4.5 Empirical Results

The first benchmark for the relative-phase equivalence checking algorithms creates a remote EPR pair, which is an EPR pair between the first and last qubits, via nearest-neighbor interactions [6]. The circuit is shown in Figure 6. Specifically, it transforms the initial state $|00\ldots0\rangle$ into $(1/\sqrt{2})(|00\ldots0\rangle + |10\ldots1\rangle)$. The circuit size is varied, and the final state is compared to the state $(e^{0.345i}/\sqrt{2})|00\ldots0\rangle + (e^{0.457i}/\sqrt{2})|10\ldots1\rangle$.

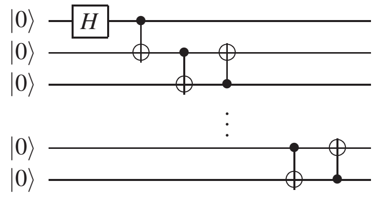


Figure 6: Remote EPR-pair creation between the first and last qubits via nearest-neighbor interactions.

The results in Figure 7a show that all algorithms run quickly. The inner product is the slowest, yet it runs in approximately 0.2 seconds at 1000 qubits, a small fraction of the 7.6 seconds required to create the QuIDD state vectors. Regressions of the runtime and memory data reveal linear complexity for all algorithms to within 1% error. This is not unexpected since the QuIDD representations of the states grow linearly with the number of qubits (see Figure 7b), and the complex modulus reduces the number of different terminals prior to computing the inner product. These results illustrate that in practice, the inner product and element-wise division algorithms can perform better than their worst-case complexity. Element-wise division should be preferred when QuIDD states are compact since unlike the other algorithms, it computes the relative-phase factors.

The Hamiltonian simulation circuit shown in Figure 8 is taken from [13, Figure 4.19, p. 210]. When its one-qubit gate (boxed) varies with Δt, it produces a variety of diagonal operators, all of which are equivalent up to relative phase. Empirical results for such equivalence checking were similar to the remote EPR pair results and exhibited the same asymptotic trends. As before, the matrix product and element-wise division algorithms perform better than their worst-case bounds, indicating that element-wise division is the best choice for compact QuIDDs.

5 Conclusions

We have shown that various DD properties can be exploited to develop efficient algorithms for the difficult problem of equivalence checking up to global and relative phase. The recursive check and element-wise division algorithms efficiently determine equivalence of states and operators up to global and relative phase, and compute the phases. They outperform QuIDD matrix and inner products, which do not compute relative-phase factors. Other QuIDD algorithms presented, such as the node-count check, non-0 terminal merge, and modulus and DD compare, provide even faster checks but only satisfy necessary equivalence conditions. Thus, they should be used to aid the more robust algorithms. A summary of the theoretical results is provided in Table 2.

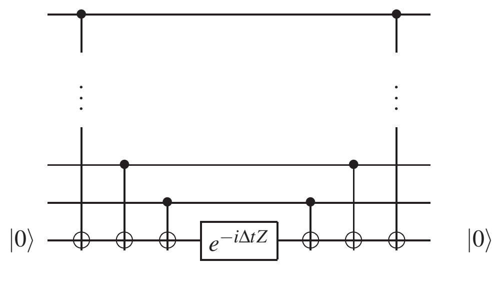


Figure 8: A quantum-circuit realization of a Hamiltonian consisting of Pauli operators.

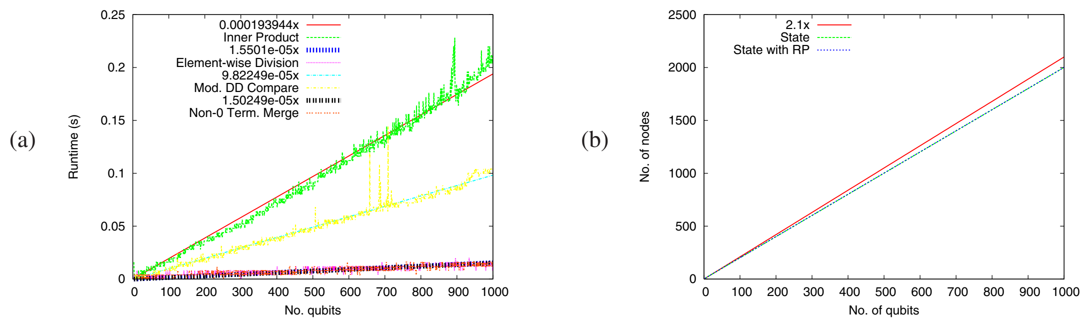


Figure 7: (a) Runtime results and (b) size in nodes plotted with regressions for inner product, element-wise division, modulus and DD compare, and non-0 terminal merge checking relative-phase equivalence of the remote EPR pair circuit.

