Improved Techniques for Preparing Eigenstates of Fermionic Hamiltonians

Dominic W. Berry,[1] Mária Kieferová,[1,2] Artur Scherer,[1] Yuval R. Sanders,[1]
Guang Hao Low,[3] Nathan Wiebe,[3] Craig Gidney,[4] and Ryan Babbush[5,*]

[1] *Department of Physics and Astronomy, Macquarie University, Sydney, NSW 2109, Australia*
[2] *Institute for Quantum Computing and Department of Physics and Astronomy, University of Waterloo, Waterloo, ON N2L 3G1, Canada*
[3] *Microsoft Research, Redmond, WA 98052, United States of America*
[4] *Google Inc., Santa Barbara, CA 93117, United States of America*
[5] *Google Inc., Venice, CA 90291, United States of America*

(Dated: December 22, 2017)

Modeling low energy eigenstates of fermionic systems can provide insight into chemical reactions and material properties and is one of the most anticipated applications of quantum computing. We present three techniques for reducing the cost of preparing fermionic Hamiltonian eigenstates using phase estimation. First, we report a polylogarithmic-depth quantum algorithm for antisymmetrizing the initial states required for simulation of fermions in first quantization. This is an exponential improvement over the previous state-of-the-art. Next, we show how to reduce the overhead due to repeated state preparation in phase estimation when the goal is to prepare the ground state to high precision and one has knowledge of an upper bound on the ground state energy that is less than the excited state energy (often the case in quantum chemistry). Finally, we explain how one can perform the time evolution necessary for the phase estimation based preparation of Hamiltonian eigenstates with exactly zero error by using the recently introduced qubitization procedure.

INTRODUCTION

One of the most important applications of quantum simulation (and of quantum computing in general) is the Hamiltonian simulation based solution of the electronic structure problem. The ability to accurately model ground states of fermionic systems would have significant implications for many areas of chemistry and materials science and could enable the in silico design of new solar cells, batteries, catalysts, pharmaceuticals, etc. [1, 2]. The most rigorous approaches to solving this problem involve using the quantum phase estimation algorithm [3] to project to molecular ground states starting from a classically guessed state [4]. Beyond applications in chemistry, one might want to prepare fermionic eigenstates in order to simulate quantum materials [5] including models of high-temperature superconductivity [6].

In the procedure introduced by Abrams and Lloyd [7], one first initializes the system in some efficient-to-prepare initial state $|\varphi\rangle$ which has appreciable support on the desired eigenstate $|k\rangle$ of Hamiltonian H. One then uses quantum simulation to construct a unitary operator that approximates time evolution under H. With these ingredients, standard phase estimation techniques invoke controlled application of powers of $U(\tau) = e^{-iH\tau}$. With probability $\alpha_k = |\langle\varphi|k\rangle|^2$, the output is then an estimate of the corresponding eigenvalue E_k with standard deviation $\sigma_{E_k} = O((\tau M)^{-1})$, where M is the total number of applications of $U(\tau)$. The synthesis of $e^{-iH\tau}$ is typically performed using digital quantum simulation algorithms, such as by Lie-Trotter product formulas [8], truncated Taylor series [9], or quantum signal processing [10].

Since the proposal by Abrams and Lloyd [7], algorithms for time-evolving fermionic systems have improved substantially [11–17]. Innovations that are particularly relevant to this paper include the use of first quantization to reduce spatial overhead [18–20] from $O(N)$ to $O(\eta \log N)$ where η is number of particles and $N \gg \eta$ is number of single-particle basis functions (e.g. molecular orbitals or plane waves), and the use of post-Trotter methods to reduce the scaling with time-evolution error from $O(\mathrm{poly}(1/\epsilon))$ to $O(\mathrm{polylog}(1/\epsilon))$ [18, 21, 22]. The algorithm of [18] makes use of both of these techniques to enable the most efficient first quantized quantum simulation of electronic structure in the literature.

Unlike second quantized simulations which necessarily scale polynomially in N, first quantized simulation offers the possibility of achieving total gate complexity $O(\mathrm{poly}(\eta)\,\mathrm{polylog}(N, 1/\epsilon))$. This is important because the convergence of basis set discretization error is limited by resolution of the electron-electron cusp [23], which cannot be resolved faster than $O(1/N)$ using any single-particle basis expansion. Thus, whereas the cost of refining second quantized simulations to within δ of the continuum basis limit is necessarily $O(\mathrm{poly}(1/\delta))$, first quantization offers the possibility of suppressing basis set errors as $O(\mathrm{polylog}(1/\delta))$, providing essentially arbitrarily precise representations.

In second quantized simulations of fermions the wavefunction encodes an antisymmetric fermionic system, but the qubit representation of that wavefunction is not necessarily antisymmetric. Thus, in second quantization it is necessary that operators act on the encoded wavefunction in a way that enforces the proper exchange statistics. This is the purpose of second quantized fermion mappings such as those explored in [24–30]. By contrast, the distinguishing feature of first quantized simulations

* Corresponding author: babbush@google.com

is that the antisymmetry of the encoded system must be enforced directly in the qubit representation of the wavefunction. This often simplifies the task of Hamiltonian simulation but complicates the initial state preparation.

In first quantization there are typically η different registers of size $\log N$ (where η is the number of particles and N is number of spin-orbitals) encoding integers indicating the indices of occupied orbitals. As only η of the N orbitals are occupied, with $\eta \log N$ qubits one can specify an arbitrary configuration. To perform simulations in first quantization, one typically requires that the initial state $|\varphi\rangle$ is antisymmetric under the exchange of any two of the η registers. Prior work presented a procedure for preparing such antisymmetric states with complexity stated to be $\widetilde{O}(\eta^2)$, though there is a step that appears to scale as $\widetilde{O}(\eta^3)$ (see Appendix A) [31, 32].

In Section I we provide a general approach for antisymmetrizing states via sorting networks. The circuit size is $O(\eta \log^c \eta \log N)$ and the depth is $O(\log^c \eta \log\log N)$, where the value of $c \geq 1$ depends on the choice of sorting network (it can be 1, albeit with a large multiplying factor). In terms of the circuit depth, these results improve exponentially over prior implementations [31, 32]. They also improve polynomially on the total number of gates needed. We also discuss an alternative approach, a quantum variant of the Fisher-Yates shuffle, which avoids sorting, and achieves a size-complexity of $O(\eta^2 \log N)$ with lower spatial overhead than the sort-based methods.

Once the initial state $|\varphi\rangle$ has been prepared, it typically will not be exactly the ground state desired. In the usual approach, one would perform phase estimation repeatedly until the ground state is obtained, giving an overhead scaling inversely with the initial state overlap. In Section II we propose a strategy for reducing this cost, by initially performing the estimation with only enough precision to eliminate excited states.

In Section III we explain how qubitization [33] provides a unitary sufficient for phase estimation purposes with exactly zero error (provided a gate set consisting of an entangling gate and arbitrary single-qubit rotations). This improves over proposals to perform the time evolution unitary with post-Trotter methods at cost scaling as $O(\text{polylog}(1/\epsilon))$. We expect that a combination of these strategies will enable quantum simulations of fermions similar to the proposal of [18] with substantially fewer T gates than any method suggested in prior literature.

I. EXPONENTIALLY FASTER ANTISYMMETRIZATION

Here we present our algorithm for imposing fermionic exchange symmetry on a sorted, repetition-free quantum array `target`. Specifically, the result of this procedure is to perform the transformation

$$|r_1 \cdots r_\eta\rangle \mapsto \sum_{\sigma \in S_\eta} (-1)^{\pi(\sigma)} |\sigma(r_1, \cdots, r_\eta)\rangle \qquad (1)$$

where $\pi(\sigma)$ is the parity of the permutation σ, and we require for the initial state that $r_p < r_{p+1}$ (necessary for this procedure to be unitary). Although we describe the procedure for a single input $|r_1 \cdots r_\eta\rangle$, our algorithm may be applied to any superposition of such states.

Our approach is a modification of that proposed in Ref. [31]; namely, to apply the reverse of a sort to a sorted quantum array. Whereas Ref. [31] claims a gate count of $O(\eta^2 \log N)$, we can report a gate count of $O(\eta \log \eta \log N)$ and a runtime of $O(\log \eta \log\log N)$.

This section proceeds as follows. We begin with a summary of our algorithm. We then explain the reasoning underlying the key step (Step 4) of our algorithm, which is to reverse a sorting operation on `target`. Next we discuss the choice of sorting algorithm, which we require to be a sorting network. Then, we assess the cost of our algorithm in terms of gate complexity and runtime and we compare this to previous work in Ref. [31]. Finally, we discuss the possibility of antisymmetrizing without sorting and propose an alternative, though more costly, algorithm based on the Fisher-Yates shuffle. Our algorithm consists of the following four steps:

1. **Prepare `seed`.** Let f be a function chosen so that $f(\eta) \geq \eta^2$ for all η. We prepare an ancillary register called `seed` in an even superposition of all possible length-η strings of the numbers $0, 1, \ldots, f(\eta) - 1$. If $f(\eta)$ is a power of two, preparing `seed` is easy: simply apply a Hadamard gate to each qubit.

2. **Sort `seed`.** Apply a reversible sorting network to `seed`. Any sorting network can be made reversible by storing the outcome of each comparator in a second ancillary register called `record`. There are several known sorting networks with polylogarithmic runtime, as we discuss below.

3. **Delete collisions from `seed`.** As `seed` was prepared in a superposition of all length-η strings, it includes strings with repeated entries. As we are imposing fermionic exchange symmetry, these repetitions must be deleted. We therefore measure `seed` to determine whether a repetition is present, and we accept the result if it is repetition-free. We prove in Appendix B that choosing $f(\eta) \geq \eta^2$ ensures that the probability of success is greater than 1/2. We further prove that the resulting state of `seed` is disentangled from `record`, meaning `seed` can be discarded after this step.

4. **Apply the reverse of the sort to `target`.** Using the comparator values stored in `record`, we apply each step of the sorting network in reverse order to the sorted array `target`. The resulting state of `target` is an evenly weighted superposition of each possible permutation of the original values. To ensure the correct phase, we apply a controlled-phase gate after each swap.

Step 4 is the key step. Having prepared (in Step 1-Step 3) a record of the in-place swaps needed to sort a symmetrized, collision-free array, we undo each of these swaps in turn on the sorted `target`. We employ a sorting network, a restricted type of sorting algorithm, because sorting networks have comparisons and swaps at a fixed sequence of locations. By contrast, many common classical sorting algorithms (like heapsort) choose locations depending on the values in the list. This results in accessing registers in a superposition of locations in the corresponding quantum algorithm, incurring a linear overhead. As a result, a quantum heapsort requires $\widetilde{O}\left(\eta^2\right)$ operations, not $\widetilde{O}(\eta)$. By contrast, no overhead is required for using a fixed sequance of locations.

