npj | Quantum Information

www.nature.com/npjqi

ARTICLE OPEN

A quantum solution for efficient use of symmetries in the simulation of many-body systems

Albert T. Schmitz[1,2*] and Sonika Johri[2]

A many-body Hamiltonian can be block-diagonalized by expressing it in terms of symmetry-adapted basis states. Finding the group orbit representatives of these basis states and their corresponding symmetries is currently a memory/computational bottleneck on classical computers during exact diagonalization. We apply Grover's search in the form of a minimization procedure to solve this problem. Our quantum solution provides an exponential reduction in memory, and a quadratic speedup in time over classical methods. We discuss explicitly the full circuit implementation of Grover minimization as applied to this problem, finding that the oracle only scales as polylog in the size of the group, which acts as the search space. Further, we design an error mitigation scheme that, with no additional qubits, reduces the impact of bit-flip errors on the computation, with the magnitude of mitigation directly correlated with the error rate, improving the utility of the algorithm in the Noisy Intermediate Scale Quantum era.

npj Quantum Information (2020)6:2 ; https://doi.org/10.1038/s41534-019-0232-1

INTRODUCTION

As several quantum computing platforms become available for general use, finding practical applications for quantum computers is a key driver for the development and adoption of quantum computing technology. Additionally, since the field is expected to remain in the Noisy Intermediate Scale Quantum (NISQ) era[1] for the next few decades, designing error mitigation strategies for these algorithms is essential. In this paper, we identify a new application for quantum computers, as well as show how the algorithm should be implemented in the NISQ era.

Much of the excitement around quantum computing started with the introduction of two algorithms: Shor's factorization algorithm[2] and Grover's search algorithm.[3] Though the former represents the paradigmatic example of quantum speed up, the latter has been criticized as often only nominally showing speed-up. The criticism stems from the fact that although the oracle-query scaling is polynomially reduced, any quantum oracle which contains all the information of the database must scale with the size of the database.[4] This suggests we must look to problems where the oracle in Grover's search can be applied efficiently, treating it as a means to invert a Boolean function.

Dürr and Høyer[5] suggested a use for Grover's algorithm as a method to find the minimal element of a database. The general idea is to hold the best-known minimum value and search for a member less than that. If a better value is found, the best-known value is updated and the process is repeated for a set number of oracle calls. Assuming the oracle can be efficiently implemented, such a process might not be ideal in all cases as it still scales exponentially compared to approximation schemes such as adiabatic evolution and related minimization processes such as quantum approximate optimization algorithm (QAOA).[6] However, as the names suggest, these are only approximate methods. Furthermore, adiabatic evolution is sensitive to phase transitions due to a closing gap, and QAOA may require significant classical computational overhead. These limitations ultimately stem from the fact that such methods are sensitive to not just order, but also 'distance.' Grover minimization (Gmin) on the other hand is only dependent on the order. It treats the minimum the same whether it is separated from the next largest value by 1 or 100. This suggests that in special cases where an exact minimum is required or where we wish to ignore distance (or there is no notion of distance), Gmin is a good alternative.

We present one such problem which occurs in the simulation of strongly correlated materials or quantum chemistry problems where one might perform an exact diagonalization calculation. A many-body Hamiltonian often contains several symmetries which might represent spin symmetries, translation symmetry or various other discrete point-symmetries such as an n-fold rotation or reflection. Collectively, these symmetries can be formalized as a discrete group. One can leverage these symmetries by using group representation theory to block-diagonalize the full Hamiltonian[7] in a symmetry-adapted basis, making the remaining diagonalization computationally cheaper.

However, to calculate the block-diagonal matrix elements, each of the original basis states must be associated to an orbit representative which, for convenience, is chosen by labeling all basis states with a single integer value, and the orbit representative is defined as the element with the smallest integer label. One must also know the group operator connecting a basis state to its representative.[8] For large systems, the Hamiltonian cannot be stored explicitly but is calculated on-the-fly during diagonalization which means the matrix elements need to be computed over and over during the computation. For this, one has to either store the representative corresponding to each element in the original basis explicitly, which becomes costly in terms of memory, or calculate them on-the-fly. Thus, finding the orbit representative has become a serious bottleneck for using symmetries in exact diagonalization problems. For large spin systems, special-purpose hardware such as FPGAs have been considered to ease this bottleneck.[9] In some cases where distributed memory systems are used for the diagonalization calculation, the overhead of the symmetry adapted basis is so large that the authors abandon the symmetry-based approach altogether.[10] A technique for addressing this bottleneck for spin-systems with translational symmetry

[1]Department of Physics and Center for Theory of Quantum Matter, University of Colorado, Boulder, CO 80309, USA. [2]Intel Labs, Intel Corporation, Hillsboro, OR 97124, USA. *email: albert.schmitz@colorado.edu

Published in partnership with The University of New South Wales

npj nature partner journals

is proposed in refs [8,11] using a divide & conquer method based upon sub-lattice coding. This splits the costs between memory and computational time, by reducing both by a polynomial amount. In this paper, we consider the use of Gmin as a quantum solution to this problem. As for its use in the NISQ era, we consider the full circuit implementation for a benchmark case as well as the effects of error on the performance of the algorithm. This includes an error-mitigation scheme based on real-time post-selection on measurement results between coherent steps of the algorithm.

RESULTS AND DISCUSSION

The use of Gmin as applied to this problem results in a quadratic speed-up over the classical algorithms, and requires virtually no classical memory and relatively little quantum memory. See Table 1 for a cost comparison between classical and quantum methods. We also improve upon the textbook version of Gmin to optimize the number of oracle calls and reduce the number of qubits required to implement the oracle. Furthermore, we show that for many reasonable problem instances, the oracle is poly-log in the size of the group and dimension of the Hamiltonian's Hilbert space assuming the group action generators can be efficiently simulated on a quantum computer, making this a viable use for Grover's algorithm in practical applications.

Our error mitigation strategies result in a marked improvement in the algorithm's performance, which can be shown theoretically (see section 'Performance of AEM Gmin') and numerically (see sections 'Similation of error mitigation strategies' and 'Simulation of realistic hardware'). In particular, error mitigation increases performance in the presence of bit-flip error by changing the dependence of probability of success (P_{success}) on the deviations from ideal error rate scaling ($1/\delta$) from exponential degradation to polynomial degradation for small enough error rates. This increases the stability of the algorithm's performance to variations in error rates without the need to manually increase the number of oracle calls. Figure 7 demonstrates this numerically by varying error rates for a set problem instance, while Fig. 8 shows a similar stability with an increase in the size of the problem instance for a set error rate given by realistic hardware parameters for superconducting qubits.

We have thus identified a new application for the Gmin algorithm, and provided a full quantum solution for the problem. Since Grover's search often comes with the caveat of not having an efficiently implementable oracle, our work is notable for finding a practical use for Grover's algorithm as the oracle is expected to scale poly-logarithmically with the size of the group. We have discussed both the structure of the algorithm and refinements to the original version, as well as a full gate decomposition for the simplest group given by modular addition. We discussed how we can leverage the intermediate measurement steps to mitigate the effects of error, increasing the likelihood of the algorithm being useful in the NISQ era.

The algorithm discussed is far more general than what has been presented here. In addition to being a sub-routine in classical exact diagonalization, our algorithm could also be called by a larger quantum algorithm, which is performing a simulation of a many-body quantum system using symmetry-adapted basis states. Moreover, we achieve a reasonably sized oracle by leveraging the structure of the group, whereas the unstructured nature of the search is encapsulated in the arbitrary labeling of positions/basis states. In a similar fashion, we can envisage using Gmin to find/prepare the ground state of some Hamiltonian. In such a case, one leverages the structure of Hamiltonian dynamics by replacing the group action operator with phase estimation.[12] Furthermore, using Gmin as a sub-routine in classical exact diagonalization is an example of the power of interfacing quantum and classical machines for hybrid algorithms, while the error mitigation scheme we have designed is also likely to be generally applicable to oracles using ancilla qubits. We hope to explore this more in future work.

METHODS

Overview of the problem and the quantum solution

We first briefly review symmetry-adapted basis states and how their matrix elements are calculated following refs: [7,8] For a given many-body problem instance, Let H be the Hamiltonian. We then characterize its symmetries by operators $g \in G$, such that

$$[H, g] = 0. \tag{1}$$

From the group and its associated representation theory, we define the symmetry-adapted basis states as

$$|v_a\rangle \propto \sum_{g\in G} \chi(g)^*_a g|v\rangle, \tag{2}$$

where a indexes some one-dimensional representation of G, $\chi_a(g)$ is the character of the a^{th} representation evaluated at g and $|v\rangle$ are the original "position" basis states, such that the action of g on the basis states is $g|v\rangle = |gv\rangle$. One can see that two symmetry-adapted basis states $|v_a\rangle, |u_a\rangle$ are equal (once normalized) so long as $|u\rangle \in \text{orbit}(|v\rangle)$, where $\text{orbit}(|v\rangle)$ is the set of all basis elements connected to $|v\rangle$ by a group element. Therefore, each block of the Hamiltonian in this basis is characterized by just the representation index, with the states in each block represented by unique orbits, so we can choose a single representative $|\tilde{v}\rangle$ for each orbit. For simplicity, let us assume the group action is free, which is to say $gv = v$ if and only if g is the identity element. Then all states have the same normalization constant up to phase, $\mathcal{N}$. Since $\sum_{g\in G}\chi_a(g)\chi^*_a(g) = |G|$, we find that $\mathcal{N} = \frac{1}{\sqrt{|G|}}$. We can now calculate the matrix elements of H for a

Table 1. List of the different methods for solving the group representative problem.

