Learning the quantum algorithm for state overlap

New Journal of Physics
The open access journal at the forefront of physics

Deutsche Physikalische Gesellschaft DPG
IOP Institute of Physics

Published in partnership with: Deutsche Physikalische Gesellschaft and the Institute of Physics

RECEIVED
9 June 2018

REVISED
10 October 2018

ACCEPTED FOR PUBLICATION
18 October 2018

PUBLISHED
14 November 2018

PAPER

Learning the quantum algorithm for state overlap

Lukasz Cincio[1], Yiğit Subaşı[1], Andrew T Sornborger[2] and Patrick J Coles[1]

[1] Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM 87545, United States of America
[2] Information Sciences, Los Alamos National Laboratory, Los Alamos, NM 87545, United States of America

E-mail: lcincio@lanl.gov

Keywords: quantum computing, machine-learning, state overlap

Abstract

Short-depth algorithms are crucial for reducing computational error on near-term quantum computers, for which decoherence and gate infidelity remain important issues. Here we present a machine-learning approach for discovering such algorithms. We apply our method to a ubiquitous primitive: computing the overlap $\mathrm{Tr}(\rho\sigma)$ between two quantum states ρ and σ. The standard algorithm for this task, known as the Swap Test, is used in many applications such as quantum support vector machines, and, when specialized to $\rho = \sigma$, quantifies the Renyi entanglement. Here, we find algorithms that have shorter depths than the Swap Test, including one that has a constant depth (independent of problem size). Furthermore, we apply our approach to the hardware-specific connectivity and gate sets used by Rigetti's and IBM's quantum computers and demonstrate that the shorter algorithms that we derive significantly reduce the error—compared to the Swap Test—on these computers.

1. Introduction

Quantum supremacy [1] may be coming soon [2]. While it is an exciting time for quantum computing, decoherence and gate fidelity continue to be important issues [3]. Ultimately these issues limit the depth of algorithms that can be implemented on near-term quantum computers (NTQCs) and increase the computational error for short-depth algorithms. Furthermore, NTQCs do not currently have enough qubits and sufficient gate fidelities to fully leverage the benefit of quantum error-correcting codes [4, 5]. This highlights the need for general methods to reduce the depth of quantum algorithms in order to avoid the accumulation of errors.

Analytical efforts to find short-depth algorithms face several challenges. First, quantum algorithms are fairly non-intuitive to classically trained minds. Second, actual NTQCs may not be fully connected. Third, different NTQCs use different fundamental gate sets. It may not be obvious how to optimize algorithms for a given connectivity and a given gate set. This motivates the idea of an automated approach for discovering and optimizing quantum algorithms [6–19].

An analogous problem in classical computing, known as logic synthesis, has a relatively longer history and has been extensively studied [20]. Machine-learning methods have been used in this context. For instance [21] shows how logic optimization algorithms can be discovered automatically through the use of deep learning.

In this work, we take a machine-learning approach to developing quantum algorithms, see figure 1. Our approach can be applied either to ideal hardware to derive fundamental algorithms or to a non-fully connected hardware with a non-ideal gate set to derive hardware-specific algorithms. We conceptually divide a quantum computation into the available resources, consisting of input qubits (data qubits and ancilla qubits) and output measurements, and the algorithm, consisting of a quantum gate sequence and classical post-processing of the measurement results (see figure 1). Fixing the resources as hyperparameters, we optimize the algorithm in a task-oriented manner, i.e. by minimizing a cost function that quantifies the discrepancy between the algorithm's output and the desired output. The task is defined by a training data set that exemplifies the desired computation. Our machine-learning approach is used to discover small algorithm instances that can be later manually generalized to arbitrary problem size.

We emphasize that our work goes beyond quantum compiling, which has received recent attention [11–16]. Quantum compiling corresponds to finding a hardware-specific gate sequence that generates the same unitary as

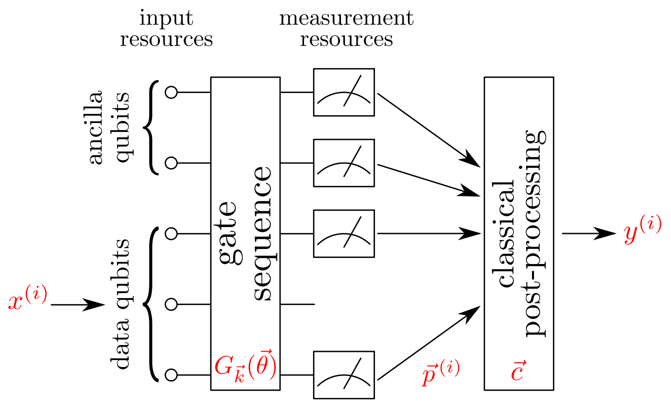

Figure 1. Machine-learning approach to discovering and optimizing quantum algorithms. We optimize an algorithm for a given set of resources, which includes input resources (ancilla and data qubits) and measurement resources (i.e. which qubits can be measured). The algorithm is then determined by the quantum gate sequence and the classical post-processing of the measurement results. To find the algorithm that computes the function $x \to f(x)$, we minimize a cost function that quantifies the discrepancy between the desired output $f(x^{(i)})$ and the actual output $y^{(i)}$ for a set of training data inputs $\{x^{(i)}\}$. If the training data are sufficiently general, the algorithm that minimizes the cost should be a general algorithm that computes $f(x)$ for any input x.

a high-level gate sequence defined for an idealized hardware. Various techniques have been employed in these works such as temporal planning (e.g. [11]). Machine-learning techniques have also been used to decompose small scale unitaries into one and two-body gates [17, 18]. Although our method can be used in this way to optimally compile a known unitary or gate sequence, our main goal is to discover novel algorithms via task-oriented programming.

Other automated algorithm-discovery approaches have been employed in the literature. Gepp and Stocks [9] review much of the early work to evolve quantum algorithms using genetic programming such as [10] (for more recent work see for example [19]). In these approaches the gate set is typically discrete. An alternative approach is to define an ansatz or template for the quantum circuit composed of gates that depend on continuous parameters. The circuit is then trained to perform a given task by tuning these parameters [6, 7]. Our approach is distinct from previous works in that we do not start with an ansatz or template for the quantum circuit; nor do we restrict to a discrete gate set as is usually done in algorithms based on genetic programming. In this sense our approach combines desirable aspects of the two types of approaches in the literature.

We apply our approach to a ubiquitous task: computing the overlap between two quantum states. This computation yields $|\langle\psi|\phi\rangle|^2$ for two pure states $|\psi\rangle$ and $|\phi\rangle$, and more generally gives $\mathrm{Tr}(\rho\sigma)$ for two (possibly mixed) states ρ and σ. Furthermore, when specialized to the case $\rho = \sigma$, it computes the purity $\mathrm{Tr}(\rho^2)$ of a given state ρ.

There is a well-known algorithm for this task called the Swap Test [22, 23]. In quantum optics the Swap Test has a simple physical implementation [24–26]. However, for gate-based quantum computers (e.g. IBM's, Google's, and Rigetti's superconducting quantum computers and IonQ's trapped-ion quantum computer), the optimal implementation of the Swap Test is not obvious, and for single-qubit states involves 14 and 34 gates for IBM's five-qubit and Rigetti's 19-qubit quantum computer respectively, see figure 2. Larger gate count for Rigetti's computer is mainly due to its lower connectivity. For example, the Swap Test was experimentally implemented on a five-qubit computer based on trapped ions [27] to quantify entanglement, with an algorithm employing 7 two-qubit gates and 11 one-qubit gates. Figures 2(B) and (C) respectively show decompositions of the Swap Test for IBM's and Rigetti's quantum computers [28, 29]. This highlights the non-trivial nature of implementing the Swap Test algorithm.