Our algorithm allows for any choice of sorting network. Two useful choices are the odd-even mergesort [34] and the bitonic sort [34, 35]. These both have complexity $O(\eta \log^2 \eta)$, though the odd-even mergesort is slightly more efficient. These algorithms are also highly parallelizable, and have depth only $O(\log^2 \eta)$. The asymptotically best sorting networks have depth $O(\log \eta)$ and complexity $O(\eta \log \eta)$, though there is a large constant which means they are less efficient for realistic η [36, 37]. There is also a sorting network with $O(\eta \log \eta)$ complexity with a better multiplicative constant [38], though its depth is $O(\eta \log \eta)$ (so it is not logarithmic).

We now briefly explain how to make a sorting network reversible, as is necessary for Step 2. A sorting network is a type of comparator network, meaning a circuit constructed entirely out of primitive operations called comparators. A comparator in the non-reversible classical sense accepts the input (a, b) and returns $(\min\{a, b\}, \max\{a, b\})$. We explain how to implement a reversible, hence quantum, comparator in Appendix C. A reversible sorting network is constructed from reversible comparators instead of the standard kind. The implementation of sorting networks in quantum algorithms has previously been considered in Refs. [39, 40].

Assuming we use an asymptotically optimal sorting network, the circuit depth for our algorithm is $O(\log \eta \log\log N)$ and the gate complexity is $O(\eta \log \eta \log N)$. The dominant cost of the algorithm comes from Step 2 and Step 4, each of which have $O(\eta \log \eta)$ comparators that can be parallelized to ensure the sorting network executes only $O(\log \eta)$ comparator rounds. Each comparator for Step 4 has a complexity of $O(\log N)$ and a depth of $O(\log\log N)$, as we show in Appendix C. The comparators for Step 2 have complexity $O(\log \eta)$ and depth $O(\log\log \eta)$, which is less because $\eta < N$. Thus Step 2 and Step 4 each have gate complexity $O(\eta \log \eta \log N)$ and runtime $O(\log \eta \log\log N)$.

The other two steps in our algorithm have smaller cost. Step 1 has constant depth and $O(\eta \log \eta)$ complexity. Step 3 requires $O(\eta)$ comparisons because only nearest-neighbour comparisons need be carried out on `seed` after sorting. These comparisons can be parallelized over two rounds, with complexity $O(\eta \log \eta)$ and circuit depth $O(\log\log \eta)$. Then the result for any of the registers being equal is computed in a single qubit, which has complexity $O(\eta)$ and depth $O(\log \eta)$. Thus the complexity of Step 3 is $O(\eta \log \eta)$ and the total circuit depth is $O(\log \eta)$. We give further details in Appendix C. Thus, our algorithm has an exponential runtime improvement over the proposal in Ref. [31]. We also have a polynomial improvement in gate complexity, which is $\widetilde{O}(\eta)$ for our algorithm but $\widetilde{O}(\eta^3)$ for Ref. [31].

Our runtime is likely optimal for symmetrization, at least in terms of the η scaling. Symmetrization takes a single computational basis state and generates a superposition of $\eta!$ computational basis states. Each single-qubit operation can increase the number of states in the superposition by at most a factor of two, and two-qubit operations can increase the number of states in the superposition by at most a factor of four. Thus, the number of one- and two-qubit operations is at least $\log_2(\eta!) = O(\eta \log \eta)$. In our algorithm we need this number of operations between the registers. If that is true in general, then η operations can be parallelized, resulting in minimum depth $O(\log \eta)$. It is more easily seen that the total number of registers used is optimal. There are $O(\eta \log \eta)$ ancilla qubits due to the number of steps in the sort, but the number of qubits for the system state we wish to symmetrize is $O(\eta \log N)$, which is asymptotically larger.

Our quoted asymptotic runtime and gate complexity scalings assume the use of sorting networks that are asymptotically optimal. However, these algorithms have a large constant overhead making it more practical to use an odd-even mergesort, leading to depth $O(\log^2 \eta \log\log N)$. Note that is possible to obtain complexity $O(\eta \log \eta \log N)$ and number of ancilla qubits $O(\eta \log \eta)$ with a better scaling constant using the sorting network of Ref. [38].

Given that the cost of our algorithm is dictated by the cost of sorting algorithms, it is natural to ask if it is possible to antisymmetrize without sorting. Though the complexity and runtime both turn out to be significantly worse than our sort-based approach, we suggest an alternative antisymmetrization algorithm based on the Fisher-Yates shuffle. The Fisher-Yates shuffle is a method for applying to a length-η target array a permutation chosen uniformly at random using a number of operations scaling as $O(\eta)$. Our algorithm indexes the positions to be swapped, thereby increasing the complexity to $\widetilde{O}(\eta^2)$. Briefly put, our algorithm generates a superposition of states as in Step II of Ref. [31], then uses these as control registers to apply the Fisher-Yates shuffle to the orbital numbers. The complexity is $O(\eta^2 \log N)$, with a factor of $\log N$ due to the size of the registers. We reset the control registers, thereby disentangling them, using $O(\eta \log \eta)$ ancillae. We provide more details of this approach in Appendix D.

To conclude this section, we have presented an algorithm for antisymmetrizing a sorted, repetition-free quantum register. The dominant cost of our algorithm derives from the choice of sorting network, whose asymptotically optimal gate count complexity and runtime are,

respectively, $O(\eta \log \eta \log N)$ and $O(\log \eta \log \log N)$. This constitutes a polynomial improvement in the first case and exponential in the second case over previous work in Ref. [31]. As in Ref. [31], our antisymmetrization algorithm constitutes a key step for preparing fermionic wavefunctions in first quantization. viousl

II. FEWER PHASE ESTIMATION REPETITIONS BY PARTIAL EIGENSTATE PROJECTION REJECTION

Once the initial state $|\varphi\rangle$ has been prepared, it typically will not be exactly the ground state (or other eigenstate) desired. In the usual approach, one would perform phase estimation repeatedly, in order to obtain the desired eigenstate $|k\rangle$. The number of repetitions needed scales inversely in $\alpha_k = |\langle\varphi|k\rangle|^2$, increasing the complexity. We propose a practical strategy for reducing this cost which is particularly relevant for quantum chemistry. Our approach applies if one seeks to prepare the ground state with knowledge of an upper bound on the ground state energy $\tilde{E}_0$, together with the promise that $E_0 \leq \tilde{E}_0 < E_1$. With such bounds available, one can reduce costs by restarting the phase estimation procedure as soon as the energy is estimated to be above $\tilde{E}_0$ with high probability. That is, one can perform a phase estimation procedure that gradually provides estimates of the phase to greater and greater accuracy, for example as in Ref. [41]. If at any stage the phase is estimated to be above $\tilde{E}_0$ with high probability, then the initial state can be discarded and re-prepared.

Performing phase estimation within error ϵ typically requires evolution time for the Hamiltonian of $1/\epsilon$, leading to complexity scaling as $1/\epsilon$. This means that, if the state is the first excited state, then an estimation error less than $E_1 - \tilde{E}_0$ will be sufficient to show that the state is not the ground state. The complexity needed would then scale as $1/(E_1 - \tilde{E}_0)$. In many cases, the final error required, ϵ_f, will be considerably less than $E_1 - \tilde{E}_0$, so the majority of the contribution to the complexity comes from measuring the phase with full precision, rather than just rejecting the state as not the ground state.

Given the initial state $|\varphi\rangle$ which has initial overlap of α_0 with the ground state, if we restart every time the energy is found to be above $\tilde{E}_0$, then the contribution to the complexity is $1/[\alpha_0(E_1 - \tilde{E}_0)]$. There will be an additional contribution to the complexity of $1/\epsilon_f$ to obtain the estimate of the ground state energy with the desired accuracy, giving an overall scaling of the complexity of

$$O\left(\frac{1}{\alpha_0(E_1 - \tilde{E}_0)} + \frac{1}{\epsilon_f}\right). \tag{2}$$

In contrast, if one were to perform the phase estimation with full accuracy every time, then the scaling of the complexity would be $O(1/(\alpha_0\epsilon_f))$. Provided $\alpha_0(E_1 - \tilde{E}_0) > \epsilon_f$, the method we propose would essentially eliminate the overhead from α_0.

In cases where α_0 is very small, it would be helpful to apply amplitude amplification. A complication with amplitude amplification is that we would need to choose a particular initial accuracy to perform the estimation. If a lower bound on the excitation energy, $\tilde{E}_1$, is known, then we can choose the initial accuracy to be $\tilde{E}_1 - \tilde{E}_0$. The success case would then correspond to not finding that the energy is above $\tilde{E}_0$ after performing phase estimation with that precision. Then amplitude amplification can be performed in the usual way, and the overhead for the complexity is $1/\sqrt{\alpha_0}$ instead of $1/\alpha_0$.

All of this discussion is predicated on the assumption that there are cases where α_0 is small enough to warrant using phase estimation as part of the state preparation process and where a bound meeting the promises of $\tilde{E}_0$ is readily available. We now discuss why these conditions are anticipated for many problems in quantum chemistry. Most chemistry is understood in terms of mean-field models (e.g. molecular orbital theory, ligand field theory, the periodic table, etc.). Thus, the usual assumption (empirically confirmed for many smaller systems) is that the ground state has reasonable support on the Hartree-Fock state (the typical choice for $|\varphi\rangle$) [42–45]. However, this overlap will decrease as a function of both basis size and system size. As a simple example, consider a large system composed of n copies of non-interacting subsystems. If the Hartree-Fock solution for the subsystem has overlap α_0, then the Hartree-Fock solution for the larger system has overlap of exactly α_0^n, which is exponentially small in n.

It is literally plain-to-see that the electronic ground state of molecules is often protected by a large gap. The color of many molecules and materials is the signature of an electronic excitation from the ground state to first excited state upon absorption of a photon in the visible range (around 0.7 Hartree); many clear organics have even larger gaps in the UV spectrum. Visible spectrum $E_1 - E_0$ gaps are roughly a hundred times larger than the typical target accuracy of $\epsilon_f = 0.0016$ Hartree ("chemical accuracy")[1]. Furthermore, in many cases the first excited state is perfectly orthogonal to the Hartree-Fock state for symmetry reasons (e.g. due to the ground state being a spin singlet and the excited state being a spin triplet). Thus, the gap of interest is really $E^* - E_0$ where $E^* = \min_{k>0} E_k$ subject to $|\langle\varphi|k\rangle|^2 > 0$. Often the $E^* - E_0$ gap is much larger than the $E_1 - E_0$ gap.

For most problems in quantum chemistry a variety of scalable classical methods are accurate enough to compute upper bounds on the ground state energy $\tilde{E}_0$ such that $E_0 \leq \tilde{E}_0 < E^*$, but not accurate enough to obtain chemical accuracy (which would require quantum

[1] The rates of chemical reactions are proportional to $e^{-\beta\Delta A}/\beta$ where β is inverse temperature and ΔA is a difference in free energy between reactants and the transition state separating reactants and products. Chemical accuracy is defined as the maximum error allowable in ΔA such that errors in the rate are smaller than a factor of ten at room temperature [4].

computers). Classical methods usually produce upper bounds when based on the variational principle. Examples include mean-field and Configuration Interaction Singles and Doubles (CISD) methods [46].