Method	Cl. Mem	Q Mem	Time
Look-up	$\mathcal{O}(\|V\|)$	0	$\mathcal{O}(1)$
On-the-fly	$\mathcal{O}(1)$	0	$\mathcal{O}(\|G\|C(G))$
Divide & conquer	$\mathcal{O}(\|V\|^{\frac{1}{m}})$	0	$\mathcal{O}(\|G\|C(G)) - \mathcal{O}(C(G))$[a]
Gmin	$\mathcal{O}(1)$	$\mathcal{O}(2\log\|V\| + \log\|G\|)$	$\mathcal{O}(\sqrt{\|G\|}(\mathcal{C}(\tilde{G}) + \text{polylog}(\|V\|)))$

We generally expect that $\log|G| \ll \log|V| \leq |G| \ll |V|$. $C(G)$ is the classical cost to calculate the action of G on an arbitrary v, while $\mathcal{C}(G)$ is the quantum cost to calculate the action of G on an arbitrary v; m represents the number of sublattices in the divide & conquer method

[a]In general, divide & conquer does reduce the time cost versus on-the-fly. For a general group which satisfies the necessary properties required by divide & conquer (see main text), the most we can assume is that the search is over a subgroup of size $\mathcal{O}(|G|)$, but with a smaller coefficient. However, in many important cases, most notably translation groups, this search for most v is of size $O(1)$; see ref. [8] for details

given block via

$$\begin{aligned}\langle\tilde{v}_a|H|\tilde{u}_a\rangle &= \tfrac{1}{|G|}\sum_{g_1,g_2\in G}\chi_a(g_1)\chi_a^*(g_2)\langle\tilde{v}|g_1^{-1}Hg_2|\tilde{u}\rangle\\ &= \tfrac{1}{|G|}\sum_{g\in G}\sum_{g_2\in G}\chi_a(g_2g^{-1})\chi_a^*(g_2)\langle g\tilde{v}|H|\tilde{u}\rangle\\ &= \sum_{g\in G}\chi_a(g)\langle g^{-1}\tilde{v}|H|u\rangle\\ &= \sum_{v\in\mathrm{orbit}(\tilde{v})}\chi_a(g_v)\langle v|H|\tilde{u}\rangle,\end{aligned} \tag{3}$$

where we have used the fact that all members of G commute with the Hamiltonian, $\chi_a(g)$ is a one-dimensional representation of the group and define g_v such that $g_v v = \tilde{v}$. As we can see, one needs g_v to calculate the appropriate character. In practice, one calculates the action of H on the representative state $|\tilde{u}\rangle$, then sorts all coefficients of the resulting vector according to the orbits to form the appropriately weighted sum for each orbit. If the group action is not free, then one also has to calculate and store the normalization factors which also enter the sum.

Once obtained, one then uses these symmetry-reduced block-diagonalized matrix elements and applies techniques such as exact diagonalization or the Lanczos algorithm[13] to each of these blocks.

In the rest of this section, we describe the problem of finding the group orbit representative with some comments on the classical methods which are used to solve it. We then propose a quantum method based on Gmin, which exponentially reduces the memory cost while yielding a quadratic reduction in computational time.

Orbit representative problem statement

With the above motivation, we formally state the orbit representative problem.

Problem statement: suppose we have some finite group G with a group action $G\times V \to V$ such that $(g,v)\mapsto gv$. We shall refer to V as the position set and its members positions, though they may not correspond to physical position, but rather index some basis set for a Hamiltonian's Hilbert space. Furthermore, we have some function $\mathrm{int}(\mathrm{v})$ which totally orders the set V. We assume int maps to the integer value used to label$v$ (What follows could be mapped to more exotic orderings including partial orders if the phase comparator discussed below can be generalized to the given ordering efficiently.). Define the orbit of a position $\mathrm{orbit}(v) = \{gv : \text{for all } g\in G\}$, which is represented by $\tilde{v}\in\mathrm{orbit}(v)$ such that for all $u\in\mathrm{orbit}(v)$, $\mathrm{int}(\tilde{v})\le\mathrm{int}(\mathrm{u})$, i.e. it is the smallest element.

Given a member $v\in V$, find the orbit representative $\tilde{v}$ as well as the group element which gives that representative, i.e find g_v such that $g_v v = \tilde{v}$.

Note that based on the application of this problem from the last section, a near-minimum value for the orbit representative is not sufficient; we need the exact minimum. In the general case, one expects that $\log|G| \ll \log|V| \le |G| \ll |V|$. Table 1 gives a list of the solutions to this problem including Gmin and compares the costs. We denote the classical time complexity cost of computing the group action on an arbitrary member of V by $C(G)$ and in general, the quantum time-complexity cost of implementing an operator A on a quantum computer as $\mathcal{C}(A)$.

Classical solutions

There are three classical means of addressing this problem:

1. Look-up: Store orbit representatives corresponding to every element in V and connecting group elements in a look-up table. This can then be efficiently searched when needed, but it requires $\mathcal{O}(|V|)$ amount of memory.
2. On-the-fly: When needed, calculate the full orbit to find the smallest element and the connecting group element. This is efficient in terms of memory, but the computation scales as $\mathcal{O}(|G|)$.
3. Divide & conquer: There exist sub-lattice coding methods,[8] which allow one to split the costs between memory and computation (see Table 1 for these costs). In this method, one stores the "partial" representative for one of the sublattices, then searches over a coset of a sub-group which fixes this partial representative to find the overall representative.

While the divide & conquer method represents a significant reduction in the resources needed, this bottleneck can still be prohibitively expensive. Divide & conquer also requires some assumptions on the group such as it is a sub-group of the permutation group on local spins or degrees of freedom and that it possesses a property referred to as sublattice stability in ref. [8] (Sublattice stability can be regarded as the property that there exists a factorization of the group into a direct product, where one factor permutes sublattices and the second factor permutes the local degrees of freedom uniformly within each sublattice). Our method makes no such assumptions. Furthermore, because divide & conquer still requires what is essentially a search over a sub-group, in principle, the quantum methods presented below could be used in conjunction with divide & conquer to achieve a quadratic reduction in the time cost with only a logarithmic amount of quantum memory. To the best of our knowledge, no one has considered using quantum methods for solving this problem as we discuss in the next section.

Overview of the Gmin algorithm

In this section, we look to use the Gmin algorithm to solve the problem. We first review the algorithm as given in ref. [5] and then adapt it for this problem, which includes modifications to optimize the memory and time costs.

Gmin utilizes the function $f_v : G\to V$ such that $g\mapsto f_v(g) = \mathrm{int}(gv)$ acting on an unsorted database of $|G|$ items; g acts as an index and we want to find the index which points to the smallest value in f_v. To encode the group, we also introduce an index on the group elements $g : \mathbb{N}_{<|G|}\to G$ such that $x\mapsto g(x)$ (For notation convenience, we equivocate f_v with $f_v\circ g$ and we mean the latter throughout the remainder of the paper.). Then the number of bits (qubits) needed to index all members of the group is $m = \mathcal{O}(\log|G|)$. The original algorithm proceeds as follows:

Let a be some real, positive number, which we refer to as the oracle budget parameter from which we define $a\sqrt{|G|}$ as the oracle budget. Using two quantum registers each of size m (referred to as the group registers), choose an index $0<y<|G|-1$ randomly, and repeat the following, using no more than $a\sqrt{|G|}$ Grover steps:

1. Initialize the two registers in the state $\left(\frac{1}{\sqrt{|G|}}\sum_x|x\rangle\right)|y\rangle$,
2. Mark all x in the first register such that $f_v(x)<f_v(y)$,
3. Apply a "Grover search with an unknown number of marked elements" (Gsun)[14] to the first register and
4. Measure the first register with outcome y'; if $f_v(y')<f_v(y)$, $y\leftarrow y'$.

It is argued in the reference that for $a=\frac{45}{2}$, the second register holds the minimum value with a probability of at least 50%. Below, we discuss how to relate the success rate and a using numerical methods. Section I of the Supplementary Information gives a modified analytic derivation such that one finds a better value of $a=\frac{45}{8}$ to achieve a success rate of at least 50%.