Here, our machine-learning approach finds algorithms with a shorter depth than the Swap Test for computing the overlap. We do this by initially specializing the training data to one- and two-qubit states and then manually generalizing the resulting algorithms to input states of arbitrary size. We first consider the same 'quantum resources' as the Swap Test (access to a qubit ancilla and measurement on the ancilla), and our approach reduces the gate count to 4 controlled-NOTs (CNOTs) and 4 one-qubit gates. We call this our Ancilla-based algorithm (ABA). Then we allow for the additional resource of measuring all of the qubits, which gives an even shorter depth algorithm that essentially corresponds to a Bell-basis measurement with classical post-processing. We call this our Bell-basis algorithm (BBA). This algorithm has a constant depth of two gates, while the classical post-processing scales linearly in the number of qubits of the input states. In that regard, our machine-learning approach independently discovered the algorithm of Garcia-Escartin and Chamorro-Posada for computing state overlap [24]. We also find short-depth algorithms for the specific hardware connectivity and

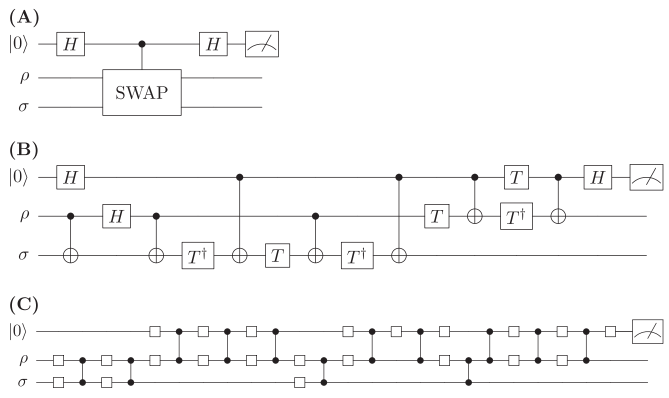

Figure 2. Swap Test circuits. (A) The canonical Swap Test circuit. H indicates the Hadamard gate. (B) The Swap Test circuit adapted for IBM's five-qubit quantum computer, constructed by decomposing controlled-swap into the Toffoli gate, via [34, 35], and then manually eliminating gates that had no effect on the output. T is the $\pi/8$ phase gate. (C) The structure of a Swap Test circuit, showing the locations of the one-qubit gates and controlled-Z gates, constructed automatically by Rigetti's compiler for their 19-qubit quantum computer. Appendix A gives the full specification of that circuit.

gate sets used by IBM's and Rigetti's quantum computers, which is crucial for reducing the computational error. Indeed, we found that our short-depth algorithms reduced the root mean square (rms) error (compared to the Swap Test) by 66% on IBM's five-qubit computer and by 70% on Rigetti's 19-qubit computer.

Due to the fundamental nature of computing state overlap, the Swap Test appears in many applications. In quantum supervised learning [30, 31], which subsumes quantum support vector machines [32], the Swap Test is used to assign each data vector to a cluster. The Swap Test allows one to quantify entanglement for many-body quantum states [27, 33] using the Renyi order-2 entanglement, given by $H^{(2)} = -\log\mathrm{Tr}(\rho^2)$. The Swap Test is useful for benchmarking on a quantum computer, since it can quantify the purity $\mathrm{Tr}(\rho^2)$ and hence the amount of decoherence that has occurred. For all of the above applications, one of our shorter-depth algorithms can be directly substituted in place of the Swap Test.

Note that if ρ and σ represent states on n qubits, the difficulty for computing $\mathrm{Tr}(\rho\sigma)$ scales exponentially with n for a classical computer. In contrast, the Swap Test has a circuit depth that grows linearly in n, giving an exponential speedup. Our ABA also has this property of scaling linearly with n, and it reduces the number of gates in the circuit by a factor of $\sim$2.3 (relative to the Swap Test circuit decomposed in terms of CNOTs, as shown in figure 2(B)). On the other hand, our BBA has the nice feature that its circuit depth is constant, independent of n (although the complexity of its classical post-processing grows linearly in n). Due to its constant circuit depth, the BBA seems to be the best algorithm for quantifying state overlap on NTQCs.

In what follows, we first present our machine-learning approach for discovering quantum algorithms. This approach can be used to find other algorithms besides the one that computes the overlap and hence should be of independent interest. We also give the full details of the approach and discuss its scaling with various resources. Next, we present our main results: short-depth circuits for computing state overlap on idealized hardware. Then, we present hardware-specific algorithms for computing overlap. Finally we discuss our implementation of these algorithms on Rigetti's and IBM's quantum computers, leading to a reduction in the computational error relative to the Swap Test.

2. Machine-learning approach

Our machine-learning approach is summarized in figure 1. The variables are divided up into the hyperparameters (i.e. the 'resources') and the optimization parameters (i.e. the 'algorithm').

2.1. Resources

The hyperparameters are the quantum resources of the circuit. At the input, the resources are the number of ancilla qubits and data qubits that store the input data for the computation. At the output, the resources are the locations of the measurements (see figure 1). As an example, in the Swap Test for single-qubit states, we are allowed access to one ancilla qubit and two data qubits at the input, and we can measure only the ancilla qubit at the output.

The input data may be classical or quantum, depending on the computation of interest. In the case of state overlap, the input data are quantum states and hence no encoding is necessary. However, for completeness, we note that our approach also applies to classical inputs, in which case the encoding (i.e. storing the classical data in the quantum state of the data qubits) can be treated as a hyperparameter that one fixes while optimizing the algorithm.

2.2. Algorithm

Our approach searches for an optimal algorithm, where we consider the algorithm to be a quantum gate sequence with associated classical post-processing. We parameterize (and hence optimize over) both the gate sequence and the post-processing.

Let us first consider the gate sequence. We define a gate set $\mathcal{A} = \{A_j(\theta)\}$. Here, each gate A_j is either a one-qubit or two-qubit gate and may also have an internal continuous parameter θ. Hence, $\mathcal{A}$ is a discrete set, but each element of $\mathcal{A}$ may have a continuous parameter associated with it. The precise choice of $\mathcal{A}$ depends on which hardware one is considering. For example, the connectivity differs between IBM and Rigetti hardware, and the former employs CNOT gates while the latter employs controlled-Z gates. For IBM's five-qubit computer 'ibmqx4' we can write out the gate set as

$$\begin{aligned}\mathcal{A}_{\text{ibmqx4}} = \{&\text{CNOT}^{10},\ \text{CNOT}^{20},\ \text{CNOT}^{21},\ \text{CNOT}^{32},\\ &\text{CNOT}^{24},\ \text{CNOT}^{34},\ U^0(\theta),\ U^1(\theta),\ U^2(\theta),\\ &U^3(\theta),\ U^4(\theta)\},\end{aligned} \tag{1}$$

where $U^j(\theta)$ is an arbitrary gate on qubit j and CNOT^{jk} is a CNOT from control qubit j to target qubit k. Angles θ in equation (1) may be encoding multiple parameters. In this article, we treat all one-qubit gates equally in the sense that all one-qubit gates are equally complex to implement, although our approach could easily be generalized to account for different complexities for different one-qubit gates.

We consider a generic sequence of d gates

$$G_{\vec{k}}(\vec{\theta}) = A_{k_d}(\theta_d)\ \cdots\ A_{k_2}(\theta_2)A_{k_1}(\theta_1), \tag{2}$$

where $\vec{k} = (k_1,\ \dots,\ k_d)$ is the vector of indices describing which gates are employed in the gate sequence and $\vec{\theta} = (\theta_1,\dots,\theta_d)$ is the vector of continuous parameters associated with these gates.

The measurement results give rise to an outcome probability vector $\vec{p} = (p_1,\dots,p_l,\ \dots)$. The desired output might be one of these probabilities p_l, or it might be some simple function of these probabilities. Hence, we allow for some simple classical post-processing of $\vec{p}$ in order to reveal the desired output. While there is enormous freedom in applying a function to $\vec{p}$, we consider a simple linear combination of probabilities:

$$g(\vec{p}) = \vec{c}\cdot\vec{p} = \sum_l c_l p_l, \tag{3}$$

where $\vec{c}$ is a vector of coefficients whose elements are chosen according to $c_l \in \{-1, 0, 1\}$. This post-processing is sufficient for the application in this paper (state overlap), although other applications may require a more general form of post-processing. Note that in our approach it is enough to consider measurements in the computational basis, as any change of the measurement basis can be incorporated into the gate sequence in equation (2). In particular, this implies that equation (3) is general enough to cover the expectation values of all Pauli product operators.