As a concrete example, consider a calculation on the water molecule in its equilibrium geometry (bond angle of 104.5°, bond length of 0.9584 Å) in the minimal (STO-3G) basis set performed using OpenFermion [47] and Psi4 [48]. For this system, $E_0 = -75.0104$ Hartree and $E_1 = -74.6836$ Hartree. However, $\langle\varphi|1\rangle = 0$ and $E^* = -74.3688$ Hartree. The classical mean-field energy provides an upper bound on the ground state energy of $\tilde{E}_0 = -74.9579$ Hartree. Therefore $E^* - \tilde{E}_0 \approx 0.6$ Hartree, which is about 370 times ϵ_f. Thus, using our strategy, for $\alpha_0 > 0.003$ there is very little overhead due to the initial state $|\psi\rangle$ not being the exact ground state. In the most extreme case for this example, that represents a speedup by a factor of more than two orders of magnitude. However, in some cases the ground state overlap might be high enough that this technique provides only a modest advantage. While the Hartree-Fock state overlap in this small basis example is $\alpha_0 = 0.972$, as the system size and basis size grow we expect this overlap will decrease (as argued earlier).

Another way to cause the overlap to decrease is to deviate from equilibrium geometries [42, 43]. For example, we consider this same system (water in the minimal basis) when we stretch the bond lengths to 2.25× their normal lengths. In this case, $E_0 = -74.7505$ Hartree, $E^* = -74.6394$ Hartree, and $\alpha_0 = 0.107$. The CISD solution provides an upper bound $\tilde{E}_0 = -74.7248$. In this case, $E^* - \tilde{E}_0 \approx 0.085$ Hartree, about 50 times ϵ_f. Since $\alpha_0 > 0.02$, here we speed up state preparation by roughly a factor of α_0^{-1} (more than an order of magnitude).

III. PHASE ESTIMATION UNITARIES WITHOUT APPROXIMATION

Normally, the phase estimation would be performed by Hamiltonian simulation. That introduces two difficulties: first, there is error introduced by the Hamiltonian simulation that needs to be taken into account in bounding the overall error, and second, there can be ambiguities in the phase that require simulation of the Hamiltonian over very short times to eliminate.

These problems can be eliminated if one were to use Hamiltonian simulation via a quantum walk, as in Refs. [49, 50]. There, steps of a quantum walk can be performed exactly, which have eigenvalues related to the eigenvalues of the Hamiltonian. Specifically, the eigenvalues are of the form $\pm e^{\pm i \arcsin(E_k/\lambda)}$. Instead of using Hamiltonian simulation, it is possible to simply perform phase estimation on the steps of that quantum walk, and invert the function to find the eigenvalues of the Hamiltonian. That eliminates any error due to Hamiltonian simulation. Moreover, the possible range of eigenvalues of the Hamiltonian is automatically limited, which eliminates the problem with ambiguities.

The quantum walk of Ref. [50] does not appear to be appropriate for quantum chemistry, because it requires an efficient method of calculating matrix entries of the Hamiltonian. That is not available for the Hamiltonians of quantum chemistry, but they can be expressed as sums of unitaries, as for example discussed in Ref. [21]. It turns out that the method called qubitization [33] allows one to take a Hamiltonian given by a sum of unitaries, and construct a new operation with exactly the same functional dependence on the eigenvalues of the Hamiltonian as for the quantum walk in Refs. [49, 50].

Next, we summarize how qubitization works [33]. One assumes black-box access to a signal oracle V that encodes H in the form:

$$(|0\rangle\langle 0|_a \otimes \mathbb{1}_s)\, V\, (|0\rangle\langle 0|_a \otimes \mathbb{1}_s) = |0\rangle\langle 0|_a \otimes H/\lambda \tag{3}$$

where $|0\rangle_a$ is in general a multi-qubit ancilla state in the computational basis, $\mathbb{1}_s$ is the identity gate on the system register and $\lambda \geq \|H\|$ is a normalization constant. For Hamiltonians given by a sum of unitaries,

$$H = \sum_{j=0}^{d-1} a_j U_j \qquad a_j > 0, \tag{4}$$

one constructs

$$U = (A^\dagger \otimes \mathbb{1})\,\text{SELECT-U}(A \otimes \mathbb{1}), \tag{5}$$

where A is an operator for state preparation acting as

$$A\,|0\rangle = \sum_{j=0}^{d-1} \sqrt{a_j/\lambda}\,|j\rangle \tag{6}$$

with $\lambda = \sum_{j=0}^{d-1} a_j$, and

$$\text{SELECT-U} = \sum_{j=0}^{d-1} |j\rangle\langle j| \otimes U_j. \tag{7}$$

For U that is Hermitian, which is the case for quantum chemistry, we can simply take $V = U$. If U is not Hermitian, then we may construct a Hermitian V as

$$V = |+\rangle\langle -| \otimes U + |-\rangle\langle +| \otimes U^\dagger \tag{8}$$

where $|\pm\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$. The multiqubit ancilla labelled "a" would then include this additional qubit, as well as the ancilla used for the control for SELECT-U. In either case we can then construct a unitary operator called the qubiterate as follows:

$$W = i(2\,|0\rangle\langle 0|_a \otimes \mathbb{1}_s - \mathbb{1})V. \tag{9}$$

The qubiterate transforms each eigenstate $|k\rangle$ of H as

$$W\,|0\rangle_a\,|k\rangle_s = i\frac{E_k}{\lambda}\,|0\rangle_a\,|k\rangle_s + i\sqrt{1 - \left|\frac{E_k}{\lambda}\right|^2}\,|0k^\perp\rangle_{as} \tag{10}$$

$$W\,|0k^\perp\rangle_{as} = i\frac{E_k}{\lambda}\,|0k^\perp\rangle_{as} - i\sqrt{1 - \left|\frac{E_k}{\lambda}\right|^2}\,|0\rangle_a\,|k\rangle_s \tag{11}$$

where $|0k^{\perp}\rangle_{as}$ has no support on $|0\rangle_a$. Thus, W performs rotation between two orthogonal states $|0\rangle_a |k\rangle_s$ and $|0k^{\perp}\rangle_{as}$. Restricted to this subspace, the qubiterate may be diagonalized as

$$W |\pm k\rangle_{as} = \mp e^{\mp i \arcsin(E_k/\lambda)} |\pm k\rangle_{as} \tag{12}$$

$$|\pm k\rangle_{as} = \frac{1}{\sqrt{2}} \left(|0\rangle_a |k\rangle_s \pm |0k^{\perp}\rangle_{as} \right). \tag{13}$$

This spectrum is exact, and identical to that for the quantum walk in Refs. [49, 50]. This procedure is also simple, requiring only two queries to U and a number of gates to implement the controlled-Z operator $(2|0\rangle\langle 0|_a \otimes \mathbb{1}_s - \mathbb{1})$ scaling linearly in the number of controls.

We may replace the time evolution operator with the qubiterate W in phase estimation, and phase estimation will provide an estimate of $\arcsin(E_k/\lambda)$ or $\pi - \arcsin(E_k/\lambda)$. In either case taking the sine gives an estimate of E_k/λ, so it is not necessary to distinguish the cases. Any problems with phase ambiguity are eliminated, because performing the sine of the estimated phase of W yields an unambiguous estimate for E_k. Note also that $\lambda \geq \|H\|$ implies that $|E_k/\lambda| \leq 1$.

More generally, any unitary operation $e^{if(H)}$ that has eigenvalues related to those of the Hamiltonian would work so long as the function $f(\cdot) : \mathbb{R} \to (-\pi, \pi)$ is known in advance and invertible. One may perform phase estimation to obtain a classical estimate of $f(E_k)$, then invert the function to estimate E_k. To first order, the error of the estimate would then propagate like

$$\sigma_{E_k} = \left| \left(\frac{df}{dx}\bigg|_{x=E_k} \right) \right|^{-1} \sigma_{f(E_k)}. \tag{14}$$

In our example, with standard deviation σ_{phase} in the phase estimate of W, the error in the estimate is

$$\sigma_{E_k} = \sigma_{\text{phase}} \sqrt{\lambda^2 - E_k^2} \leq \lambda \, \sigma_{\text{phase}} \, . \tag{15}$$

Obtaining uncertainty ϵ for the phase of W requires applying W a number of times scaling as $1/\epsilon$. Hence, obtaining uncertainty ϵ for E_k requires applying W a number of times scaling as λ/ϵ. For Hamiltonians given by sums of unitaries, as in chemistry, each application of W uses $O(1)$ applications of state preparations and SELECT-U operations. In terms of these operations, the complexities of Section II have multiplying factors of λ.

CONCLUSION

We have described three techniques which we expect will be practical and useful for the quantum simulation of fermionic systems. Our first technique provides an exponentially faster method for antisymmetrizing configuration states, a necessary step for simulating fermions in first quantization. We expect that in virtually all circumstances the gate complexity of this algorithm will be nearly trivial compared to the cost of the subsequent phase estimation. Then, we showed that when one has knowledge of an upper bound on the ground state energy that is separated from the first excited state energy, one can prepare ground states using phase estimation with lower cost. We discussed why this situation is anticipated for many problems in chemistry and provided numerics for a situation in which this trick reduced the gate complexity of preparing the ground state of molecular water by more than an order of magnitude. Finally, we explained how qubitization [33] provides a unitary that can be used for phase estimation without introducing the additional error inherent in Hamiltonian simulation.

We expect that these techniques will be useful in a variety of contexts within quantum simulation. In particular, we anticipate that the combination of the three techniques will enable exceptionally efficient quantum simulations of chemistry based on methods similar to those proposed in [18]. While specific gate counts will be the subject of a future work, we conjecture that such techniques will enable simulations of systems with roughly a hundred electrons on a million point grid with fewer than a billion T gates. With such low T counts, simulations such as the mechanism of Nitrogen fixation by ferredoxin, explored for quantum simulation in [51], should be practical to implement within the surface code in a reasonable amount of time with fewer than a few million physical qubits and error rates just beyond threshold.

ACKNOWLEDGEMENTS

The authors thank Matthias Troyer for relaying the idea of Alexei Kitaev that phase estimation could be performed without Hamiltonian simulation. We thank Jarrod McClean for discussions about molecular excited state gaps. DWB is funded by an Australian Research Council Discovery Project (Grant No. DP160102426).

AUTHOR CONTRIBUTIONS

DWB proposed the algorithms of Section I and the basic idea behind Section II as solutions to issues raised by RB. MK, AS and YRS worked out and wrote up the details of Section I and associated appendices. RB connected developments to chemistry simulation, conducted numerics, and wrote Section II with input from DWB. Based on discussions with NW, GHL suggested the basic idea of Section III. CG helped to improve the gate complexity of our comparator circuits. Remaining aspects of the paper were written by RB and DWB with assistance from MK, AS and YRS.

