




Flexible constraint compilation in the parity architecture

Roeland ter Hoeven ^{1,2}, Anette Messinger ¹, and Wolfgang Lechner ^{1,2,*}

¹Parity Quantum Computing GmbH, A-6020 Innsbruck, Austria

²Institute for Theoretical Physics, University of Innsbruck, A-6020 Innsbruck, Austria



(Received 26 April 2023; accepted 12 September 2023; published 12 October 2023)

We present tools and methods to generalize parity compilation to digital quantum computing devices with arbitrary connectivity graphs and construct circuit implementations for the constraint Hamiltonian of higher-order constrained binary optimization problems. In particular, we show how even nonlocal constraints can be efficiently implemented without expensive SWAP gates. We show how the presented tools can be used to optimize the total circuit depth and CNOT count of the quantum approximate optimization algorithm in the parity architecture and highlight the advantages of the flexible compilation using various examples. We derive the relation between the developed gate sequences and the traditional approach that uses SWAP gates. The result can be applied to improve the implementation of many other nonlocal operators.

DOI: [10.1103/PhysRevA.108.042606](https://doi.org/10.1103/PhysRevA.108.042606)

I. INTRODUCTION

Quantum systems have improved a lot in terms of qubit numbers and coherence in the last years [1–3], but still one of the major challenges of building scalable quantum computers is qubit connectivity. Quantum noise [4–6] such as crosstalk errors can lead to issues like frequency crowding [7] and therefore limits the number of qubits that are connected. The parity architecture [8–11] offers a way to solve optimization problems, and in a recent work [12] also to implement arbitrary quantum algorithms, using only local interactions. Compiling hard optimization problems (e.g., with constraints and higher-order interactions) to the parity architecture on restricted quantum hardware is still a challenge.

In this paper, we outline a way to do parity compilation on arbitrary devices for arbitrary problems. Previous approaches required all parity constraints to be local, which made it impossible to complete the parity mapping for some sparsely connected devices (e.g., linear chains). Instead, we show how to implement nonlocal constraints by introducing a concept called bridging, which allows for a more efficient implementation than using SWAP gates. The more nonlocal the constraints are, the more resources (e.g., CNOT gates) they require. We, therefore, also propose methods to minimize the nonlocality of the constraints for a given optimization problem and hardware layout.

We show that, for every optimization problem mapped to the parity architecture, one can find an implementation of the constraint terms using nearest-neighbor CNOT (or other

entangling) gates and local rotations. In particular, we provide tools to make an optimal choice of constraints and qubit layout to minimize the CNOT count or circuit depth. The main steps towards this rely on the following three tasks.

(1) Finding a valid set of constraints for a given parity mapping, where the maximum number of qubits in each constraint is restricted.

(2) Implementing (potentially nonlocal) constraints using local operations.

(3) Evaluating and minimizing the cost of implementing a constraint or a set of constraints based on the position of the corresponding qubits and the connectivity of the physical device.

The presented methods can be used to find a parity mapping for sparse or irregular connectivity graphs, for example, on hardware platforms with missing or noisy qubits, or on platforms using nonrectangular lattice geometries [7,13,14].

The next section will introduce the parity architecture and the quantum approximate optimization algorithm (QAOA) [15,16]. After that the three main tasks defined above are addressed. Finally, we show how to solve constrained optimization problems within this framework.

II. PARITY ARCHITECTURE AND QAOA

Within this work, we consider Hamiltonians defining an optimization problem of the form

$$\hat{H} = \sum_{i=1}^N J_i \hat{\sigma}_z^{(i)} + \sum_{i=1}^N \sum_{j>i} J_{ij} \hat{\sigma}_z^{(i)} \hat{\sigma}_z^{(j)} + \sum_{i=1}^N \sum_{j>i} \sum_{k>j} J_{ijk} \hat{\sigma}_z^{(i)} \hat{\sigma}_z^{(j)} \hat{\sigma}_z^{(k)} + \dots, \quad (1)$$

where $\hat{\sigma}_z^{(i)}$ are Pauli operators representing the spin variables $s_i \in \{+1, -1\}$ of the optimization problem and the J coefficients define the strength of the corresponding interactions.

*wolfgang@parityqc.com; wolfgang.lechner@uibk.ac.at

Published by the American Physical Society under the terms of the [Creative Commons Attribution 4.0 International license](https://creativecommons.org/licenses/by/4.0/). Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

Some of the J coefficients may be 0 for specific optimization problems, resulting in a more sparse Hamiltonian. For constrained optimization problems, there are additional side conditions

$$\left(\sum_{i=1}^N c_i \hat{\sigma}_z^{(i)} + \sum_{i=1}^N \sum_{j>i} c_{ij} \hat{\sigma}_z^{(i)} \hat{\sigma}_z^{(j)} + \sum_{i=1}^N \sum_{j>i} \sum_{k>j} c_{ijk} \hat{\sigma}_z^{(i)} \hat{\sigma}_z^{(j)} \hat{\sigma}_z^{(k)} + \dots \right) |\psi\rangle = C_V |\psi\rangle \quad (2)$$

for a state $|\psi\rangle$ to be a valid solution. Note that the coefficients c must be consistent with the total value C_V to allow for solutions. These conditions can either be added to the problem Hamiltonian by penalizing nonfulfilling configurations, or they can be fulfilled implicitly by modifying the dynamics of the optimization process such that they are never violated. We focus on the second option in this work.