In summary, the free parameters that we optimize over (while fixing the hyperparameters) are the gate sequence vector $\vec{k}$, the continuous parameter vector $\vec{\theta}$, and the post-processing vector $\vec{c}$. For a given set of resources, these three vectors define the quantum algorithm, which we denote $Q_{\vec{m}}$, where $\vec{m} = (\vec{k},\ \vec{\theta},\ \vec{c})$ is the concatenated vector.

2.3. Optimization

Optimizing these parameters involves defining and minimizing a cost function. The cost quantifies the discrepancy between the desired output and the actual output for a given training data set.

Suppose we want to find the algorithm that computes the function $x \to f(x)$. We generate data of the form

$$T = \{(x^{(i)}, f(x^{(i)}))\}_{i=1}^{2N}. \tag{4}$$

Half of this data is used for training the algorithm, i.e. optimizing the cost function. The other half is used for testing, i.e. evaluating the algorithm's performance. The training data must be sufficiently general to cover the space of possible inputs. An estimate of the amount of training data needed for state overlap is $N \approx 2^{2n_D}$, where n_D is the number of data qubits. This can be seen by noting that our algorithm (which includes both the gate sequence and the post-processing) acts as a linear map from the data qubits' density operator space, which has dimension 2^{2n_D}, to the output which is just a number and hence has dimension one. So our algorithm is basically a 1×2^{2n_D} matrix, and an estimate of the number of constraints (and hence the number of training data points) needed to fix the algorithm's parameters is 2^{2n_D}.

For example, when training the algorithm that computes overlap, $x^{(i)} = (\rho^{(i)}, \sigma^{(i)})$ consists of two quantum states $\rho^{(i)}$ and $\sigma^{(i)}$, and $f(x^{(i)}) = \text{Tr}(\rho^{(i)}\sigma^{(i)})$ quantifies their overlap. One can show that any algorithm that

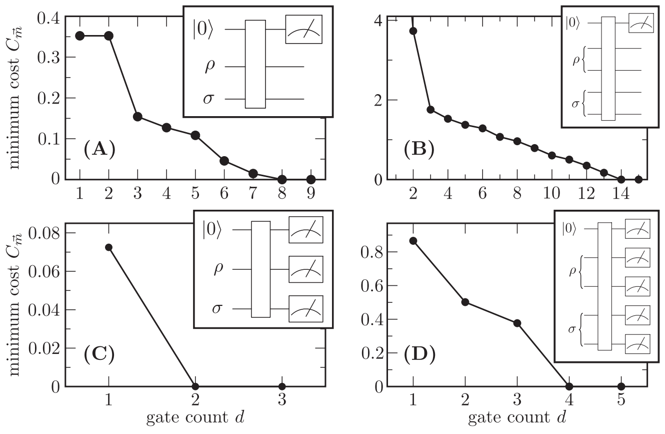

Figure 3. Final cost that we obtained after minimizing our cost function versus the circuit gate count *d*. (A) The resources allowed (shown in the inset) are the same as those allowed in the Swap Test, i.e. one ancilla qubit, two data qubits, and one measurement on the ancilla. This results in a minimum gate count of $d_{\min} = 8$. (B) The number of qubits in ρ and σ is increased, resulting in $d_{\min} = 14$ for $n = 2$ qubits. This procedure leads to the discovery of a general algorithm presented in figure 5. (C) Allowing for additional resources (shown in the inset) of measurements on all of the qubits results in a minimum gate count of $d_{\min} = 2$. (D) Again we increase the number of qubits in ρ and σ, giving $d_{\min} = 4$ for $n = 2$ qubits, when measurements on all qubits are allowed. As a result, a general algorithm is obtained, as shown in figure 6.

computes pure-state overlap also computes mixed-state overlap. Hence, we generate our training data by randomly choosing pure states according to the Haar measure.

Next we define a cost function. For algorithm $Q_{\vec{m}}$, the cost is

$$C_{\vec{m}} = \sum_{i=1}^{N} (f(x^{(i)}) - y_{\vec{m}}^{(i)})^2. \tag{5}$$

The cost quantifies the difference between the ideal output $f(x^{(i)})$ and the actual output $y_{\vec{m}}^{(i)}$ for each training data point. The actual output can be written as

$$y_{\vec{m}}^{(i)} = y_{(\vec{k},\vec{\theta},\vec{c})}^{(i)} = \vec{c}\cdot\vec{p}_{(\vec{k},\vec{\theta})}^{(i)}, \tag{6}$$

where $\vec{c}$ is the post-processing vector and $\vec{p}_{(\vec{k},\vec{\theta})}^{(i)}$ is the outcome probability vector for input $x^{(i)}$. For example, in the Swap Test, the outcome probability vector corresponds to the ancilla qubit's measurement in the *Z* basis. Choosing $\vec{c} = (1, -1)$ ensures that $y_{\vec{m}}^{(i)}$ is the expectation value of the Pauli *Z* operator.

For a fixed circuit gate count *d*, we search over the algorithm space to minimize the cost, as discussed below. We consider various *d*, incrementing from small to large values. When an exact algorithm exists, we typically are able to minimize the cost. That is, we can find a $Q_{\vec{m}}$ with $C_{\vec{m}} \approx 0$, for $d \geqslant d_{\min}$, where $d_{\min}$ is the minimum number of gates needed to minimize the cost (see figure 3 for example plots of final cost versus *d*). Note that some elements of the gate set in equation (1) commute with each other. As a consequence, there are typically many $Q_{\vec{m}}$ that give zero cost for $d \geqslant d_{\min}$. This freedom is used to simplify the algorithm at the end of the cost optimization. So, in the main results section, we present our simplest representation of such algorithms.

2.4. Details of the optimization techniques

The cost in equation (5) is a function of several parameters that can be divided into two groups: discrete and continuous. Discrete parameters are those which describe the circuit topology and post-processing of the algorithm. These are the gate sequence vector $\vec{k}$ and the post-processing vector $\vec{c}$. The angles $\vec{\theta}$ are treated as continuous parameters. They define all gates that depend on a parameter. For IBM and Rigetti architectures considered here, angles $\vec{\theta}$ specify all one-qubit gates present in the algorithm. Only the total number of gates *d* is fixed during optimization, which means that while the length of $\vec{k}$ does not change, the number of elements of $\vec{\theta}$ may vary as the optimization proceeds.

The optimization is performed in iterations until the cost reaches a (possibly local) minimum. Figure 4 shows a schematic description of a single iteration of the optimization algorithm. Each iteration begins with an attempt to modify $\vec{k}$ and $\vec{c}$. While modifying $\vec{k}$, we consider random updates that may involve an arbitrary number of gates. However, updates affecting a smaller number of gates are more probable. In this process, we may change the position or support of a given gate or change its type, e.g. from a one-qubit gate to a CNOT. The update is constrained to result in an algorithm that cannot be easily shortened. For example, the gate sequence that involves two one-qubit gates next to each other is not allowed, since those gates can be combined to a single

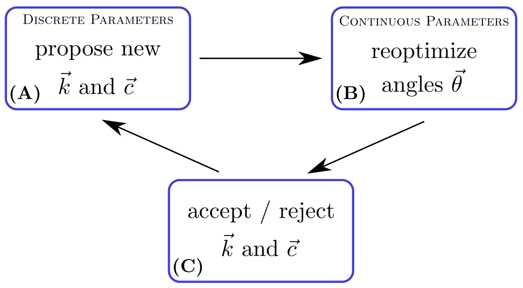

Figure 4. Schematic view of one iteration of the cost optimization procedure. (A) Iteration begins with a random update to the gate sequence vector $\vec{k}$ that describes the algorithm's structure and a random update to the post-processing vector $\vec{c}$. (B) Continuous parameters $\vec{\theta}$ of every one-qubit gate are reoptimized using the steepest descent method. (C) The optimization in the previous step gives a cost that is compared with the current best one. Based on the outcome of that comparison, new vectors $\vec{k}$ and $\vec{c}$ are either accepted or rejected. See text for details.

one-qubit gate. This is a desired feature, as we optimize with a fixed total number of gates. Similarly, we randomly modify $\vec{c}$ giving more preference to changes affecting fewer measurements.