[1] L. Mueck, Nature Chemistry **7**, 361 (2015).
[2] M. Mohseni, P. Read, H. Neven, S. Boixo, V. Denchev, R. Babbush, A. Fowler, V. Smelyanskiy, and J. Martinis, Nature **543**, 171 (2017).
[3] A. Y. Kitaev, eprint arXiv: quant-ph/9511026 (1995).
[4] A. Aspuru-Guzik, A. D. Dutoi, P. J. Love, and M. Head-Gordon, Science **309**, 1704 (2005).
[5] B. Bauer, D. Wecker, A. J. Millis, M. B. Hastings, and M. Troyer, e-print arXiv: 1510.03859 (2015).
[6] Z. Jiang, K. J. Sung, K. Kechedzhi, V. N. Smelyanskiy, and S. Boixo, e-print arXiv:1711.05395 (2017).
[7] D. S. Abrams and S. Lloyd, Physical Review Letters **83**, 5162 (1999).
[8] D. W. Berry, G. Ahokas, R. Cleve, and B. C. Sanders, Communications In Mathematical Physics **270**, 359 (2007).
[9] D. W. Berry, A. M. Childs, R. Cleve, R. Kothari, and R. D. Somma, Physical Review Letters **114**, 090502 (2015).
[10] G. H. Low and I. L. Chuang, Physical Review Letters **118**, 010501 (2017).
[11] J. D. Whitfield, J. Biamonte, and A. Aspuru-Guzik, Mol. Phys. **109**, 735 (2011).
[12] M. B. Hastings, D. Wecker, B. Bauer, and M. Troyer, Quantum Information & Computation **15**, 1 (2015).
[13] D. Poulin, M. B. Hastings, D. Wecker, N. Wiebe, A. C. Doherty, and M. Troyer, Quantum Information & Computation **15**, 361 (2015).
[14] K. Sugisaki, S. Yamamoto, S. Nakazawa, K. Toyota, K. Sato, D. Shiomi, and T. Takui, The Journal of Physical Chemistry A **120**, 6459 (2016).
[15] F. Motzoi, M. Kaicher, and F. Wilhelm, e-print arXiv: 1705.10863 (2017).
[16] R. Babbush, N. Wiebe, J. McClean, J. McClain, H. Neven, and G. K.-L. Chan, e-print arXiv: 1706.0023 (2017).
[17] I. D. Kivlichan, J. McClean, N. Wiebe, C. Gidney, A. Aspuru-Guzik, G. K.-L. Chan, and R. Babbush, e-print arXiv: 1711:04789 (2017).
[18] I. D. Kivlichan, N. Wiebe, R. Babbush, and A. Aspuru-Guzik, Journal of Physics A: Mathematical and Theoretical **50**, 305301 (2017).
[19] I. Kassal, S. P. Jordan, P. J. Love, M. Mohseni, and A. Aspuru-Guzik, Proceedings of the National Academy of Sciences **105**, 18681 (2008).
[20] B. Toloui and P. J. Love, e-print arXiv: 1312.2579 (2013).
[21] R. Babbush, D. W. Berry, I. D. Kivlichan, A. Y. Wei, P. J. Love, and A. Aspuru-Guzik, New Journal of Physics **18**, 033032 (2016).
[22] R. Babbush, D. W. Berry, I. D. Kivlichan, A. Y. Wei, P. J. Love, and A. Aspuru-Guzik, e-print arXiv: 1506.01029 (2015).
[23] T. Kato, Communications on Pure and Applied Mathematics **10**, 151 (1957).
[24] R. D. Somma, G. Ortiz, J. Gubernatis, E. Knill, and R. Laflamme, Physical Review A **65**, 17 (2002).
[25] J. T. Seeley, M. J. Richard, and P. J. Love, Journal of Chemical Physics **137**, 224109 (2012).
[26] A. Tranter, S. Sofia, J. Seeley, M. Kaicher, J. McClean, R. Babbush, P. V. Coveney, F. Mintert, F. Wilhelm, and P. J. Love, International Journal of Quantum Chemistry **115**, 1431 (2015).
[27] S. Bravyi, J. M. Gambetta, A. Mezzacapo, and K. Temme, e-print arXiv: 1701.08213 (2017).
[28] V. Havlicek, M. Troyer, and J. D. Whitfield, Physical Review A **95**, 032332 (2017).
[29] K. Setia and J. D. Whitfield, e-print arXiv: 1712.00446 (2017).
[30] M. Steudtner and S. Wehner, e-print arXiv:1712.07067 (2017).
[31] D. S. Abrams and S. Lloyd, Physical Review Letters **79**, 4 (1997).
[32] N. J. Ward, I. Kassal, and A. Aspuru-Guzik, Journal Of Chemical Physics **130**, 194105 (2008).
[33] G. H. Low and I. L. Chuang, e-print arXiv: 1610.06546 (2016).
[34] K. E. Batcher, Communications of the ACM **32**, 307 (1968).
[35] K. J. Liszka and K. E. Batcher, International Conference on Parallel Processing **1**, 105 (1993).
[36] M. Ajtai, J. Komlós, and E. Szemerédi, in *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83 (ACM, New York, NY, USA, 1983) pp. 1–9.
[37] M. S. Paterson, Algorithmica **5**, 75 (1990).
[38] M. T. Goodrich, in *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14 (ACM, New York, NY, USA, 2014) pp. 684–693.
[39] S.-T. Cheng and C.-Y. Wang, IEEE Transactions on Circuits and Systems I: Regular Papers **53**, 316 (2006).
[40] R. Beals, S. Brierley, O. Gray, A. W. Harrow, S. Kutin, N. Linden, D. Shepherd, and M. Stather, Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences **469** (2013).
[41] B. L. Higgins, D. W. Berry, S. D. Bartlett, H. M. Wiseman, and G. J. Pryde, Nature **450**, 393 (2007).
[42] H. Wang, S. Kais, A. Aspuru-Guzik, and M. R. Hoffmann, Physical Chemistry Chemical Physics **10**, 5388 (2008).
[43] L. Veis and J. Pittner, The Journal of Chemical Physics **140**, 1 (2014).
[44] J. R. McClean, R. Babbush, P. J. Love, and A. Aspuru-Guzik, The Journal of Physical Chemistry Letters **5**, 4368 (2014).
[45] R. Babbush, J. McClean, D. Wecker, A. Aspuru-Guzik, and N. Wiebe, Physical Review A **91**, 022311 (2015).
[46] T. Helgaker, P. Jorgensen, and J. Olsen, *Molecular Electronic Structure Theory* (Wiley, 2002).
[47] J. R. McClean, I. D. Kivlichan, D. S. Steiger, Y. Cao, E. S. Fried, C. Gidney, T. Häner, V. Havlíček, Z. Jiang, M. Neeley, J. Romero, N. Rubin, N. P. D. Sawaya, K. Setia, S. Sim, W. Sun, K. Sung, and R. Babbush, e-print arXiv: 1710.07629 (2017).
[48] R. M. Parrish, L. A. Burns, D. G. A. Smith, A. C. Simmonett, A. E. DePrince, E. G. Hohenstein, U. Bozkaya, A. Y. Sokolov, R. Di Remigio, R. M. Richard, J. F. Gonthier, A. M. James, H. R. McAlexander, A. Kumar, M. Saitow, X. Wang, B. P. Pritchard, P. Verma, H. F. Schaefer, K. Patkowski, R. A. King, E. F. Valeev, F. A. Evangelista, J. M. Turney, T. D. Crawford, and C. D. Sherrill, Journal of Chemical Theory and Computation

13, 3185 (2017).
[49] A. M. Childs, Communications in Mathematical Physics **294**, 581 (2010).
[50] D. W. Berry and A. M. Childs, Quantum Information & Computation **12**, 29 (2012).
[51] M. Reiher, N. Wiebe, K. M. Svore, D. Wecker, and M. Troyer, Proceedings of the National Academy of Sciences **114**, 7555 (2017).
[52] M. Bellare, J. Kilian, and P. Rogaway, Journal of Computer and System Sciences **61**, 362 (2000).
[53] D. E. Knuth, *The art of computer programming*, Vol. 3 (Pearson Education, 1997).
[54] M. Codish, L. Cruz-Filipe, T. Ehlers, M. Mller, and P. Schneider-Kamp, Journal of Computer and System Sciences (2016), https://doi.org/10.1016/j.jcss.2016.04.004.
[55] C. Jones, Physical Review A **87**, 022328 (2013).
[56] C. Gidney, e-print arXiv: 1709.06648 (2017).
[57] R. Durstenfeld, Communications of the ACM **7**, 420 (1964).
[58] M. A. Nielsen and I. L. Chuang, *Quantum Computing and Quantum Information* (Cambridge University Press, 2000).

Appendix A: Complexity Scaling of Ref. [31]

An approach to prepare appropriately antisymmetrized states starting from an ordered state (where $r_1, \ldots, r_\eta$ are in ascending order) was proposed in Ref. [31]. The complexity scaling with η given in that work was $\widetilde{O}(\eta^2)$, but there is a step that appears to scale as $\widetilde{O}(\eta^3)$. In Step III of that proposal, a permutation is generated by setting $B'[i]$ equal to the $B[i]$th natural number that is not contained in the set $\{B'[1], \ldots, B'[i-1]\}$. To implement this step one would need to go through $O(\eta)$ natural numbers, and for each perform equality testing with each of the $O(\eta)$ numbers $\{B'[1], \ldots, B'[i-1]\}$. This would need to be done for each of $O(\eta)$ values of i, which would yield overall complexity $\widetilde{O}(\eta^3)$. The same step is required in Ref [32] and thus that procedure also appears to have complexity $\widetilde{O}(\eta^3)$ despite also claiming to scale as $\widetilde{O}(\eta^2)$.

Appendix B: Analysis of 'Delete Collisions' Step

In this Appendix, we explain the most difficult-to-understand step of our algorithm: the step in which we delete collisions from `seed`. There are two important points that require explanation. First, we have to show that the probability of failure is small. Second, we have to show that the resulting state of `seed` is disentangled from `record`, as we wish to uncompute `record` during the final step of our algorithm.

To explain these two points, we begin with an analysis of the state of `seed` after Step 1. The state of `seed` is

$$\frac{1}{f(\eta)^{\eta/2}} \sum_{\ell_0,\ldots,\ell_{\eta-1}=0}^{f(\eta)-1} |\ell_0, \ldots, \ell_{\eta-1}\rangle \,. \tag{B1}$$

We can decompose the state space of `seed` into two orthogonal subspaces: the 'repetition-free' subspace

$$\text{span}\left\{|\ell_0, \ldots, \ell_{\eta-1}\rangle \,|\forall i \neq j : \ell_i \neq \ell_j\right\} \tag{B2}$$

and its orthogonal complement. If we project the state of `seed` onto the repetition-free subspace, we obtain the unnormalized vector

$$\frac{1}{f(\eta)^{\eta/2}} \sum_{0\leq \ell_0 < \ldots < \ell_{\eta-1} < f(\eta)} \sum_{\sigma \in S_\eta} |\sigma\left(\ell_0, \ldots, \ell_{\eta-1}\right)\rangle \,. \tag{B3}$$

The norm of this vector is

$$\frac{\eta!}{f(\eta)^\eta} \binom{f(\eta)}{\eta}, \tag{B4}$$

which is equal to $1 - C(f(\eta), \eta)$ in the terminology of Proposition A.1 in [52].