To make this algorithm more explicit, we must address how to implement the second and third steps, which is equivalent to a method for implementing Gsun and its oracle. In general for Grover search, if the number of marked elements is known, one can apply the exact number of Grover steps, as embodied in the unitary operator Grov shown in Fig. 1a, to reach one of the marked states with high probability. However, this probability is not monotonic with the number of oracle calls. One can "overshoot" the target state and reduce the probability of reaching the answer with additional oracle calls. Thus, not knowing the number of marked elements could be problematic if we do not include some additional procedures. We refer those unfamiliar with Grover's search algorithm to refs [3,12] for details. Reference[14] provides a solution given by Gsun. Gsun iterates the search and randomly draws the number of Grover steps from a running interval. Those authors prove that the probability of selecting a marked element is asymptotically bounded below by $\frac{1}{4}$, thus insuring we can find a marked element with probability greater that 50% after a number of oracle calls that still scales as $\sqrt{|G|}$.

To mark elements as in step two, we must define the oracle. According to refs, [3,14] marking an element means the oracle produces the action on any computational basis state $|x\rangle$,

$$\mathrm{Oracle}|x\rangle = \begin{cases}-|x\rangle & \text{if } x \text{ is marked,}\\ |x\rangle & \text{otherwise}\end{cases}. \tag{4}$$

Note the second step requires we calculate $f_v(x)$ and $f_v(y)$, which implies we also require quantum registers to hold these values. There may exist multiple methods for implementing such an oracle, but the simplest and perhaps cheapest method for our problem is to further hold the value v in a quantum register of size $n=\mathcal{O}(\log|V|)$, which we refer to as the first position register. Furthermore, we replace the second group register with a

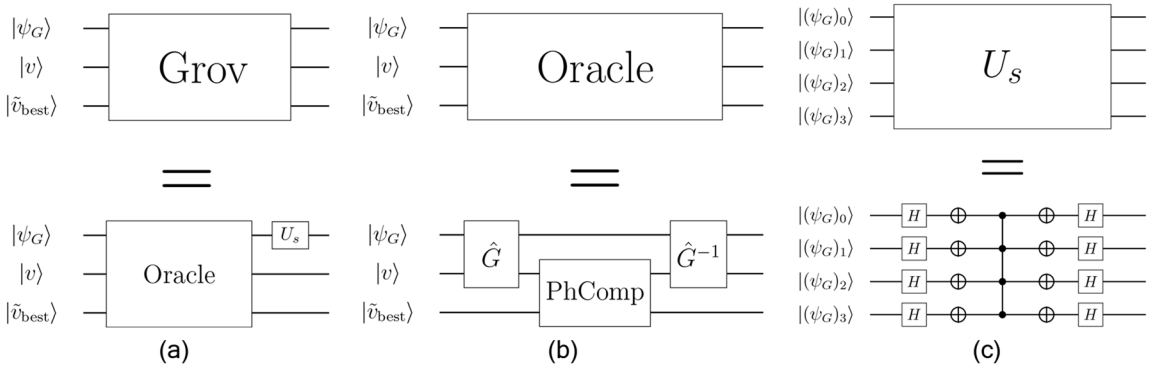

Fig. 1 Grov broken down into high-level components. a Circuit diagram for Grov. **b** Circuit diagram for our proposed oracle. **c** Circuit diagram for U_s for $|G| = 2^4$.

second position register of size n. So our method is not to store the best-known value for the group index (y in the above algorithm) as was done in previous implementations of Gmin, but rather store $\tilde{v}_{\text{best}} = f_v(y)$ in a quantum register. y can then be stored classically and updated when $\tilde{v}_{\text{best}}$ is updated. This innovation reduces the number of gates and qubits required for the oracle. The oracle is then implemented as follows: We first implement the group action operator $\hat{G}$ on the group register and the first position register which has been initialized with v such that

$$\hat{G}|x\rangle|v\rangle = |x\rangle|g(x)v\rangle. \tag{5}$$

We then apply a quantum circuit that in general acts on two quantum registers of equal size such that it applies a negative sign to the state if the computational basis state of the first register is less than that of the second. We refer to this circuit as phase comparator (PhComp), which has the behavior

$$\text{PhComp}|a\rangle|b\rangle = \begin{cases} -|a\rangle|b\rangle & \text{if } a < b, \\ |a\rangle|b\rangle & \text{otherwise} \end{cases}. \tag{6}$$

So after applying the group action operator, we apply PhComp to the two position registers, and then uncompute the group action operator. This completes the oracle as show in Fig. 1b. To complete a single step of Grov (Fig. 1a), we then apply the usual reflection operator defined as

$$U_s = I - 2|s\rangle\langle s| = V(I - 2|0\rangle\langle 0|)V^\dagger, \tag{7}$$

where $|s\rangle = \frac{1}{\sqrt{|G|}}\sum_x |x\rangle$ and V is any unitary such that $V|0\rangle = |s\rangle$.

If we unpack Gsun and integrate this into our modified version of Gmin, the psuedo-code flow of the algorithm is shown in Algorithm 1.

Algorithm 1 Grover minimization algorithm.

1: Allocate QRegister $|\psi_G\rangle$ of size m
2: Allocate QRegister $|\psi_1\rangle$ of size n
3: Allocate QRegister $|\psi_2\rangle$ of size n
4: $v_{\text{best}} =\leftarrow v, x_{\text{best}} \leftarrow 0, c \leftarrow 0, t \leftarrow 1$
5: **while** $c < \alpha\sqrt{|G|}$ **do**
6: $p \leftarrow \text{rand}(0, t-1)$
7: $c \leftarrow c + p + 1$
8: Initialize $(|\psi_G\rangle \otimes |\psi_1\rangle \otimes |\psi_2\rangle \leftarrow |0\rangle \otimes |v\rangle \otimes |v_{\text{best}}\rangle)$
9: $V|\psi_G\rangle$
10: $(\text{Grov})^p |\psi_G\rangle |\psi_1\rangle |\psi_2\rangle$
11: $\text{Measure}(x \leftarrow |\psi_G\rangle)$
12: **if** $f_v(x) < v_{\text{best}}$ **then**
13: $v_{\text{best}} \leftarrow f_v(x)$
14: $x_{\text{best}} \leftarrow x$
15: $t \leftarrow \max(1, \beta t)$
16: **else**
17: $t \leftarrow \min(\gamma t, \sqrt{|G|})$
18: **end if**
19: **end while**
20: **return** $v_{\text{best}}, x_{\text{best}}$

Note that we have chosen the initializing best guess $v_{\text{best}} = v$, as we assume v is effectively random. Also, we count the check step in line 12 as an effective oracle call so that the classical and quantum solutions can be more accurately compared. $\gamma \in \left(1, \frac{4}{3}\right)$ and $\beta \in [0, 1]$ are additional parameters that we use to minimize a. γ is discussed in ref. [14] and controls the rate of the exponential "ramp-up" for the parameter t, which in turn determines the ceiling of the random sampling for number of oracle calls used in the Grover search step of the algorithm. In principle, a large γ reduces the time to reach $t \sim \sqrt{|G|}$, which is optimal if v_{best} is near the minimum (the number of marked elements is small; the search takes longer). However if v_{best} is far from the minimum, γ being too large and $t \sim \sqrt{|G|}$ increases the chances that we apply too many oracle calls and dramatically overshoot a state of high overlap with a marked element. Thus, we need to balance the rate at which t increases by optimizing γ. β is a parameter that we introduce here. As the algorithm was originally written, after a better value of v_{best} is found in line 13 of Algorithm 1, Gsun effectively ends and on the next cycle is re-called. Gsun then assumes it knows nothing about how close we are to the minimum by resetting the value of t back to 1 (as would be the case for $\beta = 0$). However, we do know something, namely that we are closer to the minimum than the iteration before (the number of marked elements has decreased). Thus we do not need the ramp-up time for t which is only included to address when we are far from the minimum. By including the β parameter, we are looking to exploit this limited knowledge about the number of marked elements. We discuss the exact values chosen for these parameters in section 'Oracle budget and probability success'.

Circuit implementation of Grov and its cost

We now discuss a full circuit implementation of all subroutines of Grov. As $\hat{G}$ is specified by the problem instance, we only give an explicit implementation for the group G_{add}^N, which represents addition modulo $N = 2^n$ or translation on a cycle of N positions. Otherwise, we discuss a general strategy for more complicated realistic groups.

The simplest part of Grov to implement is the standard U_s operator as defined in Eq. (7). As discussed in ref., [14] if $|G| = 2^n$ for some n, then $V = H^{\otimes n}$ is given by the Hadamard gate acting on every qubit of the group register. The remaining reflection is implemented by a controlled π phase gate on a computational 0 input, i.e. apply NOT to all qubits and then apply the multi-controlled Z gate. Finally, we uncompute everything but the multi-controlled Z gate. An example of this circuit is shown in Fig. 1c. If $|G|$ is not a power of 2, we only have to modify the change of basis given by V to some other change of basis operator such as the quantum Fourier transform (QFT). The cost of the former is $\mathcal{C}(U_s) \sim \mathcal{O}(\log|G|)$ while the latter case scales as $\mathcal{C}(U_s) \sim \mathcal{O}(\log^2|G|)$ if we use QFT.