Here we build on the results from Ref. [9], where the parity mapping was introduced for higher-order-constrained binary optimization (HCBO) problems. The parity mapping transforms the Hamiltonian by introducing a parity variable for each term and replacing the product of Pauli operators by a single operator on the new variable (e.g., $J_{ij} \hat{\sigma}_z^{(i)} \hat{\sigma}_z^{(j)} \rightarrow \tilde{J}_m \hat{\sigma}_z^{(m)}$ with $\tilde{J}_m = J_{ij}$) resulting in the final problem Hamiltonian

$$\hat{H}_P = \sum_{k=1}^K \tilde{J}_k \hat{\sigma}_z^{(k)} - \sum_{l=1}^L C_l \prod_{m_l} \hat{\sigma}_z^{(m_l)}. \quad (3)$$

Here the first term encodes the problem, originally containing K interactions, as K single-body terms on an increased number of spin variables. The second term compensates for the introduced redundancy by adding L constraints on valid physical states $|\psi\rangle$,

$$\prod_{m_l} \hat{\sigma}_z^{(m_l)} |\psi\rangle = |\psi\rangle, \quad (4)$$

where C_l has to be large enough to energetically separate the invalid states. In the rest of this paper, we will often label the physical parity qubits with the corresponding original spin variables (e.g., $J_{ij} \hat{\sigma}_z^{(i)} \hat{\sigma}_z^{(j)} \rightarrow \tilde{J}_{ij} \hat{\sigma}_z^{(i,j)}$) for clarity. The constraints can be understood as consistency conditions on the physical qubits and can be found by considering generalized cycles in the problem Hamiltonian. A generalized cycle is defined by a set of physical qubits, where each index appears an even number of times. For example, the product of three physical qubits $\hat{\sigma}_z^{(i,j)}$, $\hat{\sigma}_z^{(j,k,l)}$, and $\hat{\sigma}_z^{(k,l,i)}$ needs to satisfy the following consistency relation:

$$\hat{\sigma}_z^{(i,j)} \hat{\sigma}_z^{(j,k,l)} \hat{\sigma}_z^{(k,l,i)} |\psi\rangle \quad (5)$$

$$= \hat{\sigma}_z^{(i)} \hat{\sigma}_z^{(j)} \hat{\sigma}_z^{(k)} \hat{\sigma}_z^{(l)} \hat{\sigma}_z^{(l)} \hat{\sigma}_z^{(k)} \hat{\sigma}_z^{(i)} |\psi\rangle = |\psi\rangle, \quad (6)$$

because all indices i , j , k , and l appear an even amount of times in the product and therefore all $\hat{\sigma}_z$ operators on the original spin variables cancel out. In the next section we

will go into more detail about finding an appropriate set of constraints.

The QAOA can then be run on the parity architecture by alternating between problem unitaries of the form

$$\hat{U}_P = \exp(i\gamma \hat{H}_P), \quad (7)$$

and driver unitaries

$$\hat{U}_X = \exp\left(i\beta \sum_{k=1}^K \hat{\sigma}_x^{(k)}\right) \quad (8)$$

[17]. Here H_P is the Hamiltonian as introduced in Eq. (3). Since the energy penalties on the constraints guarantee that the ground state of the mapped problem Hamiltonian translates to the same as the ground state of the original Hamiltonian, the algorithm can find it in exactly the same way using the mapped version. Apart from the constraints introduced by the parity mapping, all terms in \hat{U}_P can be implemented through single-qubit operations. In this work, we therefore focus on the efficient implementation of the multi-qubit terms appearing as constraints. We furthermore show how parity compilation can allow easy implementations of adjusted driver unitaries to ensure the satisfaction of additional constraints of the original problem (without the need to penalize them in the problem Hamiltonian [16]).

III. FINDING A BASIS OF SHORT CONSTRAINTS

For any given problem Hamiltonian, a basis of the required constraints for a valid parity mapping can be found with linear algebra methods as introduced in Ref. [9]. In the following, we briefly review the general methodology, before focusing on a basis representation through short constraints.

We define the length of a constraint as the number of qubits involved in the constraint. The longer a constraint, the more resources it will require in the implementation of the QAOA (we will look at concrete numbers in the next section). The goal is to find a basis of short constraints that allows for a valid parity mapping to the physical problem. We use the terminology of a basis of constraints equivalent to the set of constraints which generate the stabilizer of the code space (the subspace of the physical Hilbert space which corresponds to valid logical states, i.e., has no inconsistencies when mapping parity qubits to original spin variables).