We sort the register in Step 2 before detecting repetitions in Step 3, because then it is only necessary to check adjacent registers. The probability of repetitions in unaffected by the sort, because it is unitary and does not affect whether there are repetitions. Therefore the probability of failure (detection of a repetition) in Step 3 is equal to $C(f(\eta), \eta)$. Using Proposition A.1 in [52], the probability of failure is bounded as

$$\Pr(\text{repetition}) = C(f(\eta), \eta) \leq \frac{\eta(\eta-1)}{2f(\eta)}, \tag{B5}$$

which is less than $1/2$ for $f(\eta) \geq \eta^2$. The repetition-free outcome can therefore be achieved after fewer than two attempts on average. One can improve the success probability by using a larger function f or by using amplitude amplification.

We now show that $\texttt{seed} \otimes \texttt{record}$ is in an unentangled state after Step 3. After Step 1, the state of $\texttt{seed} \otimes \texttt{record}$ projected to the repetition-free subspace can be represented (up to normalization) as

$$\sum_{0\leq \ell_0 < \ldots < \ell_{\eta-1} < f(\eta)} \sum_{\sigma \in S_\eta} |\sigma\left(\ell_0, \ldots, \ell_{\eta-1}\right)\rangle_{\texttt{seed}} \, |\iota\rangle_{\texttt{record}} \,. \tag{B6}$$

Here we represent the state of `record` as a recording of all permutations we have applied to `seed`; ι represents the identity permutation. During Step 2, a sequence of permutations $\sigma_1, \ldots, \sigma_T$ (where T depends on the choice of sorting network) is applied to `seed` and recorded on `record`. This sequence of permutations is chosen so that

$$\sigma_T \circ \cdots \circ \sigma_1 \circ \sigma\left(\ell_0, \ldots, \ell_{\eta-1}\right) = \left(\ell_0, \ldots, \ell_{\eta-1}\right), \tag{B7}$$

where $0 \leq \ell_0 < \ldots < \ell_{\eta-1} < f(\eta)$. That is to say,[2]

$$\sigma_T \circ \cdots \circ \sigma_1 \circ \sigma = \iota. \tag{B8}$$

Therefore, the state of $\texttt{seed} \otimes \texttt{record}$ after Step 3 is (up to normalization)

$$\sum_{0 \leq \ell_0 < \ldots < \ell_{\eta-1} < f(\eta)} |\ell_0, \ldots, \ell_{\eta-1}\rangle_{\texttt{seed}} \sum_{\sigma \in S_\eta} |\sigma_1, \ldots, \sigma_T\rangle_{\texttt{record}} . \tag{B9}$$

This is a product state. Therefore, `seed` can be discarded after Step 3 without affecting `record`.

Appendix C: Quantum Sorting

1. Quantum Sorting Networks

In this appendix, we expand on the implementation of quantum sorting networks and discuss some examples with favorable scaling. We also illustrate that for small number of inputs to be sorted (up to $\eta = 20$), concrete bounds have been derived for optimized circuit depth as well as the number of comparators. This may be of interest and useful for implementing quantum simulations of small molecules, also in view of the observation that $\eta \approx 20$ is nearly reaching a number of electrons for where classical simulations become intractable.

Sorting networks are logical circuits that consist of wires carrying values and comparator modules applied to pairs of wires, that compare values and swap them if they are not in the correct order. Wires represent bit strings (integers are stored in binary) in classical sorting networks and qubit strings in their quantum analogues. A classical comparator is a sort on two numbers, which gives the transformation $(A, B) \mapsto (\min(A, B), \max(A, B))$. A quantum comparator is its reversible version where we record whether the items were already sorted (ancilla state $|0\rangle$) or the comparator needed to apply a swap (ancilla state $|1\rangle$); see Figure 1.

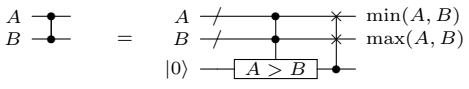

FIG. 1. The standard notation for a comparator is indicated on the left. Its implementation as a quantum circuit is shown on the right. In the first step, we compare two inputs with values A and B and save the outcome (1 if $A > B$ is true and 0 otherwise) in a single-qubit ancilla. In the second step, conditioned on the value of the ancilla qubit, the values A and B in the two wires are swapped.

[2] Note that no condition like Eq. (B8) holds in the orthogonal complement of the repetition-free subspace. There are multiple permutations that sort an unsorted array that has repeated elements, so the choice of σ would be ambiguous.

Note that the positions of comparators are set as a predetermined fixed sequence in advance and therefore cannot depend on the inputs. This makes sorting networks viable candidates for quantum computing. Many of the sorting networks are also highly parallelizable, thus allowing low-depth, often polylogarithmic, performance.

Several common sort algorithms such as the insert and bubble sorts can be represented as sorting networks. However, these algorithms have poor time complexity even after parallelization. More efficient runtime can be achieved, for example, using the bitonic sort, which is illustrated for 8 inputs in Figure 2. The bitonic sort uses $O(\eta \log^2 \eta)$ comparators and $O(\log^2 \eta)$ depth, thus achieving an exponential improvement in depth compared to common sorting techniques.

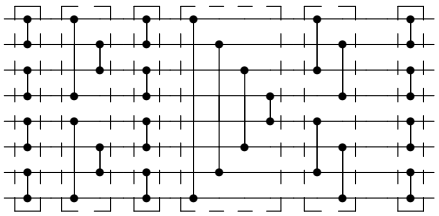

FIG. 2. Example of a *bitonic sort* on 8 inputs. The ancillae necessary to record the results as part of implementing each of the comparators are omitted for clarity. Comparators in each dashed box can be applied in parallel for depth reduction.

Optimizing sorting networks for small inputs is an active research area in parallel programming. Knuth [53] and later Codish *et al.* [54] gave networks for sorting up to 17 numbers that were later shown to be optimal in depth, and up to $\eta \leq 10$ also optimal in the number of comparators. Optimizations for up to 20 inputs have recently been achieved, see Table 1 in [54]. In such optimizations one typically distinguishes between the optimal depth problem and the problem of minimizing the overall number of comparators. For illustration, the best known sorting networks for 20 numbers require depth 11 and 92 comparators, with lower bounds reported as 10 and 73 respectively. Efficient sorting networks can be produced by in-place merging of sorting networks with smaller sizes. However, this procedure necessarily produces some overhead.

For our resource analysis we assume that the quantum sorting network has η wires, where each wire represents a quantum register of length d (i.e., consists of d qubits). The resource requirement for implementing the quantum sort is obtained by taking the (classical) sorting network depth or the overall number of comparators involved and multiplying it by the corresponding resources needed to construct a comparator. As explained above, the latter requires one query to a comparison oracle, whose circuit implementation and complexity are provided in Appendix C 2, and a conditional swap applied to the compared registers of size d controlled by the single-qubit

ancilla holding the result of the comparison.

The construction of the comparison oracle as well as the implementation of the conditional swaps both yield a network consisting predominantly of Toffoli, NOT and CNOT gates requiring $O(d)$ elementary gate operations but only $O(\log d)$ circuit depth. Indeed, as shown in Appendix C 2, the comparison oracle can be implemented such that the operations can mostly be performed in parallel with only $O(\log d)$ circuit depth.

When implementing conditional swaps on two registers of size d as part of a comparator, all elementary swaps between the corresponding qubits of these registers must be controlled by the very same ancilla qubit, namely the one encoding the result of the comparison oracle. This suggests having to perform all the controlled swaps in sequence, as they all are to be controlled by the same qubit, which would imply depth scaling $O(d)$ rather than $O(\log d)$. Yet the conditional swaps can also be parallelized. This can be achieved by first copying the bit of the ancilla holding the result of the comparison to $d-1$ additional ancillae, all initialized in $|0\rangle$. Such an expansion of the result to d copies can be attained with a parallelized arrangement of $O(d)$ CNOTs but with circuit depth only $O(\log d)$. After copying, all the d controlled elementary swaps can then be executed in parallel (by using the additional ancillae) with circuit depth only $O(1)$. After executing the swaps, the $d-1$ additional ancillae used for holding the copied result of comparison are uncomputed again, by reversing the copying process. While this procedure requires $O(d)$ ancillary space overhead, it optimizes the depth. The overall space overhead of the quantum comparator is also $O(d)$.

Taking $d = \lceil \log N \rceil$ (the largest registers used in Step 4 of our sort-based antisymmetrization algorithm), conducting the quantum bitonic sort, for instance, thus requires $O(\eta \log^2(\eta) \log N)$ elementary gates but only $O(\log^2(\eta) \log\log N)$ circuit depth, while the overall worst-case ancillary space overhead amounts to $O(\eta \log^2(\eta) \log N)$.

2. Comparison Oracle

Here we describe how to implement reversibly the comparison of the value held in one register with the value carried by a second equally-sized register, and store the result (larger or not) in a single-qubit ancilla. We term the corresponding unitary process a *'comparison oracle'*. We need to use it for implementing the comparator modules of quantum sorting networks as well as in our antisymmetrization approach based on the quantum Fisher-Yates shuffle. We first explain a naive method for comparison with depth linear in the length of the involved registers. In the second step we then convert this prototype into an algorithm with depth logarithmic in the register length using a divide and conquer approach.

Let $\mathtt{A}$ and $\mathtt{B}$ denote the two equally sized registers to be compared, and A and B the values held by these two registers. To determine whether $A > B$ or $A < B$ or $A = B$, we compare the registers in a *bit-by-bit* fashion, starting with their most significant bits and going down to their least significant bits. At the very first occurrence of an i such that $\mathtt{A}[i] \neq \mathtt{B}[i]$, i.e., either $\mathtt{A}[i] = 1$ and $\mathtt{B}[i] = 0$ or $\mathtt{A}[i] = 0$ and $\mathtt{B}[i] = 1$, we know that $A > B$ in the first case and $A < B$ in the second case. If $\mathtt{A}[i] = \mathtt{B}[i]$ for all i, then $A = B$. We now show how to infer and record the result in a reversible way.

Register	i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
$\mathtt{A}$	0	0	0	0	1	0	1	0	1
$\mathtt{B}$	0	0	0	0	0	1	1	1	0
$\mathtt{A}'$	0	0	0	0	1	1	1	1	1
$\mathtt{B}'$	0	0	0	0	0	0	0	0	0

TABLE I. Example illustrating the idea of reversible bitwise comparison. Here, $d = 9$, the value held in register $\mathtt{A}$ is 21 and the value held in register $\mathtt{B}$ is 14. The index i labels the bits of the registers, with $i = 0$ designating the most significant bits, respectively. Observe that the first occurrence of $\mathtt{A}[i] \neq \mathtt{B}[i]$ is for $i = 4$, at which stage the value of ancilla $\mathtt{A}'[4]$ is switched to 1, as $\mathtt{A}[4] > \mathtt{B}[4]$. This change causes all lesser significant bits of $\mathtt{A}'$ also to be switched to 1, whereas all bits of $\mathtt{B}'$ remain 0. Thus, the least significant bits of $\mathtt{A}'$ and $\mathtt{B}'$ contain information about which number is larger. Here, $\mathtt{A}'[8] = 1$ implies $A > B$.