Every change in $\vec{k}$ or $\vec{c}$ is followed by reoptimization of continuous parameters $\vec{\theta}$. This is an important step, as changing the gate sequence or post-processing function alone, without reoptimizing the gates' internal parameters $\vec{\theta}$, will most likely cause the cost to increase significantly, effectively suppressing any update of $\vec{k}$ or $\vec{c}$. The continuous part of the optimization is done in a sweeping fashion in which all one-qubit gates are updated sequentially. In this approach, at a given time, a single one-qubit gate is updated while all remaining gates are fixed. After the best one-qubit gate (the one that minimizes the cost) is identified, the optimization algorithm moves to the next one-qubit gate. We allow for randomly changing the order of updating one-qubit gates as a means to avoid local minima. We use a steepest descent method to optimize single one-qubit gates. Note that an arbitrary one-qubit gate can be described (up to a global phase, that does not affect the algorithm) by three real parameters. That is, the steepest descent method mentioned above operates in three-dimensional space. The continuous part of the optimization is repeated, until convergence of the cost function is achieved.

Once the continuous optimization has converged, we compare the final cost C in a given iteration with the current best one C_{best}. If the cost C is lower than the current best, the new discrete parameters $\vec{k}$ and $\vec{c}$ are accepted. If it is larger, the change is accepted with probability exponentially decreasing in the difference $C - C_{\text{best}}$ following the simulated annealing method.

Every few iterations we check whether the current gate sequence $G_{\vec{k}}$ can be compressed. This goes well beyond the simple checks following the update of vector $\vec{k}$ described above. Here, we are trying to find a subsequence of $G_{\vec{k}}$ that can be nontrivially rewritten using the same or a smaller number of gates. If such a subsequence is found, we modify $G_{\vec{k}}$ accordingly, as this may lead to shortening the gate sequence without increasing the cost. Since the total number of gates is fixed, such compression results in the ability to add gates to the sequence. If that is the case, we insert one-qubit identity gates and reoptimize their continuous parameters as described above. To check if a given subsequence can be rewritten we recursively use the same approach that we use for the full algorithm, which is essentially described in figure 4 except in this case we do not consider the post-processing vector $\vec{c}$.

We remark that the cost function may be difficult to optimize primarily due to many low lying local minima. Thus, it is important to develop techniques to increase the chances of avoiding them. We found it particularly useful to compress the gate sequence periodically, as random updates to vectors $\vec{k}$ and $\vec{c}$ tend to produce local minima that usually include redundant subsequences. As described above, we have developed automated tools to remove such subsequences, which usually allows us to escape local minima.

Let us now discuss the scaling of the approach described above. The optimization requires the cost to be evaluated multiple times during every iteration. As part of computing the cost, one has to evaluate $y_{\vec{m}}^{(i)}$ in equation (5) for each training data point, which necessarily scales exponentially with the number of qubits on a classical computer. However, it can be outsourced to a quantum computer. Such a hybrid algorithm will efficiently compute the contribution to the cost from a single element of a training data set, although the resulting cost will reflect the quantum hardware's noise. In this work, we evaluate the cost on a classical computer, as we are mainly interested in the discovery of theoretical algorithms without device-specific noise considerations.

Another aspect of the algorithm scaling is training data. In general, its size will scale exponentially with the number of data qubits. However, we would like to stress that this fact does not jeopardize our approach since we numerically obtain solutions (algorithm instances) only for a small number of data qubits. Those optimization

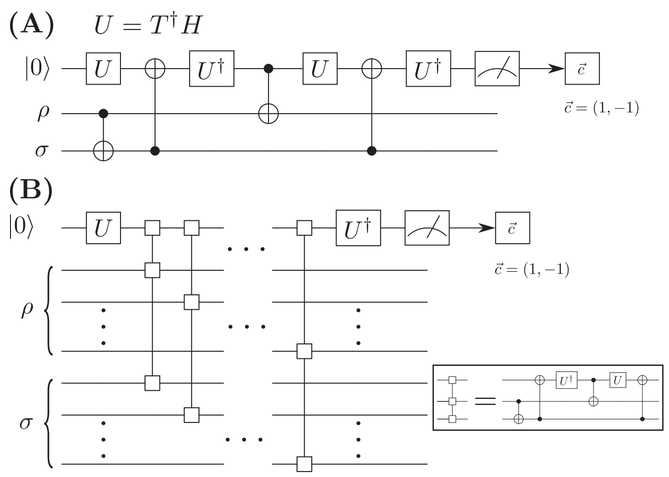

Figure 5. Our ABA, obtained by minimizing the cost for the resources shown in figures 3(A) and (B). (A) When ρ and σ are one-qubit states, we obtain a circuit with 4 CNOT gates and 4 one-qubit gates for a total of 8 gates. Here, $U = T^\dagger H$. (B) Six of these gates are combined to create a 'building block' (see inset) that is used to generalize the algorithm for input states ρ and σ of arbitrary size. The post-processing vector is $\vec{c} = (1, -1)$, independent of problem size.

problems require training data that is still manageable. Algorithm instances are then used to manually recognize the pattern and generalize to arbitrary system size.

Finally, the search space defined by $\vec{k}$ is exponentially large in the number of gates. This makes it impossible to systematically check all possibilities in the search for an optimal algorithm. On the other hand, the heuristic approach described above seems to be capable of finding the solution efficiently.

2.5. Generalization

For a fixed problem size, we minimize the cost. If the cost goes to zero (which we define as a cost less than 10^{-6}), we say we have an *algorithm instance*. In particular, this corresponds to fixing the size of the data and hence fixing n_D, the number of data qubits. To study the generalization of the algorithm, we grow the size of the problem by increasing n_D. In some cases, one may also need to increase the number of ancilla qubits, n_A, and/or the number of measurements in order to minimize the cost.

This gives us a set of algorithm instances for various problem sizes. An important challenge is to abstract a general algorithm from these instances. This challenge is particularly difficult because one can typically only find algorithm instances for small problem sizes. This is due to the fact that the search space for vectors $\vec{k}$ grows rapidly with problem size, namely as n_T^{2d}, where $n_T = n_D + n_A$ is the total number of qubits and d is the circuit gate count.

In this work, we were able to manually recognize the pattern by which the algorithm generalizes to arbitrary problem size by inspecting the various algorithm instances. In future work, we will explore automated methods for recognizing the general algorithm.

3. Main results

3.1. Overview

Our main results are short-depth algorithms for quantifying overlap on idealized quantum computing hardware. For the latter, we consider full connectivity, and we allow for arbitrary one-qubit gates as well as CNOT gates between all of the qubits.

We consider two sets of resources. The first set of resources are identical to those allowed for the Swap Test, i.e. access to one ancilla qubit and two data qubits, as well as one measurement on the ancilla qubit. The cost versus number of gates for these resources is shown in figure 3(A), and we obtained essentially zero cost for $d = 8$. To understand how the algorithm generalizes, we increase the number of qubits in ρ and σ to $n = 2$, giving a minimum gate count of $d = 14$, as shown in figure 3(B). As discussed below this generalizes to an algorithm (shown in figure 5) that we refer to as our ABA.