Algorithm	Phase type	Finds phases?	Necessary & sufficient?	$O(\cdot)$ time complexity: best-case	$O(\cdot)$ time complexity: worst-case
Inner Product	Global	Yes	N. & S.	$\|A\|\|B\|$	$\|A\|\|B\|$
Matrix Product	Global	Yes	N. & S.	$(\|A\|\|B\|)^2$	$(\|A\|\|B\|)^2$
Node-Count	Global	No	N. only	1	1
Recursive Check	**Global**	**Yes**	**N. & S.**	**1**	$\|\mathbf{A}\|+\|\mathbf{B}\|$
Modulus and Inner Product	Relative	No	N. & S.	$\|A\|\|B\|$	$\|A\|\|B\|$
Element-wise Division	**Relative**	**Yes**	**N. & S.**	$\|\mathbf{A}\|\|\mathbf{B}\|$	$\|\mathbf{A}\|\|\mathbf{B}\|$
Non-0 Terminal Merge	Relative	No	N. only	$\|A\|+\|B\|$	$\|A\|+\|B\|$
Modulus and DD Compare	Relative	No	N. & S.	$\|A\|+\|B\|$	$\|A\|+\|B\|$

Table 2: Key properties of the QuIDD-based phase-equivalence checking algorithms.

A fair amount of work has been done on optimal and heuristic synthesis for quantum circuits [14, 15]. Equivalence checking in particular plays a key role in some of these techniques since it is often necessary to verify the correctness of heuristic transformations. Future work will determine how these equivalence checking algorithms may be used as primitives to enhance such heuristics.

Acknowledgements. This work was sponsored in part by the Air Force Research Laboratory under Agreement No. FA8750-05-1-0282.

References

[1] M. H. S. Amin et al., "Superconducting quantum storage and processing," *Digest ISSCC*, pp. 296-299, Feb. 2004.

[2] R. I. Bahar et al., "Algebraic decision diagrams and their applications," *Journal of Formal Methods in System Design*, **10** (2/3), 1997.

[3] A. Barenco et al., "Elementary gates for quantum computation," *Phys. Rev. A*, **52**, 3457-3467, 1995.

[4] C. H. Bennett and G. Brassard, "Quantum cryptography: public key distribution and coin tossing", *In Proc. of IEEE Intl. Conf. on Computers, Systems, and Signal Processing*, pp. 175-179, 1984.

[5] C.H. Bennett, "Quantum cryptography using any two nonorthogonal states", *Phys. Rev. Lett.* **68**, 3121, 1992.

[6] G. P. Berman, G. V. López, and V. I. Tsifrinovich, "Teleportation in a nuclear spin quantum computer," *Phys. Rev. A* **66**, 042312, 2002.

[7] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Computers*, **C35**, pp. 677-691, 1986.

[8] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," `quant-ph/0410184`, 2004.

[9] D. Gottesman, "The Heisenberg representation of quantum computers," *Plenary speech at the 1998 International Conference on Group Theoretic Methods in Physics*, `quant-ph/9807006`, 1998.

[10] L. Grover, "Quantum mechanics helps in searching for a needle in a haystack," *Phys. Rev. Lett.* **79**, 325, 1997.

[11] A. J. G. Hey, ed., *Feynman and Computation: Exploring the Limits of Computers*, Perseus Books, 1999.

[12] D. Maslov, S. M. Falconer, and M. Mosca, "Quantum Circuit Placement: Optimizing Qubit-to-qubit Interactions through Mapping Quantum Circuits into a Physical Experiment," *Proc. DAC*, pp. 962-965, June 2007.

[13] M. A. Nielsen, I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge Univ. Press, 2000.

[14] A. K. Prasad, V. V. Shende, K. N. Patel, I. L. Markov, and J. P. Hayes, "Algorithms and data structures for simplifying reversible circuits", to appear in *ACM J. of Emerging Technologies in Computing*, 2007.

[15] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of quantum logic circuits," *IEEE Trans. on CAD* **25**, pp. 1000-1010, 2006.

[16] V. V. Shende and I. L. Markov, "Quantum circuits for incompletely specified two-qubit operators," *Quantum Information and Computation* **5** (1), pp. 49-57, 2005.

[17] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. of Computing* **26**, p. 1484, 1997.

[18] G. Song and A. Klappenecker, "Optimal realizations of simplified Toffoli gates," *Quantum Information and Computation* **4**, pp. 361-372, 2004.

[19] R. T. Stanion, D. Bhattacharya, and C. Sechen, "An efficient method for generating exhaustive test sets," *IEEE Trans. on CAD* **14**, pp. 1516-1525, 1995.

[20] R. Van Meter and K. M. Itoh, "Fast quantum modular exponentiation," *Phys. Rev. A* **71**, 052320, 2005.

[21] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Graph-based simulation of quantum computation in the density matrix representation," *Quantum Information and Computation* **5** (2), pp. 113-130, 2005.

[22] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Improving gate-level simulation of quantum circuits," *Quantum Information Processing* **2**, pp. 347-380, 2003.

[23] G. Vidal, "Efficient classical simulation of slightly entangled quantum computations," *Phys. Rev. Lett.* **91**, 147902, 2003.

[24] J. Yepez, "A quantum lattice-gas model for computational fluid dynamics," *Phys. Rev. E* **63**, 046702, 2001.
]