To achieve a reversible comparison, we employ two ancillary registers, each consisting of d qubits, and each initialized to state $|0\rangle^{\otimes d}$, respectively. We denote them by $\mathtt{A}'$ and $\mathtt{B}'$. They are introduced for the purpose of recording the result of bitwise comparison as follows. $\mathtt{A}'[i] = 1$ implies that after i bitwise comparisons we know with certainty that $A = \max(A, B)$, while $\mathtt{B}'[i] = 1$ implies $B = \max(A, B)$. These implications can be achieved by the following protocol, which is illustrated by a simple example in Table I.

To start, at $i = 0$ we compare the most significant bits $\mathtt{A}[0]$ and $\mathtt{B}[0]$, and write 1 into ancilla $\mathtt{A}'[0]$ if $\mathtt{A}[0] > \mathtt{B}[0]$, or write 1 into ancilla $\mathtt{B}'[0]$ if $\mathtt{A}[0] < \mathtt{B}[0]$. Otherwise the ancillas remain as 0. For each $i > 0$, if $\mathtt{A}'[i-1] = 0$ and $\mathtt{B}'[i-1] = 0$ we compare $\mathtt{A}[i]$ and $\mathtt{B}[i]$ and record the outcome to $\mathtt{A}'[i]$ and $\mathtt{B}'[i]$ in the same way as for $i = 0$. If however $\mathtt{A}'[i-1] = 1$ and $\mathtt{B}'[i-1] = 0$, we already know that $A > B$, so we set $\mathtt{A}'[i] = 1$ and $\mathtt{B}'[i] = 0$. Similarly, $\mathtt{A}'[i-1] = 0$ and $\mathtt{B}'[i-1] = 1$ implies $A < B$, so we set $\mathtt{A}'[i] = 0$ and $\mathtt{B}'[i] = 1$. We continue doing so until we reach the least significant bits. This results in the least significant bits of the ancillary registers $\mathtt{A}'$ and $\mathtt{B}'$ holding information about $\max(A, B)$. If these least significant bits are both 0, then $A = B$. At the end the least significant bit of $\mathtt{A}'$ has value 1 if $A > B$, and 0 if $A \leq B$. This bit can be copied to an $\mathtt{output}$ register, and the initial sequence of operations reversed to erase the other ancilla qubits.

While this algorithm works, it has the drawback that the bitwise comparison is conducted *sequentially*, which results in circuit-depth scaling $O(d)$. It also uses more

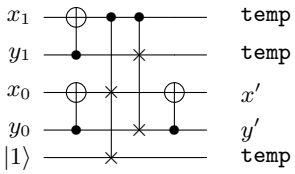

FIG. 3. A circuit that implements COMPARE2, taking a pair of 2-bit integers and outputting a pair of single bits while preserving inequalities. The input pair is $(x, y) = (x_0 + 2x_1, y_0 + 2y_1)$. The output pair is (x', y') and will satisfy $\mathrm{sign}(x' - y') = \mathrm{sign}(x - y)$. Output qubits marked "temp" store values that are not needed, and are kept until a later uncompute step where the inputs are restored. Each Fredkin gate within the circuit can be computed using 4 T gates and (by storing an ancilla not shown) later uncomputed using 0 T gates [55, 56].

ancilla qubits than necessary. We can improve upon this. We can reduce the number of ancilla qubits by reusing some input bits as output bits, and we can achieve a depth scaling of $O(\log d)$ by parallelizing the bitwise comparison. To introduce a parallelization, observe the following. Let us split the register $\mathtt{A}$ into two parts: $\mathtt{A}_1$ consisting of the first approximately $d/2$ bits and $\mathtt{A}_2$ consisting of the remaining approximately $d/2$ bits. Split register $\mathtt{B}$ in the very same way into subregisters $\mathtt{B}_1$ and $\mathtt{B}_2$. We can then determine which number is larger (or whether both are equal) for each pair (A_1, B_1) and (A_2, B_2) separately in parallel (using the method described above) and record the results of the two comparisons in ancilla registers $(\mathtt{A}'_1, \mathtt{B}'_1)$, $(\mathtt{A}'_2, \mathtt{B}'_2)$. The least significant bits of these four ancilla registers can then be used to deduce whether $A > B$ or $A < B$ or $A = B$ with just a single bitwise comparison. Thus, we effectively halved the depth by dividing the problem into smaller problems and merging them afterwards. We now explain a bottom-up implementation.

Instead of comparing the whole registers $\mathtt{A}$ and $\mathtt{B}$, our parallelized algorithm slices $\mathtt{A}$ and $\mathtt{B}$ into pairs of bits – the first slice contains $\mathtt{A}[0]$ and $\mathtt{A}[1]$, the second slice consists of $\mathtt{A}[2]$ and $\mathtt{A}[3]$, etc., and in the very same way for $\mathtt{B}$. The key step takes the corresponding slices of $\mathtt{A}$ and $\mathtt{B}$ and overwrites the second bit of each slice with the outcome of the comparison. The first bit of each slice is then ignored, so that the comparison results stored in the second bits become the next layer on which bitwise comparisons are performed. We denote the i'th bit forming the registers of the j^{th} layer by $\mathtt{A}^j[i]$ and $\mathtt{B}^j[i]$. The original registers $\mathtt{A}$ and $\mathtt{B}$ correspond to $j = 0$: $\mathtt{A}^0 \equiv \mathtt{A}$ and $\mathtt{B}^0 \equiv \mathtt{B}$. The part of the circuit that implements a single bitwise comparison is depicted in Figure 3. We denote the corresponding transformation by 'COMPARE2', i.e. $(A^{j+1}[i], B^{j+1}[i]) = \text{COMPARE2}(A^j[2i], B^j[2i], A^j[2i+1], B^j[2i+1])$, meaning that it prepares the bits $A^{j+1}[i], B^{j+1}[i]$ storing the comparison result.

At each step, comparisons of the pairs of the original

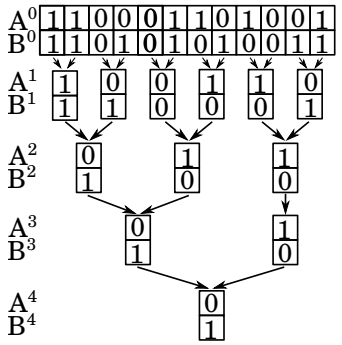

FIG. 4. Parallelized bitwise comparison. Observe how each step reduces the size of the problem by approximately one half, while using a constant depth for computing the results.

arrays can be performed in parallel, and produce two new arrays with approximately half the size of the original ones to record the results. Thus, at each step we approximately halve the size of the problem, while using a constant depth for computing the results. The basic idea is illustrated in Figure 4. This procedure is repeated for $\lceil \log d \rceil$ steps[3] until registers $\mathtt{A}^{\text{fin}} := \mathtt{A}^{\lceil \log d \rceil}$ and $\mathtt{B}^{\text{fin}} := \mathtt{B}^{\lceil \log d \rceil}$ both of size 1 have been prepared.

This parallelized algorithm is perfectly suited for comparing arrays whose length d is a power of 2. If d is not a power of 2, we can either pad $\mathtt{A}$ and $\mathtt{B}$ with 0s prior to their most significant bits without altering the result, or introduce comparison of single bits (using only the first two gates from the circuit in Figure 3 with targets on $\mathtt{A}^{j+1}$ and $\mathtt{B}^{j+1}$ registers respectively).

Formally, we can express our comparison algorithm as follows, here assuming d to be a power of 2:

for $j = 0, \ldots, \log d - 1$ **do**
  for $i = 0, \ldots, \mathrm{size}(A^j)/2 - 1$ **do**
    $\big(A^{j+1}[i], B^{j+1}[i]\big) = \text{COMPARE2}(A^j[2i], B^j[2i], A^j[2i+1], B^j[2i+1])$
  end for
end for
return $(A^{\log d - 1}[0], B^{\log d - 1}[0])$

The key feature of this algorithm is that all the operations of the inner loop can be performed in parallel. Since one application of COMPARE2 requires only constant depth and constant number of operations, our comparison algorithm requires only depth $O(\log d)$.

Our comparison algorithm constructed above can indeed be used to output a result that distinguishes between $A > B$, $A < B$ and $A = B$. Observe that its reversible execution results in the ancillary single-qubit registers $\mathtt{A}^{\text{fin}}$ and $\mathtt{B}^{\text{fin}}$ generated in the very last step of the algorithm holding information about which number is larger or whether they are equal. Indeed, $\mathtt{A}^{\text{fin}}[0] = \mathtt{B}^{\text{fin}}[0]$ implies $A = B$, $\mathtt{A}^{\text{fin}}[0] < \mathtt{B}^{\text{fin}}[0]$ implies $A < B$, and

[3] All logarithms are taken to the base 2.

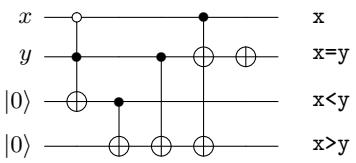

FIG. 5. A circuit that determines if two bits are equal, ascending, or descending. When the comparison is no longer needed, the results are uncomputed by applying the circuit in reverse order.

$\mathtt{A}^{\mathrm{fin}}[0] > \mathtt{B}^{\mathrm{fin}}[0]$ implies $A > B$. The three cases are separated into three control qubits by using the circuit shown in Figure 5. These individual control qubits can be used to control further conditional operations that depend on the result of the comparison.

For the purpose in our applications (comparator modules of quantum sorting networks or quantum Fisher-Yates shuffle), we only need to condition on whether $A > B$ is true or false. Thus, we only need the first operation from the circuit in Figure 5 which takes a single qubit initialized to $|0\rangle$ and transforms it into the output of the comparison oracle. After the output bit has been produced, we must reverse the complete comparison algorithm (invert the corresponding unitary process), thereby uncomputing all the ancillary registers that have been generated along this reversible process and restoring the input registers `A` and `B`.

The actual 'comparison oracle' thus takes as inputs two size-d registers `A` and `B` (holding values A and B) and a single-qubit ancilla `q` initialized to $|0\rangle$. It reversibly computes whether $A > B$ is true or false by executing the parallelized comparison process presented above. It copies the result (which is stored in $\mathtt{A}^{\mathrm{fin}}$) to ancilla `q`. It then executes the inverse of the comparison process. It outputs `A` and `B` *unaltered* and the ancilla `q` holding the result of the oracle: $q = 1$ if $A > B$ and $q = 0$ if $A \leq B$. As shown, this oracle has circuit size $O(d)$ but depth only $O(\log d)$ and a T-count of $8d + O(1)$.