We next consider an implementation of PhComp as defined in Eq. (6) by considering a bitwise comparison of the input registers. We start with the most significant bit of the binary expansion of a computational input value and proceed to the least significant. At the i^{th} bit, we need to calculate two binary values, the first representing whether or not we should apply the π phase at the current bit, and the second representing whether or not we should continue to compare on the remaining lesser bits. That is, if the two bits differ, the value containing 1 is greater, so we need to prevent any additional phases from being apply on lesser bits. A truth table for this calculation is given in Table 2 for input bits a_i and b_i. From this, we find that $(\text{apply phase})_i = \bar{a}_i b_i$ conditioned on the truth (AND-ed with) all greater $(\text{continue})_j = \bar{a}_j \oplus b_j$ bits for $j > i$. So our method of implementing PhComp is to NOT all qubits of the first register a and then compare from

Table 2. Truth table used to form PhComp.

a_i	b_i	$(\text{continue})_i$	$(\text{apply phase})_i$
0	0	1	0
0	1	0	1
1	0	0	0
1	1	1	0

the most to least significant qubit. At the i^{th} qubit, we calculate $(\text{continue})_i$ on b_i using CNOT, but not before calculating $(\text{apply phase})_i$ in the phase with a multi-control Z gate between $\overline{a}_i$, b_i and all the b_j for $j > i$ (which now contain the (continue) bits). Finally, we uncompute the CNOT and NOT gates. An example circuit is shown in Fig. 2. Assuming the cost of a multi-control Z gate scales linearly with the number of controls, the cost of PhComp is $\mathcal{C}(\text{PhComp}) \sim \mathcal{O}(\log^2 |V|)$. However, if we have additional ancilla qubits available, we can use these to reduce $C(\text{PhComp}) \sim \mathcal{O}(\log |V|)$. See Section II of the Supplementary Information for details.

The form of the group action operator, $\hat{G}$, is entirely dependent on the group. We take the simplest case first which is an abelian group with a single cycle, whereby $g(x) = g^x$ for the group generator g. We assume we can form a circuit for the operator $\hat{g}$ acting on a position register which achieves

$$\hat{g}|v\rangle = |gv\rangle. \tag{8}$$

We then control $\hat{g}^{2^i}$ on the i^{th} qubit of the group register as show in Fig. 3a. This method can then be generalized to multi-cycle abelian groups by subdividing the group register so there is one subregister for each cycle and generate a circuit similar to Fig. 3a for each cycle. If the group is non-abelian, one has to consider a strategy for indexing powers of the generators and their order. For example, suppose the group is generated by two non-commuting operators g_1 and g_2. Each generator forms its own abelian subgroup so we can use the same strategy for them separately and with their own sub-group register. Furthermore, the order for applying these operators can be controlled by a single qubit, $|\text{order}\rangle$, using the circuit in Fig. 3b. If $|\text{order}\rangle = |0\rangle$, then the group operator applied is $g_1^{x_1} g_2^{x_2}$ and if $|\text{order}\rangle = |1\rangle$ then the group operator applied is $g_2^{x_2} g_1^{x_1}$. We note this may not be the most efficient method in terms of qubit use for the group register qubits. For example, if $x_2 = 0$, then the state of $|\text{order}\rangle$ does not matter and so there are redundant index states in the group register. This is also the case if there are redundancies in the order of non-zero powers of the generators, i.e. $g_1^{x_1} g_2^{x_2} = g_2^{x_2} g_1^{x_1}$, for some values of the indices. The most efficient method depends on the group, but we include this example to demonstrate that, in principle, one can handle non-abelian groups using roughly the same strategy as was used for abelian groups.

The scaling of $\hat{G}$ is highly dependent on the group being used, but it should be clear that in many reasonable cases, the scaling should be $\mathcal{C}(\hat{G}) \sim \log(|G|)\mathcal{C}(\hat{g})$, where we assume the generators can be implement at cost $\mathcal{C}(\hat{g}) \sim \mathcal{C}(\hat{g}^n) \sim \text{polylog}(|V|)$ for any power n. That is, implementing the power of a generator must not scale with that power. To demonstrate the importance of this, consider the single-cycle abelian case. If we implement $\hat{g}^2$ with two copy of $\hat{g}$ and so on for the other powers, then

$$\mathcal{C}(\hat{G}) \sim \mathcal{C}(\hat{g}) \sum_{k=1}^{\log|G|} 2^k \sim \mathcal{C}(\hat{g})|G|. \tag{9}$$

Clearly, this is not efficient, and our oracle scales with the size of the search space. However, if the implementation of the powers of $\hat{g}$ can be simplified so as to scale on the order of $\hat{g}$ or less, then we achieve our desired scaling $C(\hat{G}) \sim \mathcal{C}(\hat{g})\log|G|$. It is reasonable to believe this is possible in the general case. Suppose we take for granted the complexity of a quantum circuit corresponding to a periodic operator scales with the size of its period. $\hat{g}^2$ has half the period of $\hat{g}$ and $\hat{g}^4$ has half the period of $\hat{g}^2$ and so on. So one would expect that $\hat{g}$ is actually the most expensive power to implement.

To make this discussion more concrete, consider the example of the group representing addition mod $N = 2^n$ for some n which we denote G_{add}^N, i.e.

$$\hat{G}_{\text{add}}^N|x\rangle|y\rangle = |x\rangle|x + y\rangle, \tag{10}$$

where mod N is implicit. Implementing this operator using the methods discussed here (We are aware of better in-place adders, namely those which calculate the addition in the phase or via some ripple-carry scheme.[15] We stick to this less-efficient implementation of the adder as it imitates our more generic construction of $\hat{G}$.), one can use $\hat{g}_{\text{add}}$ consisting of a sequence of multi-control NOT gates as shown in Fig. 3c (where we recall that $\hat{g}_{\text{add}}|y\rangle = |y + 1\rangle$). It is clear that $\hat{g}_{\text{add}}^2$ is given by removing all gate action and control lines on the least significant bit, and so on for the other powers. For such a simple case, it is easy to see this simplification, but for a compiler which only moves commutative gates and considers local pattern matching, this dramatic reduction might go unexploited, so a manually optimized implementation might be preferred.

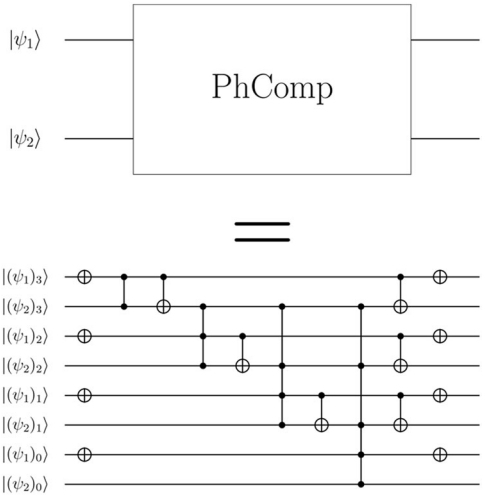

Fig. 2 Circuit diagram for PhComp. An explicit example of our circuit implementation of PhComp for $|V| = 2^4$.

It is worth considering the specific case of G for spin Hamiltonians as this is the most natural use for a quantum solution to this problem. The natural mapping of the problem would assign one qubit to each spin in the physical system, and most geometric symmetry generators such as those for translation or rotation (as opposed to spin symmetries), can be simulated by a $\hat{g}_{\text{spin}}$ consisting of swap gates. For example, translation on a spin chain would use a $\hat{g}_{\text{spin}}$ consisting of a cascade of nearest-neighbor swaps. Note that here and in general, $\mathcal{C}(\hat{g}_{\text{spin}}) \sim \mathcal{O}(|G_{\text{spin}}|) = \mathcal{O}(\log|V|)$, but this is because the group is already exponentially small in the size of V. So in general, we expect $\mathcal{C}(\hat{G}_{\text{spin}}) \sim |G_{\text{spin}}| \log|G_{\text{spin}}|$.

In terms of comparing costs with the classical on-the-fly method, we expect $C(G)$ to be of the order of $\mathcal{C}(\hat{G})$ in which case the quantum solution out performs the classical one. For a general group, the classical divide & conquer method has a smaller constant coefficient, so the quantum solution outperforms it for relatively larger group sizes, but always uses exponentially less memory.