The second set of resources we consider allows for measurements on all of the qubits. For these additional resources, figure 3(C) shows that zero cost is obtained for $d = 2$. To recognize how this algorithm generalizes, we increase the number of qubits to $n = 2$, giving a minimum gate count of $d = 4$, as shown in figure 3(D). The surprising result is that the ancilla qubit is not used at all in this algorithm, even though we train the algorithm in

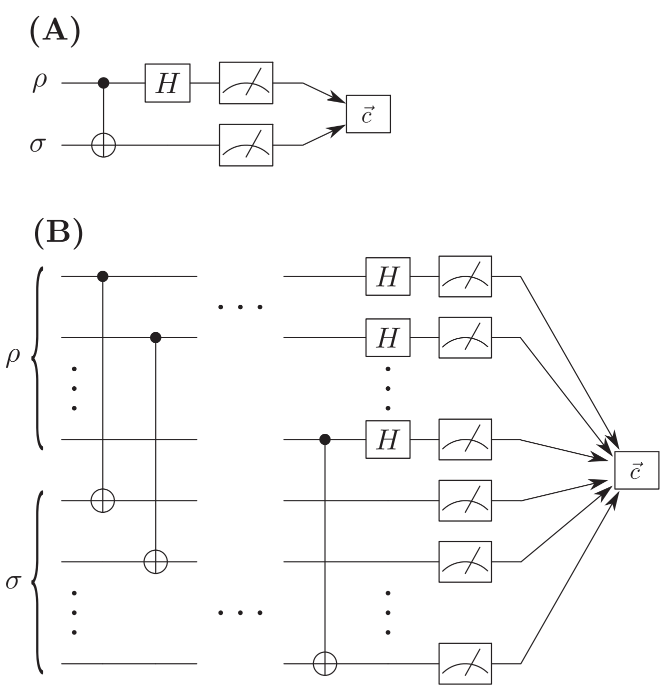

Figure 6. Our Bell-basis algorithm, obtained by minimizing the cost for the resources shown in figures 3(C) and (D). (A) When ρ and σ are one-qubit states, we obtain a circuit with one CNOT followed by a Hadamard and measurements on both qubits with a post-processing vector $\vec{c} = (1,\ 1,\ 1,\ -1)$. (B) The CNOT and Hadamard gates form a 'building block' that is used to generalize the algorithm for input states ρ and σ of arbitrary size. Since these gates can be parallelized, the quantum circuit depth is independent of problem size. On the other hand, the complexity of classical post-processing grows linearly with n, and the post-processing vector can be written as $\vec{c} = (1,\ 1,\ 1,\ -1)^{\otimes n}$ if one orders the qubits into pairs from ρ and σ.

the presence of an ancilla. This allows us to display the resulting general algorithm, our BBA, in figure 6 without the ancilla qubit.

In both cases discussed above, we managed to discover the general (valid for arbitrary problem size) form of the algorithm from its two smallest instances. We expect that in other applications, the general form of the algorithm may be harder to find and more sophisticated tools will have to be developed.

3.2. Ancilla-based algorithm

Figure 5(A) shows the ABA for one-qubit states ρ and σ. The unitary U in this circuit is $U = T^{\dagger}H$. This circuit employs 4 CNOT gates and 4 one-qubit gates for a total of 8 gates. It uses a simple post-processing vector $\vec{c} = (1,\ -1)$ that amounts to measuring the Pauli Z operator on the ancilla qubit, which is the same observable measured in the Swap Test. Not only does this circuit have a lower gate count than typical implementations of the Swap Test (see e.g. the circuit in figure 1(B)), but actually it implements a completely different unitary.

Let S_{ABA} denote the Schmidt rank (across the cut between ancilla and the data qubits) of the unitary G_{ABA} associated with the ABA gate sequence. It can be verified that $S_{\mathrm{ABA}} = 3$. This means that G_{ABA} is not locally equivalent to a controlled-SWAP, whose analogously defined Schmidt rank is 2. Thus, the ABA is fundamentally different from the Swap Test: it cannot be obtained from the Swap Test by local operations.

The general form of the ABA is given in figure 5(B). There is a repeating unit, shown in the inset of the figure, that is applied on each pair of qubits composing ρ and σ as well as on the ancilla qubit. This unit has 4 CNOT gates, so the overall algorithm employs $4n$ CNOT gates and $6n + 2$ total gates. Hence, the gate count grows linearly with the number of data qubits.

3.3. Bell-basis algorithm

Figure 6(A) shows the BBA for one-qubit states ρ and σ. This circuit employs one CNOT gate followed by one Hadamard gate, with both qubits being measured. It is straightforward to show that this corresponds to a Bell basis measurement. The post-processing is a bit more complicated, with $\vec{c} = (1,\ 1,\ 1,\ -1)$, which corresponds to summing the probabilities for the 00, 01, and 10 outcomes and subtracting probability of the 11 outcome. The above post-processing is equivalent to measuring the expectation value of a controlled-Z operator.

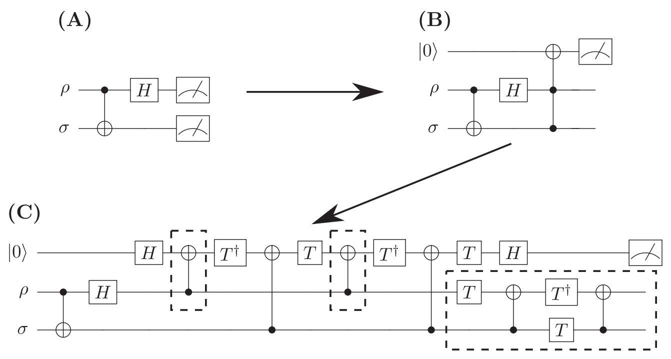

Figure 7. Equivalence between our ABA and BBA. The two-qubit measurement and classical post-processing in the BBA can be converted to a Toffoli gate with an ancilla as the target followed by a measurement on the ancilla. This takes us from circuit (A) to circuit (B). Inserting into circuit (B) the optimal decomposition of the Toffoli gate from [35] gives circuit (C). Finally one does three simplifications of this circuit to obtain the ABA, indicated by the dashed boxes in (C). Namely, the first boxed CNOT in (C) has trivial action and hence can be removed. The second boxed CNOT in (C) can be flipped such that the ancilla is the control qubit, which introduces some Hadamards. One of these Hadamards cancels with the first Hadamard in (C), and two others combine with T and $T^\dagger$ to make the $U^\dagger$ and U shown in figure 5(A). Finally the five gates enclosed in the last dashed box in (C) have no effect on the measurement and hence can be removed.

The generalization of this algorithm is given in figure 6(B). The repeating unit is simply a CNOT and Hadamard, applied on each pair of qubits composing ρ and σ. Furthermore, every qubit is measured at the output. The total number of gates is simply $2n$, and hence grows linearly with the number of qubits. However, more importantly, the CNOT and Hadamard on each qubit pair can be performed in parallel. This crucial fact means that this algorithm has a constant depth, independent of problem size. Namely, the depth is two quantum gates.

On the other hand, the classical post-processing is somewhat complicated, and its complexity scales linearly with the problem size. Namely, the post-processing vector can be written as $\vec{c} = (1,\ 1,\ 1,\ -1)^{\otimes n}$, provided that one arranges the qubits in the order $P_1Q_1P_2Q_2 \ldots. P_nQ_n$, where $P_1P_2 \ldots P_n$ and $Q_1Q_2 \ldots Q_n$ are the subsystems composing ρ and σ respectively. The linear scaling of post-processing follows from the fact that one does not explicitly compute $\vec{c} \cdot \vec{p}$ in equation (3). Rather one bins individual measurement outcomes into one of two bins (either the 1 or -1 bin). Here, the bin is determined by first assigning each of the n qubit pairs a value of 1 or -1, based on the associated eigenvalue of the controlled-Z operator, and then multiplying these n values. The overlap $\mathrm{Tr}(\rho\sigma)$ is then given by the weighted average over all outcomes, where the weights correspond to the bin label (either 1 or -1).

Nevertheless, for NTQCs, due to decoherence and gate infidelity, it is better for the classical post-processing to grow linearly in n than for the quantum circuit depth to grow linearly in n. Hence, the BBA seems to be the superior algorithm in that case.