Appendix D: Symmetrization Using The Quantum Fisher-Yates Shuffle

In this appendix we present an alternative approach for antisymmetrization that is not based on sorting, yielding a size- and depth-complexity $O(\eta^2 \log N)$, but with a lower spatial overhead than the sort-based method. Our alternative symmetrization method uses a quantum variant of the well-known Fisher-Yates shuffle, which applies a permutation chosen uniformly at random to an input array `input` of length η. A standard form of the algorithm is given in [57].

We consider the following variant of the Fisher-Yates shuffle:

for $k = 1, \ldots, (\eta - 1)$ **do**

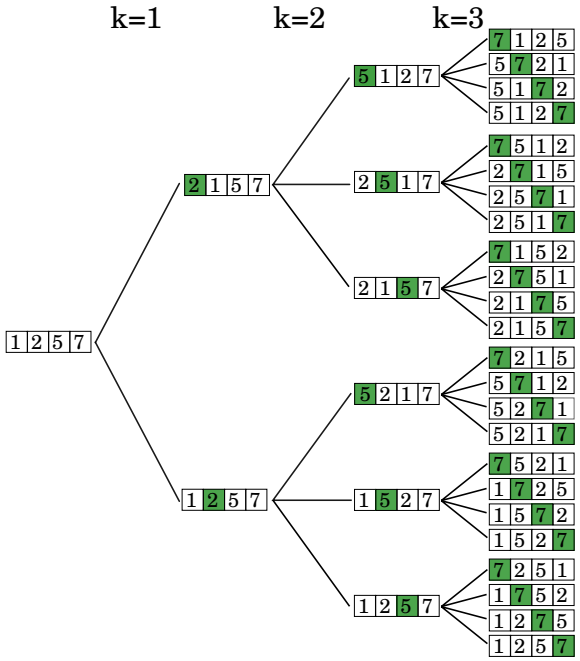

FIG. 6. A tree diagram for the Fisher-Yates shuffle applied to an example sorted array. Here the green boxes identify the array entry that has been swapped at each stage of the shuffle. Observe that the green boxes also label the largest value in the array truncated to position k.

Choose ℓ uniformly at random from $\{0, \ldots, k\}$.
Swap positions k and ℓ of `input`.
end for

The basic idea is illustrated in Figure 6 for $\eta = 4$.

There are two key steps that turn the Fisher-Yates shuffle into a quantum algorithm. First, our quantum implementation of the shuffle replaces the random selection of swaps with a superposition of all possible swaps. To achieve this superposition, the random variable is replaced by an equal-weight superposition $\frac{1}{\sqrt{k+1}} \sum_{\ell=0}^{k} |\ell\rangle$ in an ancillary register (called `choice`). At each step of the quantum Fisher-Yates shuffle, the `choice` register must begin and end in a fiducial initial state.

In order to reset the `choice` register, we introduce an additional `index` register, which initially contains the integers $0, \ldots, \eta - 1$. We shuffle both the length-η `input` register and the `index` register, and the simple form of `index` enables us to easily reset `choice`. The resulting state of the joint $\mathtt{input} \otimes \mathtt{index}$ register is still highly entangled; however, provided `input` was initially sorted in ascending order, we can disentangle `index` from `input`.

Our quantum Fisher-Yates shuffle consists of the following steps:

1. **Initialization.** Prepare the `choice` register in the state $|0\rangle$. Prepare the `index` register in the state $|0, 1, \ldots, \eta - 1\rangle$. Also set a classical variable $k = 1$.

2. **Prepare `choice`.** Transform the `choice` register from $|0\rangle$ to $\frac{1}{\sqrt{k+1}} \sum_{\ell=0}^{k} |\ell\rangle$.

3. **Execute swap.** Swap element k of `input` with the element specified by `choice`. If a non-trivial swap was executed (i.e. if `choice` did not specify k), apply a phase of -1 to the `input` register. Also swap element k of `index` with the element specified by `choice`.

4. **Reset `choice`.** For each $\ell = 1, \ldots, k$, subtract ℓ from the `choice` register if position ℓ in `index` is equal to k. The resulting state of `choice` is $|0\rangle$.

5. **Repeat.** Increment k by one. If $k < \eta$, go to Step 2. Otherwise, proceed to the next step.

6. **Disentangle `index` from `input`.** For each $k \neq \ell = 0, 1, \ldots, \eta - 1$, subtract 1 from position ℓ of `index` if the element at position k in `input` is greater than the element at position ℓ in `input`. The resulting state of `index` is $|0, 0, \ldots, 0\rangle$, which is disentangled from `input`.

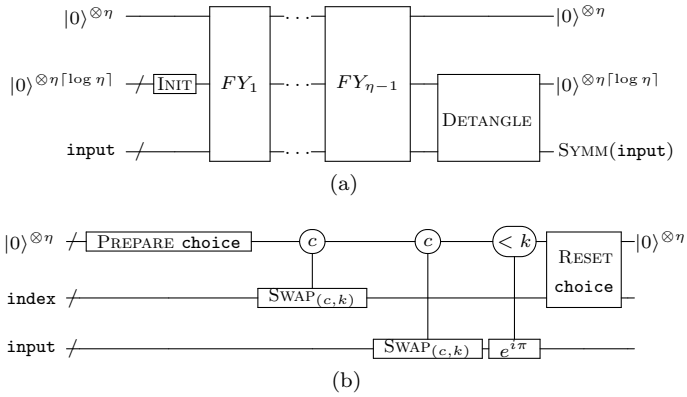

FIG. 7. An overview of symmetrization by quantum Fisher-Yates shuffle. (a) High-level view of the algorithm. The procedure acts on registers labeled (top to bottom) `choice`, `index` and `input`. (b) Detail for the Fisher-Yates block FY_k. The first register (again labeled `choice`) is used to select the target of the two selected swap steps. Then a phase $e^{i\pi} = -1$ is applied to the `input` register if a swap was performed, i.e. if the `choice` register encodes a value less than k. Each block FY_k is completed by resetting the `choice` register back to its original state $|0\rangle^{\otimes\eta}$.

We present an overview of the algorithm in Figure 7. At the highest level, depicted in Figure 7a, we apply an initialization procedure to `index`, then $\eta - 1$ 'Fisher-Yates' blocks (FY_k for $k = 1, \ldots, \eta-1$), and finally a disentangling ('DETANGLE') procedure on `index` and `input`. Following the DETANGLE procedure, the ancillary registers `choice` and `index` are reset to their initial all-zero states and the `input` register has been symmetrized. In each Fisher-Yates block, depicted in Figure 7b, we apply the preparation operator Π_k to `choice`, apply selected swaps on `choice`+`index` and `choice`+`input`, then apply a phase conditioned on `choice` to `input`, and finally reset the `choice` register. Preparing and resetting `choice` as well as executing swap are therefore part of each Fisher-Yates block and are thus each applied a total of $\eta - 1$ times (for each of $k = 1, \ldots, \eta - 1$). Their gate counts and circuits depths must thus be multiplied by $(\eta - 1)$. Disentangling `index` and `input` is the most expensive step, but it is executed only once, so it contributes only an additive cost to the overall resource requirement.

In what follows, we explain each step of the algorithm and justify their corresponding resource contributions, which are briefly summarized here: Step 1 requires $O(\eta \log \eta)$ gates but has a negligible depth $O(1)$. Step 2 requires $O(\eta)$ gates and has the same depth complexity. Step 3 requires $O(\eta \log N)$ gates and has also depth $O(\eta \log N)$. Step 4 requires $O(\eta \log \eta)$ gates but has only depth $O(\log \eta)$. As Step 2 to Step 4 are repeated $\eta - 1$ times, the total gate count before Step 6 is $O(\eta^2 \log N)$. Finally, Step 6 requires $O(\eta^2 \log N)$ gates and has depth $O\left(\eta^2 \left[\log \log N + \log \eta\right]\right)$. Thus the total gate count of the quantum Fisher-Yates shuffle is $O(\eta^2 \log N)$. Because most of the gates need to be performed sequentially, the overall circuit depth of the algorithm is also $O(\eta^2 \log N)$.

Our complexity analysis is given in terms of elementary gate operations, a term we use loosely. Generally speaking, we treat all single-qubit gates as elementary and we allow up to two controls for free on each single-qubit gate. This definition of elementary gates includes several standard universal gate sets such as Clifford+T and Hadamard+Toffoli. A more restrictive choice of elementary gate set only introduces somewhat larger constant factors in most of the procedure. The exception is the application of Π_k in the first step of FY_k, where we require the ability to perform controlled single-qubit rotations of angle $\arcsin\left(\sqrt{\frac{\ell}{\ell+1}}\right)$, where $\ell = 1, \ldots, k$. The Solovay-Kitaev theorem implies a gate-count overhead that grows polylogarithmically in the inverse of the error tolerance. We now proceed by analyzing each step to the quantum Fisher-Yates shuffle.

1. Initialization

The first step is to initialize `choice` in the state $|0\rangle^{\otimes\eta}$. This is assumed to have zero cost. The `index` register is set to the state $|0, 1, \ldots, \eta - 1\rangle$ that represents the positions of the entries of `input` in ascending order. Because each of the η entries in `index` must be capable of storing any of the values $0, 1, \ldots, \eta - 1$, the size of `index` is $\eta\lceil\log \eta\rceil$ qubits. This step requires $O(\eta \log \eta)$ single-qubit gates that can be applied in parallel with circuit depth $O(1)$.

2. Fisher-Yates Blocks

Each Fisher-Yates block has three stages: prepare `choice`, executed selected swaps, and reset `choice`. The

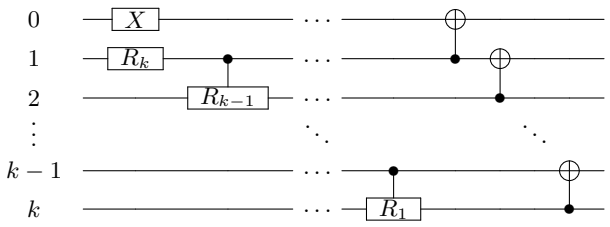

FIG. 8. Circuit for preparing the `choice` register at the beginning of block FY_k. See Eq. (D5) for the definition of R_ℓ.

exact steps depend on the encoding of the `choice` register; in particular, whether it is binary or unary.

We elect the conceptually simplest encoding of `choice`, which is a kind of unary encoding. We use η qubits (labelled $0, 1, \ldots, \eta$), define

$$|\texttt{null}\rangle = |0\rangle^{\otimes \eta} \tag{D1}$$

and encode

$$|\ell\rangle = X_\ell \, |\texttt{null}\rangle \,, \tag{D2}$$

where X_ℓ is the Pauli X applied to the qubit labelled ℓ.