Oracle budget and probability of success

To complete the algorithm, we need to determine the constants α, β and γ. As we have the exact solution for the probability of success for a single Grover search[3,14] (line 9–11 of Algorithm 1), we are able to simulate the classical parts of the algorithm using G_{add}^N as the group in order to determine the behavior of these parameters. Note that, without error, the oracle query complexity is unaffected by the details of the group (aside from its size), so the following results should be general. As we know the solution to the orbit representative problem for this trivial example, we can run the simulation until the correct answer is obtained. This allows us to empirically determine the probability of success as a function of the total number of calls. For a window of probabilities $P_{\text{success}} \in [0.2, 0.995]$ (We clearly do not care about probabilities less that 20% and beyond 99.5%, the rate of increase of the probability is hard to discriminate within our simulation), we find that the asymptotic form of the probability for large N is given by

$$P_{\text{success}} \sim 1 - \exp\left(-\frac{T^2}{a^2 N}\right), \tag{11}$$

where T is the number of oracle calls and a is the rate parameter which is a function of only β, γ and is empirically determined. By linearizing Eq. (11) with $\frac{1}{a}$ as the slope, we can calculate the rate parameter as is the case in

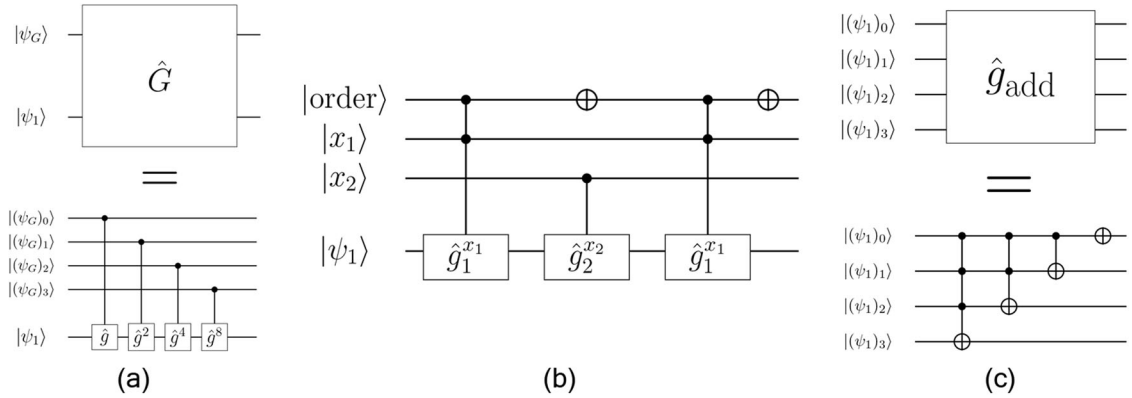

Fig. 3 Strategies for the circuit implementation of the group action operator, $\hat{G}$. a Circuit diagram for the group action operator $\hat{G}$ for a single-cycle abelian group generated by g and simulated by $\hat{g}$ for $|V| = 2^4$. **b** Method for implementing a group with two non-commuting generators. $|\text{order}\rangle$ is a single qubit which determines the order the generators are applied. **c** Circuit diagram for $\hat{g}_{\text{add}}$ for $|V| = 2^4$.

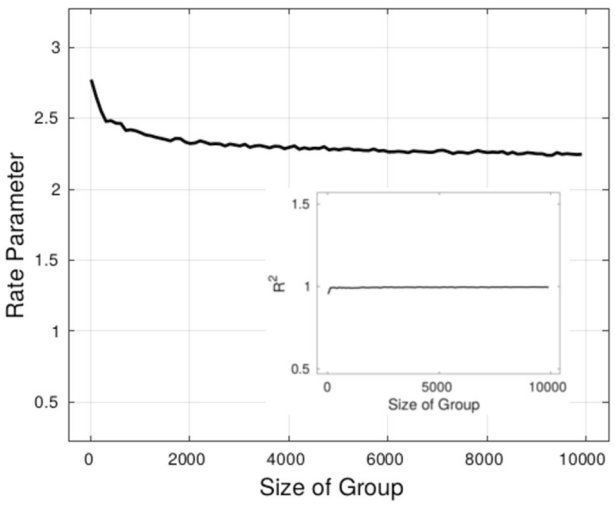

Fig. 4 Plot of the rate parameter as a function of group size from the simulation of the classical parts of Algorithm 1. The insert shows the R^2-value of the linear regression used to derive the rate parameter; $\beta = 0.95$, $\gamma = 1.15$.

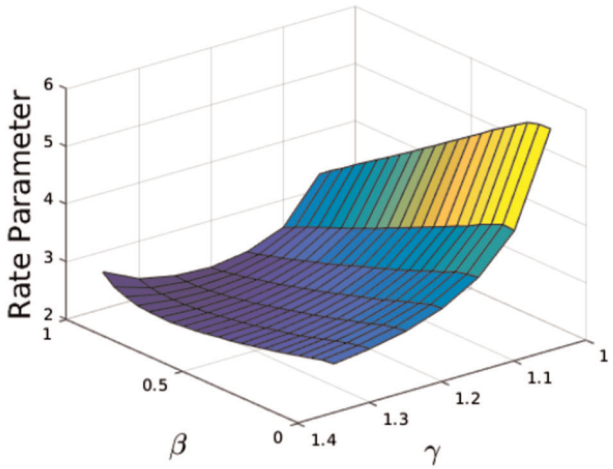

Fig. 5 Survey of the Rate Parameter. Plot of the rate parameter a as a function of β, γ.

Fig. 4 as well as demonstrate this is the correct asymptotic form. One can see that the R^2-value of the linear regression asymptotically approaches 1 and the rate parameter approaches a constant for fixed β, γ. This allows us to determine the oracle budget parameter a. For a given application, if we allow for a probable error in the solution of $\epsilon > 0$, then a is given by

$$a \sim a\sqrt{-\ln \epsilon}. \tag{12}$$

So we want to determine the values of β and γ such that we minimize a. Figure 5 shows a survey of a as a function of β and γ. From this, we have chosen $\gamma = 1.15$ and $\beta = 0.95$ as the near optimal values. This value of γ is near previously discussed values, where ref. [14] suggests $\frac{6}{5}$. However, β being near one suggests we gain a good deal of information knowing that the number of marked items has decreased from one call of Gsun to another. For comparison, if we use $\epsilon = 0.5$ and $a \approx 2$–4, the resulting oracle budget parameter is $a \approx 1.6$–3.3, which is a considerable reduction compared to $a \approx 5.6$ for the analytic value previously discussed. For applications which require a high probability of success, i.e. $\epsilon = 0.01$, we obtain $a \sim 4.3$–8.6.

Full simulation for a perfect quantum machine

To check the behavior of Algorithm 1, we implement a full quantum simulation using the Intel Quantum Simulator (Intel-QS)[16] and $G^{2^n}_{\text{add}}$ as our group for $n = 4$–8. This requires 12–24 qubits using no additional ancilla to reduce the depth of the quantum circuits. Although $G^{2^n}_{\text{add}}$ is not a useful problem instance, it does maximize the group size relative to the number of positions, i.e. $\log|G| = \log|V|$ and so this represents the most efficient benchmark using the fewest qubits. Furthermore just as with the simulation of the classical part of Gmin from the last section, knowing the correct answer allows us to avoid choosing an oracle budget, and instead run the algorithm until the solution is found to determine the probability of success as a function of total number of calls (Still, we do choose a hard stop of $a = \frac{45}{2}$, which as we established is on the high side for a reasonable oracle budget). As our simulation is exact, i.e. we are treating the quantum machine as perfect, the details of the group do not affect the results. All quantum subroutines are implemented according to the discussion in section 'Circuit implementation of Grov and its cost', where multi-controlled gates have been broken down to one- and two-qubit gates using methods from ref. [17] This was done to better simulate the algorithm acting on real hardware once noise is added in section 'Simulation of error mitigation strategies'.

Figure 6 shows the probability of success as a function of oracle calls, where the insert shows an effective rate parameter. We note that the probability in Eq. (11) is asymptotically correct in the limit of large N and as such, the curves for these smaller group sizes do not fit this form well. Instead, we define the effective rate parameter, a_{eff} via

$$\frac{1}{a_{\text{eff}}\sqrt{N}} = \text{avg}\left(\text{diff}_T\left(\sqrt{-\ln(1 - P_{\text{success}})}\right)\right), \tag{13}$$

where we treat P_{success} as a function of oracle calls, T, and diff_T is the difference between the expression in the brackets evaluated at successive values of T. Despite the poor fit, a_{eff} is still indicative of the trends. We then

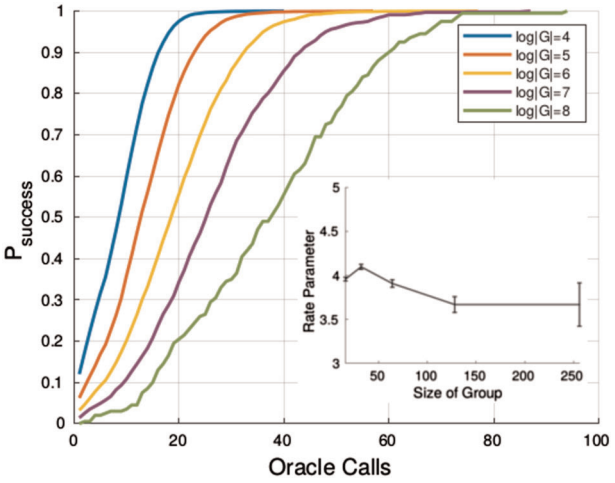

Fig. 6 Plot of the simulated value of P_{success} as a function of number of oracle calls for various group sizes. The insert shows the effective rate parameter from Eq. (13) for these different group sizes. The number of trials used is 10,000, 10,000, 4000, 1000 and 200, respectively. Error bars for the effective rate parameter are calculated according to Eq. (14).