3.4. Discussion

In 2013, Garcia-Escartin and Chamorro-Posada discovered the BBA for computing state overlap [24]. We were unaware of this important result until after our machine-learning approach found our BBA. More generally, it appears that the quantum computing community seems to be unaware of this article, perhaps because the article was presented in the language of quantum optics rather than that of quantum computing. Indeed, the ancilla-based version of the Swap Test, shown in figure 1, continues to be the algorithm employed in the quantum computing literature (e.g. see [27, 33]).

Although our two algorithms look very different, one can actually show a simple equivalence between our ABA and our BBA. One can see this by converting the classical post-processing in the BBA into a quantum gate. In particular, this gate would be a Toffoli gate, controlled by the two data qubits with the target being an ancilla qubit prepared in the $|0\rangle$ state. Appendix B shows proof of this statement. After inserting the Toffoli gate (see figure 7(B)), one would do a measurement of the Pauli Z observable on the ancilla to decode the state overlap. By replacing the Toffoli gate with its decomposition from [35] and simplifying the resulting circuit, one can then obtain our ABA (see figure 7(C)). In this sense, our ABA is essentially our BBA but with the classical post-processing transformed into Toffoli gates and a measurement on the ancilla. This equivalence is shown in figure 7 for one-qubit states. The generalization to multi-qubit states is straightforward.

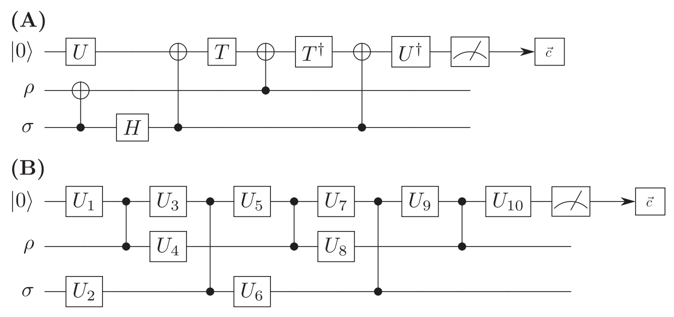

Figure 8. Ancilla-based algorithm adapted (via our machine-learning approach) to commercial hardware. (A) ABA adapted to IBM's five-qubit computer, $U = T^{\dagger} H$. (B) ABA adapted to Rigetti's 19-qubit computer. One-qubit unitaries have the following form: $U_1 = U_8 = H, U_2 = U_3 = U_6^{\dagger} = U_7^{\dagger} = XH, U_4 = R_X(-\pi/4)T, U_5 = T^{\dagger}HT, U_9 = R_X(\pi/4), U_{10} = R_X(-3\pi/4)$, where $R_X(\theta) = \mathrm{e}^{-\mathrm{i}\frac{\theta}{2}X}$.

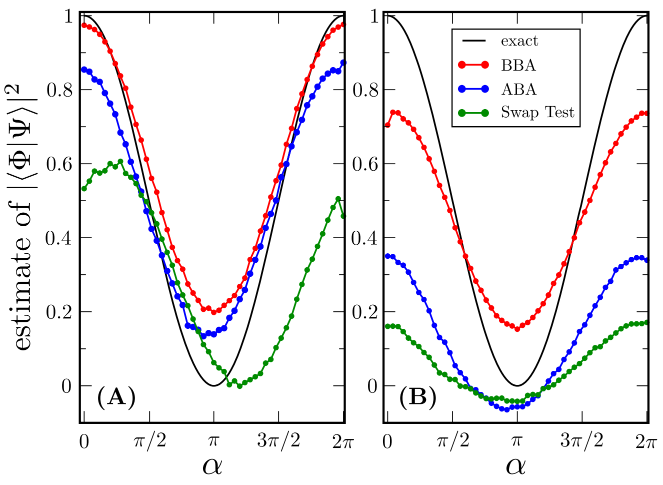

Figure 9. Experimentally observed overlaps on commercial hardware for the states $|\Psi\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and $|\Phi\rangle = (|0\rangle + \mathrm{e}^{\mathrm{i}\alpha}|1\rangle)/\sqrt{2}$. (A) Results from IBM's five-qubit computer called 'ibmqx4', with 49,152 quantum computer runs per data point. The black curve is the analytical overlap $|\langle\Phi|\Psi\rangle|^2$. The red, blue, and green curves are respectively the results for the BBA from figure 6(A), the ABA from figure 8(A), and the Swap Test from figure 2(B). (B) Results from Rigetti's 19-qubit computer, with 200,000 quantum computer runs per data point. The curves are analogous to those from panel (A). Namely, the red, blue, and green curves are respectively for the BBA from figure 6(A), the ABA from figure 8(B), and the Swap Test from figure 2(A) which Rigetti compiled to figure 2(C). The experimentally estimated overlap takes negative values for some α because the algorithm estimates the expectation value of controlled-Z operator, which has a negative eigenvalue. Another reason for this effect may be noise and other imperfections of the device.

4. Hardware-specific algorithms

Our BBA can be directly implemented on IBM's and Rigetti's quantum computers without any concern about connectivity issues (except for the minor issue that Rigetti uses controlled-Z instead of CNOT—their compiler easily makes the translation).

However, our ABA needs to be modified to account for IBM's and Rigetti's connectivity. While it is possible to manually modify the ABA to fit the connectivity, to illustrate our machine-learning approach, we numerically optimized the algorithm with the same resources as that shown in figure 3(A). The only difference is that we specified the gate set $\mathcal{A}$ to match the gate set (and hence the connectivity) of IBM's and Rigetti's computers.

The resulting algorithms that we obtained with our machine-learning approach are shown in figure 8. The ABA adapted to IBM's five-qubit computer only requires one additional gate, a Hadamard gate. The ABA adapted to Rigetti's 19-qubit computer requires an additional two-qubit gate and several additional one-qubit gates.

5. Testing our algorithms

We implemented our algorithms on IBM's five-qubit and Rigetti's 19-qubit computers. The resulting data are shown in figure 9. A caveat is that the different qubit counts for the two hardwares make it difficult to directly compare the results between these hardwares.

Table 1. Rms errors for the data shown in figure 9.

	IBM (5 qubits)	Rigetti (19 qubits)
Swap test	0.311	0.537
ABA	0.106	0.432
BBA	0.116	0.160

We considered two pure states of the form

$$|\Psi\rangle = (|0\rangle + |1\rangle)/\sqrt{2}, \tag{7}$$

$$|\Phi\rangle = (|0\rangle + e^{i\alpha}|1\rangle)/\sqrt{2}, \tag{8}$$

and we compared our results to the exact overlap $|\langle\Phi|\Psi\rangle|^2$ (black curve in figure 9). The rms errors are shown in table 1.

On both computers, the Swap Test (green curve in figure 9) performed poorly. It is noteworthy that these are only single-qubit states, and hence the results are expected to be even worse as one grows the size of these states.

Overall, our ABA performed significantly better than the Swap Test, while using the same resources, as is evident from the much smaller rms errors. The BBA, which allows for measurements on all qubits, dramatically outperformed the other algorithms on Rigetti's computer and performed roughly the same as ABA on IBM's computer. The relatively high accuracy of BBA is naturally expected due to its short depth, which mitigates the effects of decoherence and gate infidelity.

We note that there are values of the parameter α in equation (8) for which the Swap Test performs better than ABA and BBA, e.g. around $\alpha \approx \pi$. However, we believe that the rms error given in table 1 is a better indicator of algorithms' performance than the error at a particular value of α. To make this point, note that on a fully decohered (but otherwise perfect) hardware, the Swap Test is expected to return zero overlap independently of angle α. The algorithm would output the correct value for the overlap at $\alpha = \pi$ albeit for the wrong reason.

Our results show that connectivity between qubits as well as native gate set play important roles in the performance. Rigetti's 19 qubit computer offers less connectivity than IBM's five-qubit one. As a result, algorithms discovered for Rigetti's architecture are longer (compare circuits presented in figures 2 and 8) and overall perform worse. Algorithms found for IBM and Rigetti's computers suggest that for the particular problem of finding $\mathrm{Tr}(\rho\sigma)$, the ability to apply CNOT (rather that controlled-Z) results in shorter circuits. This can be seen from figure 8(B): several one-qubit gates can be eliminated by writing controlled-Z gates in terms of CNOTs.