An advantage of our encoding for `choice` is that the selected swaps require only single-qubit controls. An obvious disadvantage is the unnecessary space overhead. Although one can save space with a binary encoding, the resulting operations become somewhat more complicated and hence come at an increased time cost. Our choice of encoding is made for simplicity.

a. *Prepare `choice`*

Our preparation procedure has two stages. First, we prepare an alternative unary encoding of the state

$$|W_k\rangle := \frac{1}{\sqrt{k+1}} \sum_{\ell=0}^{k} |\ell\rangle \,, \tag{D3}$$

which we name for its resemblance to the W-state $\frac{1}{\sqrt{3}}(|001\rangle + |010\rangle + |100\rangle)$. Second, we translate the alternative unary encoding to our desired encoding. For a summary of the procedure, see Figure 8.

Next, we explain how to prepare $|W_k\rangle$ in the alternative unary encoding. The alternative encoding is

$$|\ell\rangle = \left(\prod_{\ell'=0}^{\ell} X_{\ell'} \right) |\texttt{null}\rangle \,. \tag{D4}$$

We can prepare $|W_k\rangle$ in this encoding with a cascade of controlled rotations of the form

$$R_\ell := \frac{1}{\sqrt{\ell+1}} \begin{pmatrix} 1 & -\sqrt{\ell} \\ \sqrt{\ell} & 1 \end{pmatrix} . \tag{D5}$$

Explicitly:

Apply X to qubit 0.
Apply R_k to qubit 1.
for $\ell = 1, \ldots, k-1$ **do**
  Apply $R_{k-\ell}$ controlled on qubit ℓ to qubit $\ell+1$.
end for

This is a total of $k+1$ gates, $k=1$ of which are applied sequentially.

Next we explain how to translate to the desired encoding. This is a simple procedure:

for $\ell = k, \ldots, 1$ **do**
  Apply NOT controlled on qubit ℓ to qubit $\ell - 1$.
end for

The total number of CNOT gates is k, and they must be applied in sequence. Thus the total gate count (and time-complexity) for preparing `choice` is $O(k) = O(\eta)$.

b. *Selected Swap*

We need to implement selected swaps of the form

$$\text{SELSWAP}_k := \sum_{c=0}^{\eta-1} |c\rangle\langle c|_{\texttt{choice}} \otimes \text{SWAP}(c,k)_{\texttt{target}}, \tag{D6}$$

where the $\text{SWAP}(c,k)$ operator acts on either `target` = `index` or `target` = `input`. Here the state of the `choice` register *selects* which entry in the `target` array is to be swapped with entry k. Our unary encoding of the `choice` register allows for a simple implementation of SELSWAP; see Figure 9.

Observe that only the first $k+1$ subregisters are involved of each `choice`, `index` and `input`, respectively. Also observe that, for each $i = 0, 1, \ldots, k$, `index`$[i]$ is of size $\lceil \log \eta \rceil$ whereas `input`$[i]$ is of size $\lceil \log N \rceil$. Hence, the circuit actually consists of $k\lceil \log \eta \rceil + k \lceil \log N \rceil$ ordinary 3-qubit controlled-SWAP gates that for the most part must be executed sequentially. As $\eta \leq N$, we report $O(\eta \log N)$ for both gate count and depth.

c. *Applying the controlled-phase*

Applying the controlled-phase gate is straightforward. We select a target qubit in the `input` register – it does not matter which. Then, for each $\ell = 0, 1, \ldots, k-1$, we apply a phase gate controlled on position ℓ of `choice` to the target qubit. The result is that `input` has picked up a phase of (-1) if `choice` specified a value strictly less than k. The total number of gates is $k = O(\eta)$, while the depth can be made $O(1)$.

d. *Resetting `choice` register*

The reason we execute swaps on both `index` and `input` is to enable reversible erasure of `choice` at the end of each Fisher-Yates block. This is done by scanning `index`

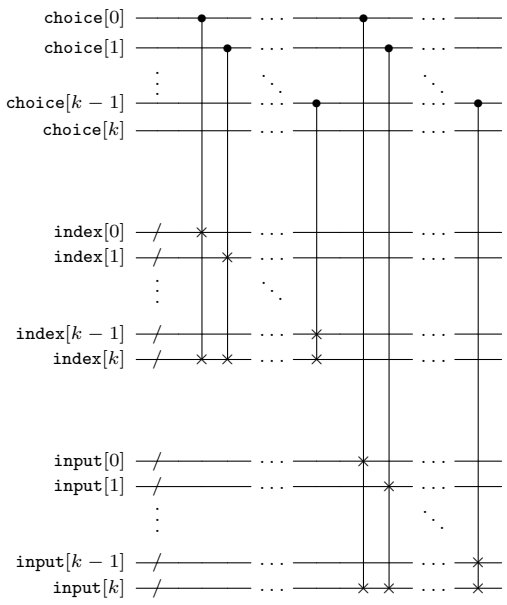

FIG. 9. Implementation of the two selected swaps $\textsc{SelSwap}_k$ as part of FY_k, with the unary-encoded `choice` as the control register and `index` and `input` as target registers, respectively. As each wire of the target registers stand for several qubits, each controlled-SWAP is to be interepreted as many bitwise controlled-SWAPs.

to find out which value of k was encoded into `choice`. In general, we know that step k sends the value k to position ℓ of `index`, where ℓ is specified by the `choice` register. We thus erase `choice` by applying a NOT operation to `choice`$[\ell]$ if `index`$[\ell] = k$. This can be expressed as a multi-controlled-NOT, as illustrated by an example in Figure 10. The control sequence of the multi-controlled-NOT is a binary encoding of the value k.

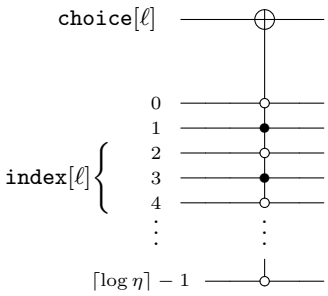

FIG. 10. Circuit for resetting `choice` register as part of iteration block FY_k. In this example $k = 10$. It consists of a series of multi-fold-controlled-NOTs, employing the ℓ-th wire of `choice` and the ℓ-th subregister `index`$[\ell]$ of size $\lceil \log \eta \rceil$, for each $\ell = 0, \ldots, k$. Note that the multi-fold-controlled-NOT is the same for all values of ℓ. The control sequence is a binary encoding of $k = 10$. The NOT erases `choice`$[\ell]$ if `index`$[\ell] = k$.

For compiling multiple controls, see Figure 4.10 in [58]. Each $\lceil \log \eta \rceil$-fold-controlled-NOT can be decomposed into a network of $O(\log \eta)$ gates (predominantly Toffolis) with depth $O(\log \eta)$. Because the $k+1$ multi-fold-controlled-NOTs (for $\ell \leq k \leq \eta - 1$) can all be executed in parallel, resetting `choice` register thus requires a circuit with $O(\eta \log \eta)$ gates but only $O(\log \eta)$ depth.

3. Disentangling `index` from `input`

The last task is to clean up and disentangle `index` from `input` by resetting the former to the original state $|0\rangle^{\otimes \eta \lceil \log \eta \rceil}$ while leaving the latter in the desired antisymmetrized superposition. This can be achieved as follows.

We compare the value carried by each of the η subregisters `input`$[\ell]$ (labeled by position index $\ell = 0, 1, \ldots, \eta - 1$) with the value of each other subregister `input`$[\ell']$ ($\ell' \neq \ell$), thus requiring $\eta(\eta - 1)$ comparisons in total. Note that these subregisters of `input` have all size $\lceil \log N \rceil$. Each time the value held in `input`$[\ell]$ is larger than the value carried by any other of the remaining $\eta - 1$ subregisters `input`$[\ell']$, we *decrement* the value of the corresponding ℓ^{th} subregister `index`$[\ell]$ of `index` *by* 1. In cases in which the value carried by `input`$[\ell]$ is smaller than `input`$[\ell']$, we do not decrement the value of `index`$[\ell]$. After accomplishing all the $\eta(\eta-1)$ comparisons within the `input` register and controlled decrements, we have reset the `index` register state to $|0\rangle^{\otimes \eta \lceil \log \eta \rceil}$ while leaving the `input` register in the antisymmetrized superposition state.

Each comparison between the values of two subregisters of `input` (each of size $\lceil \log N \rceil$) can be performed using the comparison oracle introduced in Appendix C 2. The oracle's output is then used to control the '*decrement by 1*' operation, after which the oracle is used again to uncompute the ancilla holding its result. The comparison oracle has been shown to require $O(\log N)$ gates but to have only circuit depth $O(\log \log N)$.

Decrementing the value of the $\lceil \log \eta \rceil$-sized index subregister `index`$[\ell]$ (for any $\ell = 0, 1, \ldots, \eta - 1$) by the value 1 can be achieved by a circuit depicted in Figure 11. Each such operation involves a total of $\lceil \log \eta \rceil$ multi-fold-controlled-NOTs. More specifically, it involves n-fold-controlled-NOTs for each $n = \lceil \log \eta \rceil - 1, \ldots, 0$. Note that each must also be controlled by the qubit holding the result of the comparison oracle. When decomposing each of them into a network of $O(n)$ Toffoli gates using $O(n)$ ancillae according to the method provided in Figure 4.10 in [58], the majority of the involved Toffoli gates for different values of n effectively cancel each other out. The resulting cost is only $O(\log \eta)$ Toffolis rather than $O\left(\log^2 \eta\right)$, at the expense of an additional space overhead of size $O(\log \eta)$. However, there is no need to employ new ancillae. We can simply reuse those qubits that previously composed the `choice` register for this purpose, as the latter is not being used otherwise at this stage any more.

Putting everything together, the overall circuit size for this step amounts to $O\left(\eta(\eta - 1)\left[\log N + \log \eta\right]\right)$ predom-

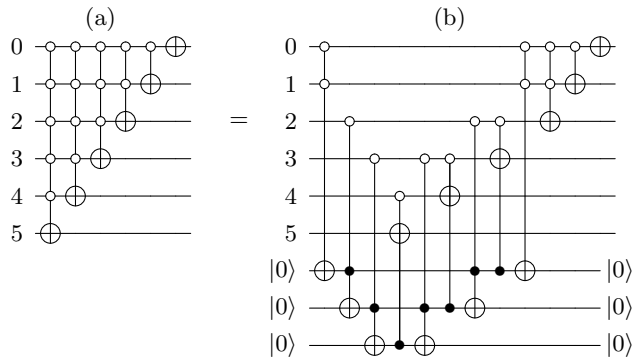

FIG. 11. Circuit implementing '*decrement by 1*' operation, applied to `index`$[\ell]$ subregisters of size $\lceil \log \eta \rceil$. (a) Example for $\eta = 64$. (b) Decomposition into a network of $O(\log \eta)$ Toffoli gates using $O(\log \eta)$ ancillae.

inantly Toffoli gates, which can then be further decomposed into CNOTs and single-qubit gates (including T gates) in well-known ways. Because $\eta \leq N$, we thus report $O(\eta^2 \log N)$ for the overall gate count for this step, while its circuit depth is $O\left(\eta^2 \left[\log \log N + \log \eta\right]\right)$.