determine error bars for a_{eff} via

$$\delta a_{\text{eff}} = \sqrt{\frac{N}{M}}\sigma_{\text{eff}} a_{\text{eff}}^2, \tag{14}$$

where σ_{eff} is the standard deviation of the expression which is averaged in Eq. (13) and M is the number of trials. From Fig. 6, we find the behavior as expected from the classical simulations. We notice, however, that the effective rate parameter is higher for the full simulation as compared to the classical simulation. Although not optimal, the effective rate parameter still suffices. For example, if we desired a 99% chance of success and we chose the rate parameter to be $a = 4$ ($a \approx 5.7$), then the oracle budgets would be 23, 32, 45, 64 and 91, respectively, for the group sizes shown in Fig. 6. From the figure, we see that we would achieve nearly or better than our target 99% chance of success.

Overview of error mitigation strategies

One of the benefits of Gmin is that the best-known value for the minimum is alway monotonically decreasing with the number of oracle calls. Unlike Grover search, this is true even for a faulty implementation on an imperfect quantum machine. Furthermore, vagaries of faulty implementation are partially compensated for by the classical random sampling of the number of oracle calls for any single coherent Grover search. Put a different way, though we allot a set oracle budget, not all these calls are implemented in a single coherent step. This suggests Gmin is a reasonable use for near-term, noisy hardware. Still, noise has its costs. In this section, we describe some strategies for mitigating the cost of errors. We then simulate some of these methods to determine their effectiveness.

We start by describing two error mitigation strategies. As mentioned, the approach to a solution is monotonic regardless of the error rates. Thus, the most obvious method is to simply increase the oracle budget, leaving all else the same, a method we refer to as static error mitigation (SEM). The obvious downside to this method is that the increase in the oracle budget would reasonably need to scale with the size of the system—assuming roughly independent error rates for each qubit—in which case, we may lose our quantum advantage. This is supported by analytic results on Grover search with a faulty oracle in refs, [18,19] where for certain toy error models, the polynomial quantum speed-up is either partially or entirely lost.

The other strategy takes advantage of the additional qubits which do not hold the search space. The two position registers are included only as a means of marking elements of the search space and implementing the oracle. As such, they should hold the same computational basis value at the beginning and end of a single call to Grov. This allows us to measure these registers without disturbing the coherence of the group register, which is responsible for the quantum speed-up. Moreover, any terms in the full state of the system (as expanded in the computational basis) which hold values in the position registers which differ from v and v_{best} are in error, and measuring the correct values projects the system back to an un-errored, or at least less-errored, state. Thus, we suggest the following: at the end of any call to Grov, measure the two position registers. If their measured values differ from that of the classically stored values v and v_{best}, we abort the remaining Grover steps on line 10 of Algorithm 1 and go back to step 8, for which the errored oracle calls do not count against our oracle budget. It is important to note that we do not randomly sample p again as this would introduce a bias toward smaller values of p as they are less likely to experience an error. We refer to this strategy as active error mitigation (AEM). This is because the total number of oracle calls, both errored and un-errored, is not fixed, but depends on the rate of error.

The downside of this method is that all the oracle calls up to the point an error is found still cost time, which is now wasted due to the error state. To mitigate this waste, before restarting the Grover search, we measure the group register and continue to check to see if a better value is found. To do so is practically free (up to one additional effective oracle call to perform the check) and it can only increase our chances of finding the minimum, even if by a minuscule amount. Moreover, simulations demonstrate the increase is significant. We refer to this as a measure-and-check strategy.

All together, the AEM version of the Gmin algorithm is presented in Algorithm 2. Note we have added a hard stop for total number of oracle calls as characterized by ℓ to avoid infinite run-time. Ideally AEM "protects" the probability of success for a fixed oracle budget and a large range of error rates. That is, P_{success} as a function of un-errored oracle calls (i.e. as a function of the c_1 count in Algorithm 2) takes the form of Eq. (11) with a rate parameter, which is only weakly dependent on the error rates. Again, the downside is the non-deterministic run-time, which can bloat if the error rate is too high.

Algorithm 2 Active-error-mitigation (AEM) Grover minimization.

1: Allocate QRegister $|\psi_G\rangle$ of size m
2: Allocate QRegister $|\psi_1\rangle$ of size n
3: Allocate QRegister $|\psi_2\rangle$ of size n
4: $v_{\text{best}} =\leftarrow v, x_{\text{best}} \leftarrow 0, \text{good} \leftarrow \text{true}$
5: $c_1 \leftarrow 0, c_2 \leftarrow 0, t \leftarrow 1$
6: **while** $c_1 < \alpha\sqrt{|G|}$ AND $c_2 < \ell|G|$ **do**
7: **if** good **then**
8: $p \leftarrow \text{rand}(0, t-1)$
9: **else**
10: good $\leftarrow$ true
11: **end if**
12: Initialize$(|\psi_G\rangle \otimes |\psi_1\rangle \otimes |\psi_2\rangle \leftarrow |0\rangle \otimes |v\rangle \otimes |v_{\text{best}}\rangle)$
13: $V|\psi_G\rangle$
14: **for** $i \in [1:p]$ **do**
15: $(\text{Grov})|\psi_G\rangle|\psi_1\rangle|\psi_2\rangle$
16: Measure$(v_1 \leftarrow |\psi_1\rangle, v_2 \leftarrow |\psi_2\rangle)$
17: **if** $v_1 \neq v$ OR $v_2 \neq v_{\text{best}}$ **then**
18: good $\leftarrow$ false
19: $c_2 \leftarrow c_2 + i + 1$
20: break **for**
21: **end if**
22: **end for**
23: Measure$(x \leftarrow |\psi_G\rangle)$
24: **if** good **then**
25: $c_1 \leftarrow c_1 + p + 1$
26: $c_2 \leftarrow c_2 + p + 1$
27: **end if**
28: **if** $f_v(x) < v_{\text{best}}$ **then**
29: $\tilde{v}_{\text{best}} \leftarrow f_v(x)$
30: $x_{\text{best}} \leftarrow x$
31: $t \leftarrow \max(1, \beta t)$
32: **else**
33: **if** good **then**
34: $t \leftarrow \min(\gamma t, \sqrt{N})$
35: **end if**
36: **end if**
37: **end while**
38: **return** $v_{\text{best}}, x_{\text{best}}$

Performance of AEM Gmin

In Section III of the Supplementary Information, we analyze the performance of AEM Gmin using a simple error model. Let the average qubit lifetime $\langle t\rangle$ (say the average between T_1 and T_2 as described below) scale as

$$\langle t\rangle \sim \frac{\delta}{4}\mathcal{C}(\mathrm{Grov})\sqrt{N}, \tag{15}$$

for some $\delta > 0$. Then we find that optimally (where the error model parameter, e, from Section III of the Supplementary Information is one) the probability of success for p AEM Grov calls, including measure-and-check when an error is found, is asymptotically

$$P^{(p)}_{\mathrm{success}} \sim \frac{\delta^2}{1+\delta^2}\left(\frac{(1-\sigma^p)}{2} + \sigma^p \sin^2((2p+1)\theta)\right), \tag{16}$$

where $\sigma \sim \exp\left(-\frac{\mathcal{C}(\mathrm{Grov})}{\langle t\rangle}\right) = \exp\left(-\frac{4}{\sqrt{N}\delta}\right)$ is the probability of having no error in a single call to Grov and $\sin^2(\theta) \sim \frac{1}{N}$. Recall that the probability of success in the absence of noise ($\delta \to \infty$) is $\sin^2((2p+1)\theta)$. Without measure-and-check after the error, this probability is degraded to $\sigma^p \sin^2((2p+1)\theta)$, in which case the probability of success is exponentially sensitive to the value of δ. With measure-and-check, the probability is only polynomially sensitive to δ. To demonstrate this, consider the case when we are searching for a single element, $p \sim \sqrt{N}$ and so $\sin^2((2p+1)\theta) \sim 1$. If $\delta = 4$, then the AEM probability of success with measure-and-check goes as $\frac{16}{17}\left(\frac{1-\exp(-1)}{2} + \exp(-1)\right) \approx 64\%$, which is reasonably better than the no-measure-and-check probability of $\exp(-1) \approx 37\%$. However, if $\delta = 1$, then the AEM probability is $\frac{1}{2}\left(\frac{1-\exp(-4)}{2} + \exp(-4)\right) \approx 25\%$ as compared to $\exp(-4) \approx 2\%$. If we go even further and take $\delta = \frac{1}{2}$, the AEM probability of success goes as $\frac{1}{10} = 10\%$ whereas without measure-and-check, it is negligible. So even though we need the coherence time to scale as $\sim\sqrt{N}$, AEM Gmin is far more forgiving for a smaller value of the coefficient δ. This is further demonstrated numerically in section 'Simulation for realistic hardware'.