6. Conclusions

This work shows that even well-known algorithms can be improved upon using an automated approach. As noted in the introduction, there are many applications that require state overlap computation, including the emerging new field of quantum machine-learning. While the Swap Test appears as a subroutine in many of these applications, we show that there are more efficient circuits to perform this subroutine.

We have found a constant depth algorithm (denoted BBA above) for computing state overlap, which is better than the linear scaling of the Swap Test. Furthermore, this algorithm performs better—with significantly lower error—even in the single-qubit case. It is therefore advisable that researchers use this algorithm henceforth for computing state overlap on NTQCs. This algorithm essentially corresponds to a measurement in the Bell basis for corresponding pairs of qubits. A key aspect of our approach that aided this algorithm's discovery was to allow for non-trivial classical post-processing, a strategy that has been used previously to shrink the depth of quantum algorithms [36]. The complexity of the post-processing for the BBA scales only linearly in the problem size (i.e. the number of qubits), ensuring that the quantum speedup that this algorithm provides is not due to the transfer of exponential complexity to the classical post-processing, but rather comes from the use of gates that can be executed in parallel.

Our main technical tool was a machine-learning method that allowed for task-oriented discovery of quantum algorithms. By task-oriented, we mean that this method defines a cost function based upon training data that are representative of the desired computation, i.e. the training data define the task. Minimizing the cost function results in a general algorithm for this computation. We emphasize that this goes far beyond quantum compiling since it allows for algorithm discovery when no algorithm is known.

Conceptually, our method separates quantum resources (ancillas, data qubits, and measurements) from algorithm parameters (gate sequence and classical post-processing). The former are fixed as hyperparameters while we optimize the latter. The algorithm's generalization is obtained by training for various problem sizes and recognizing the pattern. In future work, we plan to automate the process of pattern recognition for algorithm generalization.

As noted in [9], this field will be even more promising when quantum computers become available. This is due to the exponential speedup they provide in evaluating algorithm cost, i.e. by avoiding the exponential overhead of quantum simulation on classical computers. Indeed, some recent works propose to use quantum computers in automated algorithm learning [6, 7, 12]. Likewise our method can be extended to learning on a quantum computer by outsourcing cost evaluation to the quantum computer. This will be a topic of our future work.

Acknowledgments

The authors acknowledge helpful discussions with Francesco Caravelli. We thank Rigetti and IBM for providing access to their quantum computers. The views expressed in this paper are those of the authors and do not reflect those of Rigetti or IBM. LC was supported by the US Department of Energy through the J Robert Oppenheimer fellowship. YS acknowledges support of the LDRD program at Los Alamos National Laboratory (LANL). ATS and PJC were supported by the LANL ASC Beyond Moore's Law project.

Appendix A. Implementation details

This appendix gives details on the implementation of the Swap Test on Rigetti's 19-qubit quantum computer. The circuit, shown in figure A1, was generated by Rigetti's compiler. It consists of 22 one-qubit gates decomposed into rotations $R_Z(\alpha) = \mathrm{e}^{-\mathrm{i}\frac{\alpha}{2}Z}$ and pulses $S = \mathrm{e}^{-\mathrm{i}\frac{\pi}{4}X}$ as follows:

$$\begin{aligned}
U_1 = U_2 &= SR_Z(-3\pi/4)S,\\
U_3 &= SR_Z(-\pi/2),\\
U_4 &= S^\dagger R_Z(\pi/4)SR_Z(\pi/2),\\
U_5 &= SR_Z(\alpha_1)SR_Z(-\pi/2),\\
U_6 &= S^\dagger R_Z(\alpha_2)SR_Z(3\pi/4),\\
U_7 &= S^\dagger R_Z(-\pi/2),\\
U_8 = U_9 = U_{12} &= U_{14}^\dagger = U_{18}^\dagger = U_{21}^\dagger = S,\\
U_{10} &= S^\dagger R_Z(\pi/4)SR_Z(-\pi/2),\\
U_{11} &= S^\dagger R_Z(\alpha_3)S,\\
U_{13} &= S^\dagger R_Z(\alpha_4),\\
U_{15} &= SR_Z(\pi/4)S^\dagger R_Z(\pi),\\
U_{16} &= SR_Z(-3\pi/4)SR_Z(\pi/2),\\
U_{17} &= SR_Z(-\pi/4),\\
U_{19} = U_{20} &= SR_Z(\pi),\\
U_{22} &= R_Z(-\pi/2)SR_Z(\pi/4),
\end{aligned} \tag{A1}$$

where $\alpha_1 \simeq -0.6544\pi$, $\alpha_2 \simeq 0.7857\pi$, $\alpha_3 \simeq 0.1544\pi$ and $\alpha_4 \simeq 0.2143\pi$.

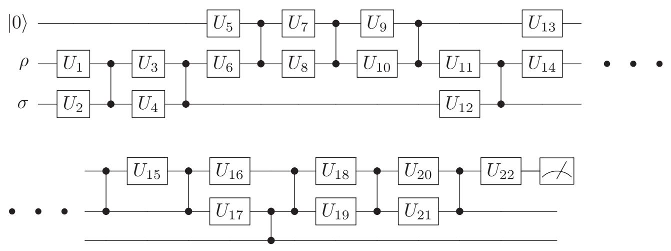
$|0\rangle$
ρ
σ
U_1 U_2 U_3 U_4 U_5 U_6 U_7 U_8 U_9 U_{10} U_{11} U_{12} U_{13} U_{14}
U_{15} U_{16} U_{17} U_{18} U_{19} U_{20} U_{21} U_{22}

Figure A1. Swap Test circuit obtained from Rigetti's compiler for their 19-qubit quantum computer. The specific form of all one-qubit gates is given by equation (A1).

Appendix B. Equivalence between ABA and BBA

Here we show that the post-processing in the BBA is equivalent to inserting a sequence of Toffoli gates followed by a measurement of Pauli Z operator as shown in figure B1. The rest of the proof of equivalence between ABA and BBA is presented in section 3.4 for one-qubit input states. Generalization to multi-qubit input states is straightforward as Toffoli gates in figure B1 are controlled by different qubits.

Let $\mathrm{CZ}_{j,k}$ denote controlled-Z gate acting on qubits j and k. Note that CZ is symmetric—the roles of control and target qubits can be exchanged. Post-processing employed in BBA is equivalent to measuring the expectation value of a product of CZ gates. The outcome of BBA is thus given by

$$\mathrm{Tr}\left[\rho \prod_{k=1}^{N} \mathrm{CZ}_{N+k,k}\right], \tag{B1}$$

where ρ is $2N$-qubit density matrix describing the state of BBA just before the measurement, see figure B1(A). We will show that this quantity is equal to the outcome of the algorithm that is obtained from BBA by replacing measurement on all qubits and subsequent post-processing with a collection of Toffoli gates followed by measurement on the ancilla qubit, as shown in figure B1(B). The outcome of that algorithm is given by

$$\mathrm{Tr}\left[\prod_{k=1}^{N} T_{N+k,k,0}(|0\rangle\langle 0| \otimes \rho)\prod_{k=1}^{N} T_{N+k,k,0} Z_0\right], \tag{B2}$$

where $T_{j,k,0}$ denotes Toffoli gate acting on qubits j, k, 0 with j, k being control qubits and 0 is the target qubit. Z_0 denotes Pauli Z operator acting on qubit 0. The expression in (B2) can be transformed as follows

$$\begin{aligned}
&\mathrm{Tr}\left[(|0\rangle\langle 0| \otimes \rho)\prod_{k=1}^{N} T_{N+k,k,0}\, Z_0 \prod_{k=1}^{N} T_{N+k,k,0}\right]\\
&= \mathrm{Tr}\left[(|0\rangle\langle 0| \otimes \rho)\mathrm{CZ}_{1,N+1}\prod_{k=2}^{N} T_{N+k,k,0}\, Z_0 \prod_{k=2}^{N} T_{N+k,k,0}\right] = \dots\\
&= \mathrm{Tr}\left[(|0\rangle\langle 0| \otimes \rho) Z_0 \prod_{k=1}^{N} \mathrm{CZ}_{N+k,k}\right]\\
&= \mathrm{Tr}\left[\rho \prod_{k=1}^{N} \mathrm{CZ}_{N+k,k}\right],
\end{aligned} \tag{B3}$$

where we used the fact that $T_{k,j,0}$ commutes with $T_{k',j',0}$, as well as $\mathrm{CZ}_{k',j'}$. We also used the following gate equivalence

$$T_{k,j,0}\, Z_0\, T_{k,j,0} = Z_0 \mathrm{CZ}_{k,j}. \tag{B4}$$

The last line in equation (B3) establishes the equivalence.