The analysis given in Section III of the Supplementary Information is a general result for Grover search with the same measure-and-check AEM strategy. For AEM Gmin, the fact that the probability of success of a single search is necessarily degraded by noise means we still need to increase the oracle budget in order that the target overall probability of success is maintained. This is done automatically by not counting errored oracle counts.

Simulation of error mitigation strategies

To simulate noisy hardware, we used the error model included in the Intel-QS package, which is based upon the Pauli-twirling approximation error model.[20] In this model, before a gate is applied, a random single qubit rotation is applied to each qubit acted on by that gate. The error unitary is given by

$$U_{\mathrm{error}} = \exp\left(iv_x X + iv_y Y + iv_z Z\right), \tag{17}$$

where X, Y, Z are the single-qubit Pauli operators. v_x, v_y and v_z are parameters chosen at random from a Gaussian distribution whose variance grows with the time from the last gate action in units of the hardware-dependent parameters T_1, T_ϕ and T_2, respectively. As X, Y and Z are dependent on one another, the parameters are related by

$$\frac{1}{T_\phi} = \frac{1}{T_2} - \frac{1}{2T_1}. \tag{18}$$

Because T_1 is associated with the X Pauli operator which flips the computational state, we can think of T_1 as the "bit-flip" error rate. Likewise, T_2 is associated with the Z Pauli operator which applies a π phase, so we can think of this as the "phase-flip" error rate. To accurately accommodate for this non-deterministic, measurement-based algorithm, some modifications had to be made to the Intel-QS. See Section IV of the Supplementary Information for details.

Simulations for both SEM and AEM are shown in Fig. 7 for $\log|G| = 4, 5$ where we have fixed either T_1 or T_2 to be a large, effectively infinite constant and varied the other. This allows us to determine the effect of each kind of error. T_1 and T_2 are measured in units of the single-qubit gate time (SQGT); See Section IV of the Supplementary Information for details.

In terms of bit-flip error, we can see that AEM does protect the rate parameter over the values of T_1 shown in Fig. 7a, b as evidenced by the flatness of the curves for AEM, $T_2 = \infty$. However, AEM only partially protects the rate parameter against phase-flip error. This should not be surprising as phase error would persist even after the projection due to measurement at the end of a call to Grov. That is, phase error tends to accumulate in the superposition of the group register and is not corrected by the AEM strategy. Still looking at the SEM results, we see that the algorithm is altogether less susceptible to phase-flip error.

The protection of the rate parameter by AEM is important as it means our choice of the oracle budget parameter is less dependent on knowing the rate of error. However, the rate parameter is no-longer directly proportional to the run-time of the algorithm as errored calls to Grov are not counted against the oracle budget. Thus, we have to evaluate whether the total run-time is better or worse under AEM, which not only includes the errored calls, but also includes the time to perform the measurements. Figure 7c, d plots the average run-time as a function of either T_1 or T_2 for a fixed, large value of the other parameter. By average run-time, we mean the average over all trials of the total run-time (to find the correct answer) of the quantum computation cycles of the algorithm, including all measurements and gates, in units of the SQGT. This does not include time to perform the classical computation cycles of the algorithm (The variability in the average run-time is maximal as the time to reach the minimum can be zero if v happens to be the minimum. For this reason, we give no error bars on the average run-time). From this figure, we see that AEM does not bloat the run-time for bit-flip error and as desired, significantly decreases the run-time for small T_1 times. It also only adds a modest, roughly constant increase for phase-flip error. Note that for higher T_1 and T_2, there is a cross-over where SEM has a smaller average run-time. This is due to the additional time needed to perform the measurements, which is only a constant time increase for each call to Grov.

From this analysis, we see that AEM is always preferred over SEM as it both protects the rate parameter and decreases the run-time except for when coherence times are sufficiently high, in which case its cost is only a constant for each call to Grov.

Reducing phase-flip error

AEM is effective against bit-flip error, but less so for phase-flip error. Even though the algorithm is less susceptible to this kind of error, it is worth considering a method for reducing phase-flip error. This can be achieved using simple fault-tolerant methods. As we are only looking to correct one channel of error, we can use simple, essentially classical fault-tolerant error-correcting codes such as a repetition code.[21] It should be sufficient to use an error-correcting code on the group register only to reduce the qubit overhead. With enough physical qubits to form robust logical qubits, we could achieve an effective $T_2 \sim \infty$ in which case AEM should fully protect the rate parameter.

Simulation for realistic hardware

AEM Gmin requires interaction between quantum and classical instructions, but unlike similar hybrid computations such as decoding an error-correcting code or variational eigensolver (VQE), the classical computation cycles are simple and should not take a significant amount of time between coherent quantum steps. Thus, AEM Gmin could stand as a good test of real-time hybrid quantum-classical computation. For this reason, we simulate AEM Gmin with realistic T_1, T_2 times using the addition group of sizes $n = \log|G| = 4, 5$ and 6. To increase the chances of a successful run, we use the maximum number of ancilla qubits to reduce the depth of the circuit. So the total qubits used is $3n + (n-2) = 4n - 2$, or 14, 18 and 22, respectively, for our cases. Methods for using the ancilla to reduce the depth are give in Section II of the Supplementary Information. We used $T_1 = T_2 = 700$ SQGTs, which are extracted from ref. [22] for superconducting qubits.

Figure 8 plots the rate parameter (Fig. 8a) and the average run-time (Fig. 8b) as a function of group size for AEM Gmin as well as SEM Gmin. Results for no noise Gmin are included for comparison. For these realistic hardware parameters, we see that the rate parameter is well-protected by AEM, and the increase in run-time over no-noise conditions is still within reason, whereas the time for SEM is beyond a reasonable run-time. When observing the simulation in real-time, we recognize for $n = 6$ the probability of failure for a single oracle call is high, implying that a test of any larger groups would require an increase in the T_1 and T_2 times as argued in section 'Performance of AEM Gmin'.