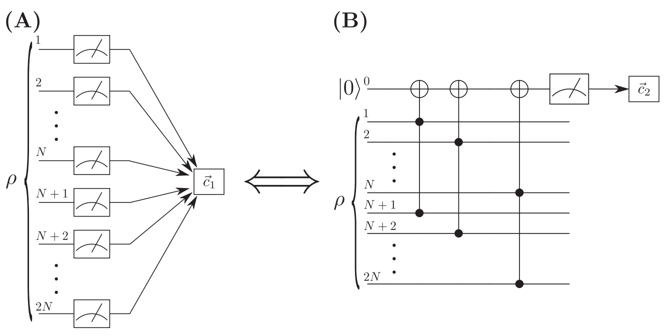

Figure B1. Post-processing that is used in BBA (panel (A)) is equivalent to the sequence of Toffoli gates followed by a measurement of the Pauli Z operator on ancilla qubit shown in panel (B). Here the post-processing vectors are $\vec{c}_2 = (1,\ -1)$ and $\vec{c}_1 = (1,\ 1,\ 1,\ -1)^{\otimes n}$, assuming qubits are arranged in order $1, N+1, 2, N+2,\ \dots,\ N, 2N$.

References

[1] Preskill J 2012 Quantum computing and the entanglement frontier arXiv:1203.5813
[2] Neill C *et al* 2017 A blueprint for demonstrating quantum supremacy with superconducting qubits *Science* **360** 195–9
[3] Ball P 2018 The era of quantum computing is here. Outlook: cloudy *Quanta Mag.* [available at: http://quantamagazine.org/the-era-of-quantum-computing-is-here-outlook-cloudy-20180124/]
[4] Fowler A G, Mariantoni M, Martinis J M and Cleland A N 2012 Surface codes: towards practical large-scale quantum computation *Phys. Rev.* A **86** 032324
[5] You H *et al* 2013 Simulating the transverse Ising model on a quantum computer: error correction with the surface code *Phys. Rev.* A **87** 032341
[6] Benedetti M, Garcia-Pintos D, Nam Y and Perdomo-Ortiz A 2018 A generative modeling approach for benchmarking and training shallow quantum circuits arXiv:1801.07686
[7] Mitarai K, Negoro M, Kitagawa M and Fujii K 2018 Quantum circuit learning *Phys. Rev.* A **98** 032309
[8] Khaneja N, Reiss T, Kehlet C, Schulte-Herbrüggen T and Glaser S J 2005 Optimal control of coupled spin dynamics: design of NMR pulse sequences by gradient ascent algorithms *J. Magn. Reson.* **172** 296–305
[9] Gepp A and Stocks P 2009 A review of procedures to evolve quantum algorithms *Genetic Program. Evolvable Mach.* **10** 181–228
[10] Lukac M *et al* 2003 Evolutionary approach to quantum and reversible circuits synthesis *Artif. Intell. Rev.* **20** 361–417
[11] Venturelli D, Do M, Rieffel E and Frank J 2018 Compiling quantum circuits to realistic hardware architectures using temporal planners *Quantum Sci. Technol.* **3** 025004
[12] Khatri S *et al* 2018 Quantum-assisted quantum compiling arXiv:1807.00800
[13] Häner T, Steiger D S, Svore K and Troyer M 2018 A software methodology for compiling quantum programs *Quantum Sci. Technol.* **3** 020501
[14] Maslov D 2017 Basic circuit compilation techniques for an ion-trap quantum machine *New J. Phys.* **19** 023035
[15] Martinez E A, Monz T, Nigg D, Schindler P and Blatt R 2016 Compiling quantum algorithms for architectures with multi-qubit gates *New J. Phys.* **18** 063029
[16] Chong F T, Franklin D and Martonosi M 2017 Programming languages and compiler design for realistic quantum hardware *Nature* **549** 180
[17] Swaddle M, Noakes L, Smallbone H, Salter L and Wang J 2017 Generating three-qubit quantum circuits with neural networks *Phys. Lett.* A **381** 3391–5
[18] Zahedinejad E, Ghosh J and Sanders B C 2016 Designing high-fidelity single-shot three-qubit gates: a machine-learning approach *Phys. Rev. Appl.* **6** 054005
[19] Las Heras U, Alvarez-Rodriguez U, Solano E and Sanz M 2016 Genetic algorithms for digital quantum simulations *Phys. Rev. Lett.* **116** 230504
[20] Hachtel G D and Somenzi F 1996 *Logic Synthesis and Verification Algorithms* (Boston, MA: Kluwer)
[21] Haaswijk W *et al* 2018 Deep learning for logic optimization algorithms *2018 IEEE Int. Symp. on Circuits and Systems (ISCAS)* (Piscataway, NJ: IEEE) pp 1–4
[22] Buhrman H, Cleve R, Watrous J and De Wolf R 2001 Quantum fingerprinting *Phys. Rev. Lett.* **87** 167902
[23] Gottesman D and Chuang I 2001 Quantum digital signatures arXiv:quant-ph/0105032
[24] Garcia-Escartin J C and Chamorro-Posada P 2013 Swap test and Hong-Ou-Mandel effect are equivalent *Phys. Rev.* A **87** 052330
[25] Patel R B, Ho J, Ferreyrol F, Ralph T C and Pryde G J 2016 A quantum Fredkin gate *Sci. adv.* **2** e1501531
[26] Ferreyrol F, Ralph T C and Pryde G J 2013 Implementation of a quantum Fredkin gate using an entanglement resource *2013 Conf. on Lasers and Electro-Optics Europe and International Quantum Electronics Conf. (CLEO EUROPE/IQEC)* p 1
[27] Linke N M *et al* 2017 Measuring the Renyi entropy of a two-site Fermi-Hubbard model on a trapped ion quantum computer arXiv:1712.08581
[28] Smith R S, Curtis M J and Zeng W J 2016 A practical quantum instruction set architecture arXiv:1608.03355
[29] Cross A W, Bishop L S, Smolin J A and Gambetta J M 2017 Open quantum assembly language arXiv:1707.03429
[30] Lloyd S, Mohseni M and Rebentrost P 2013 Quantum algorithms for supervised and unsupervised machine learning arXiv:1307.0411
[31] Wiebe N, Kapoor A and Svore K 2014 Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning *Quantum Inf. Comput.* **15** 0318–58
[32] Rebentrost P, Mohseni M and Lloyd S 2014 Quantum support vector machine for big data classification *Phys. Rev. Lett.* **113** 130503
[33] Johri S, Steiger D S and Troyer M 2017 Entanglement spectroscopy on a quantum computer *Phys. Rev.* B **96** 195136
[34] Smolin J A and DiVincenzo D P 1996 Five two-bit quantum gates are sufficient to implement the quantum Fredkin gate *Phys. Rev.* A **53** 2855
[35] Shende V V and Markov I L 2009 On the CNOT-cost of Toffoli gates *Quantum Inf. Comput.* **9** 0461–86
[36] Svore K M, Hastings M B and Freedman M 2014 Faster phase estimation *Quantum Inf. Comput.* **14** 306–28