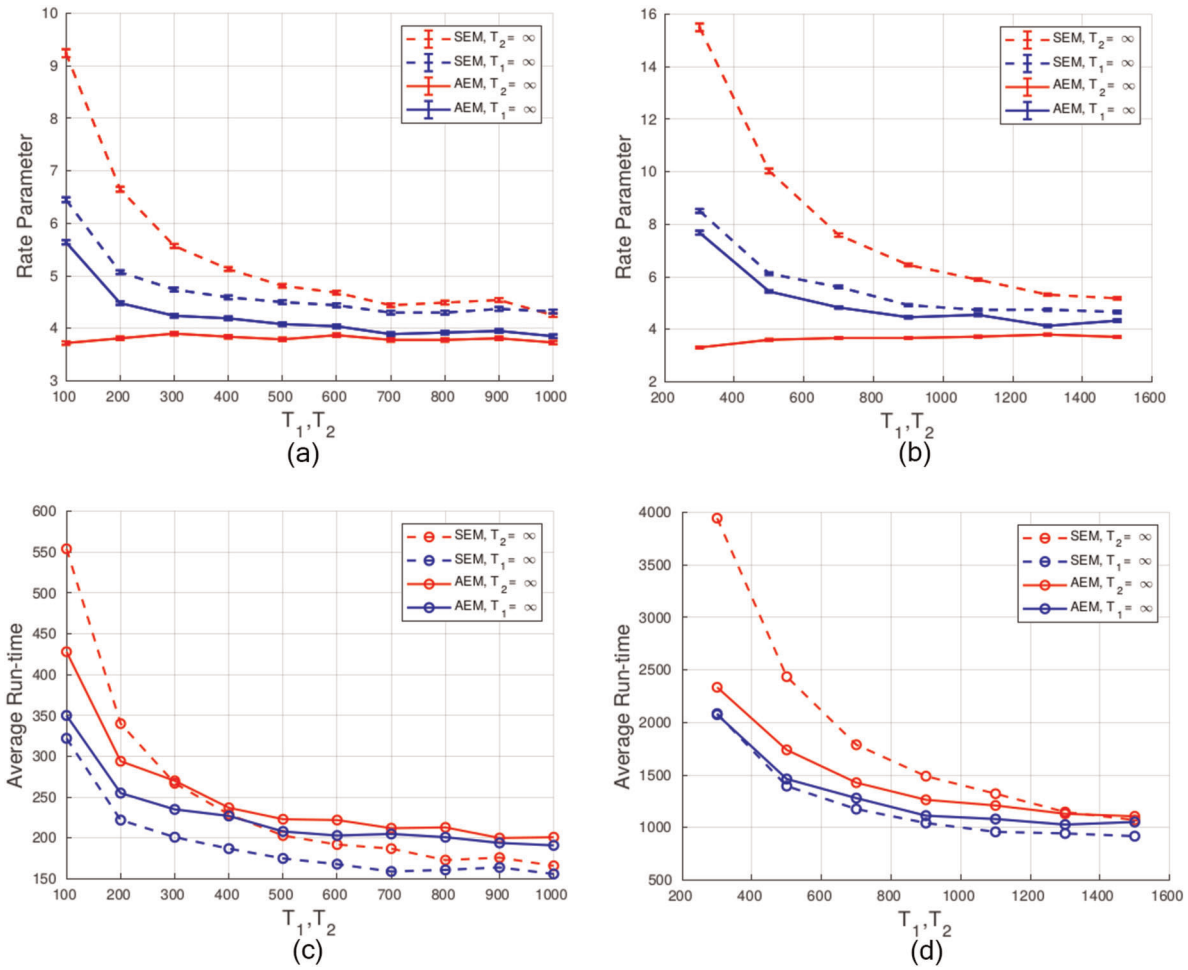

Fig. 7 Simulation plots for effective rate parameter from Eq. (13) and average run-time to contrast SEM (dashed) versus AEM (solid). For each plot, either T_1 (blue) or T_2(red) are fixed at $10^9 \sim \infty$ and the other is varied. In all cases, the number of trials is 4000. T_1, T_2 and average run-time are measured in units of the SQGT. Error bars for the effective rate parameter are calculated according to Eq. (14). **a** Effective rate parameter plot for $\log|G| = 4$. **b** Effective rate parameter plot for $\log|G| = 5$. **c** Average run-time plot for $\log|G| = 4$. **d** Average run-time plot for $\log|G| = 5$.

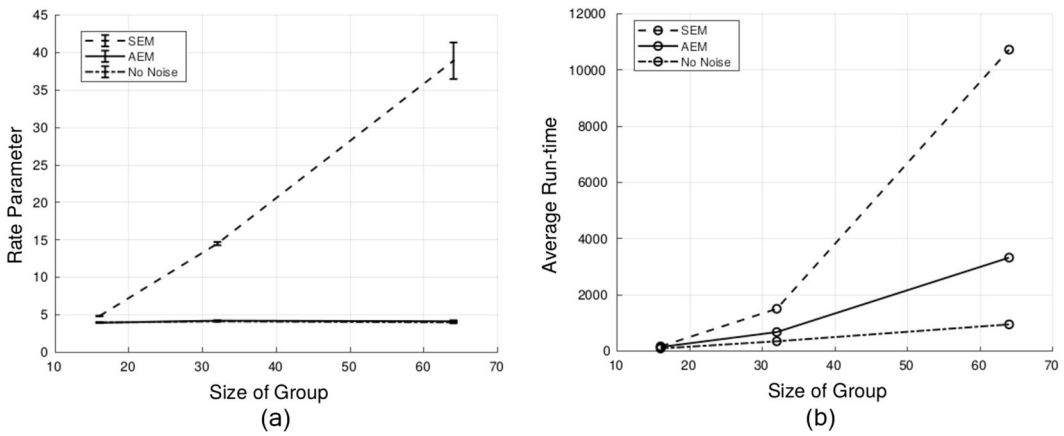

Fig. 8 Simulation plots for effective rate parameter from Eq. (13) and average run-time using the realistic parameters $T_1 = T_2 = 700$ SQGTs. The effective rate parameter for AEM and no-noise are almost indistinguishable on this scale. Average run-time is also measured in units of the SQGT. The number of trials used is 4000 for group sizes 16 and 32, and 500 for group size 64. Error bars for the effective rate parameter are calculated according to Eq. (14). **a** Rate parameter plot vs. group size. **b** Average run-time vs. group size.

DATA AVAILABILITY

The data that supports the findings of this study are available upon reasonable request to the corresponding author.

CODE AVAILABILITY

The open-source code for Intel-QS can be found at https://01.org/intel-quantum-simulator. Modifications to the Intel-QS as adapted for the contents of this work as well as the Matlab code used for the classical simulations presented in Figs 4 and 5 are available upon reasonable request to the corresponding author.

Received: 15 April 2019; Accepted: 26 November 2019;
Published online: 08 January 2020

REFERENCES

1. Preskill, J. Quantum computing in the NISQ era and beyond. *Quantum* **2**, 79 (2018).
2. Shor, P. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* **26**, 1484–1509 (1997).
3. Grover, L. K. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC'96, 212–219 (ACM, New York, NY, USA, 1996).
4. Mateus, P. & Omar, Y. Quantum pattern matching. Preprint at https://arxiv.org/pdf/quant-ph/0508237.pdf (2005).
5. Dürr, C. & Høyer, P. A quantum algorithm for finding the minimum. *CoRR*. Preprint at https://arxiv.org/abs/quant-ph/9607014 (1996).
6. Farhi, E., Goldstone, J. & Gutmann, S. A quantum approximate optimization algorithm. Preprint at https://arxiv.org/pdf/1411.4028.pdf (2014).
7. Tinkham, M. *Group Theory and Quantum Mechanics*. Dover Books on Chemistry and Earth Sciences. (Dover Publications, 2003).
8. Wietek, A. & Läuchli, A. M. Sublattice coding algorithm and distributed memory parallelization for large-scale exact diagonalizations of quantum many-body systems. *Phys. Rev. E* **98**, 033309 (2018).
9. Schulenburg, J. Spinpack using fpga, notes on fpga implementation of permutations. https://www-e.uni-magdeburg.de/jschulen/spin/.
10. Läuchli, A. M., Sudan, J. & Sørensen, E. S. Ground-state energy and spin gap of spin-$\frac{1}{2}$ kagomé-heisenberg antiferromagnetic clusters: large-scale exact diagonalization results. *Phys. Rev. B* **83**, 212401 (2011).
11. Weiße, A. Divide and conquer the hilbert space of translation-symmetric spin systems. *Phys. Rev. E* **87**, 043305 (2013).
12. Nielsen, M. A. & Chuang, I. L. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 10th edn, (Cambridge University Press: New York, NY, USA, 2011).
13. Lanczos, C. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Natl. Bur. Stand. B* **45**, 255–282 (1950).
14. Boyer, M., Brassard, G., Høyer, P. & Tapp, A. Tight bounds on quantum searching. *Fortschr. Phys.* **46**, 493–505 (1998).
15. Cuccaro, S. A., Draper, T. G., Kutin, S. A. & Moulton, D. P. A new quantum ripple-carry addition circuit. Preprint at https://arxiv.org/pdf/quant-ph/0410184.pdf (2004).
16. Smelyanskiy, M., Sawaya, N. P. D. & Aspuru-Guzik, A. qhipster: The quantum high performance software testing environment. Preprint at https://arxiv.org/pdf/1601.07195.pdf (2016).
17. Barenco, A. et al. Elementary gates for quantum computation. *Phys. Rev. A* **52**, 3457–3467 (1995).
18. Shenvi, N., Brown, K. R. & Whaley, K. B. Effects of a random noisy oracle on search algorithm complexity. *Phys. Rev. A* **68**, 052313 (2003).
19. Regev, O. & Schiff, L. In *Automata, Languages and Programming*. (eds Aceto, L., Damgård, I., Goldberg, L. A., Halldórsson, M. M., Ingólfsdóttir, A. & Walukiewicz, I.) 773–781 (Springer, Berlin, Heidelberg, 2008).
20. Geller, M. R. & Zhou, Z. Efficient error models for fault-tolerant architectures and the pauli twirling approximation. *Phys. Rev. A* **88**, 012314 (2013).
21. Terhal, B. M. Quantum error correction for quantum memories. *Rev. Mod. Phys.* **87**, 307–346 (2015).
22. O'Brien, T. E., Tarasinski, B. & DiCarlo, L. Density-matrix simulation of small surface codes under current and projected experimental noise. *NPJ Quantum Inf.* **3**, 39 (2017).

ACKNOWLEDGEMENTS

The authors would like to thank Jim Held, Justin Hogaboam, Anne Matsuura and Xiang Zou for useful discussion. A.T.S. would also like to thank Rahul M. Nandkishore.

AUTHOR CONTRIBUTIONS

A.T.S. designed the circuits, implementation and error mitigation strategies and contributed to their analysis as well as wrote and performed the simulations. S.J. first identified the problem and the quantum solution and supervised and contributed to the analysis of the findings. Both authors contributed equally to the preparation of the manuscript.

COMPETING INTERESTS

The authors declare no competing interests.

ADDITIONAL INFORMATION

Supplementary information is available for this paper at https://doi.org/10.1038/s41534-019-0232-1.

Correspondence and requests for materials should be addressed to A.T.S.

Reprints and permission information is available at http://www.nature.com/reprints

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.

© The Author(s) 2020