Consider as an example the problem Hamiltonian

$$\begin{aligned} \hat{H} = & J_{12} \hat{\sigma}_z^{(1)} \hat{\sigma}_z^{(2)} + J_{15} \hat{\sigma}_z^{(1)} \hat{\sigma}_z^{(5)} \\ & + J_{24} \hat{\sigma}_z^{(2)} \hat{\sigma}_z^{(4)} + J_{45} \hat{\sigma}_z^{(4)} \hat{\sigma}_z^{(5)} \\ & + J_{123} \hat{\sigma}_z^{(1)} \hat{\sigma}_z^{(2)} \hat{\sigma}_z^{(3)} + J_{345} \hat{\sigma}_z^{(3)} \hat{\sigma}_z^{(4)} \hat{\sigma}_z^{(5)}. \end{aligned} \quad (9)$$

The generator matrix, which maps the original problem variables into the code subspace [9], can be constructed by writing the terms as columns in a matrix. Each column will have a 1 in row i if $\hat{\sigma}_z^{(i)}$ is in the corresponding term and 0 otherwise. In the example, $\hat{\sigma}_z^{(1)} \hat{\sigma}_z^{(2)}$ becomes the first column, with nonzero entries in the first and second row. We thus arrive at the

generator matrix

$$\mathbf{G} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}. \quad (10)$$

The parity check matrix \mathbf{P} can then be found by solving the equation

$$\mathbf{G}\mathbf{P}^T \equiv 0 \pmod{2} \quad (11)$$

for a matrix of maximal rank. This corresponds exactly to the step of finding consistency conditions of the parity qubits. For the generator matrix in Eq. (10), a possible solution for the parity check matrix is

$$\mathbf{P} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}. \quad (12)$$

The rows in this matrix represent a basis of the constraint space to be implemented; we call this the target constraint space. Similar to the generator matrix, a 1 in column i of the parity check matrix indicates that there is a term $\hat{\sigma}_z^{(i)}$ in the corresponding constraint. Any basis of the constraint space describes a valid choice for a set of constraints to be implemented in the mapped problem Hamiltonian. However, we can find many different choices for a basis by performing elementary row operations. Note that, as the entries of \mathbf{G} and \mathbf{P} are in \mathbb{Z}_2 , all operations are performed modulo 2. The length of a constraint is its number of nonzero entries. In general, a constraint can be as long as the number of terms in the original Hamiltonian and it is a hard problem to find a basis that only consists of constraints up to a certain length. However, if we are only interested in finding a basis of short constraints, for example, of length 3 and 4, this can be done efficiently. For a fixed number L , all constraints of length $l \leq L$ in a system with K Hamiltonian terms can be found in complexity $\mathcal{O}(K^L)$.

The simplest way to achieve this is by taking all combinations of L physical spin variables and checking if their product forms a valid constraint. We can confirm that a constraint is valid if the corresponding row vector is in the target constraint space. Alternatively, we can check that each logical index occurs an even amount of times in the product. If that is the case, the product of physical spin variables in the constraint is 1 by definition. For example, the qubit product

$$\hat{\sigma}_z^{(1,5)} \hat{\sigma}_z^{(2,4)} \hat{\sigma}_z^{(1,2,3)} \hat{\sigma}_z^{(3,4,5)} \quad (13)$$

forms a valid constraint as

$$\hat{\sigma}_z^{(1)} \hat{\sigma}_z^{(5)} \hat{\sigma}_z^{(2)} \hat{\sigma}_z^{(4)} \hat{\sigma}_z^{(1)} \hat{\sigma}_z^{(2)} \hat{\sigma}_z^{(3)} \hat{\sigma}_z^{(3)} \hat{\sigma}_z^{(4)} \hat{\sigma}_z^{(5)} = \hat{I}. \quad (14)$$

This can equivalently be verified by checking that it is in the target constraint space directly, using the found basis and Gaussian elimination.

We can then grow a basis of short constraints, by adding to it each time we find a new and linearly independent combination of qubits that form a valid constraint. The process is finished when the constraints in the basis span the target constraint space. However, it is possible that the short constraints are not enough to span the target constraint space, in which case we can use ancilla qubits to break down long constraints

into short ones. Consider the following parity check matrix, which emerges in the parity mapping of a logical problem graph that is a single cycle of length 5:

$$\mathbf{P} = (1 \ 1 \ 1 \ 1 \ 1). \quad (15)$$

It consists of a single constraint of length 5. To obtain a basis of constraints of lengths 3 and 4, we need to add an ancilla and a row to this parity check matrix, e.g.,

$$\mathbf{P} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}. \quad (16)$$

Here, the last column corresponds to the added ancilla. The original constraint can be found by adding the first and second rows and eliminating the ancilla. With this process, any long constraint can be reduced to a number of small constraints using some ancilla qubits.

In summary, the process for a general problem will have the following steps. First, enumerate all combinations of three and four qubits and find all valid constraints that can be made. If these constraints span the target constraint space, then pick the smallest subset of them which still spans the target constraint space. Otherwise, if there are still missing constraints, go over the original basis of (potentially long) constraints of the target constraint space and identify the constraints which are not in the initial space of short constraints. For these constraints, it is impossible to find a decomposition into short constraints, so we need ancillas to break them down. Add the resulting constraints including ancillas to the basis of the short constraints and keep iterating over the target constraint space basis until all constraints are included.

Note that, in general, it is not necessary to stick to a basis of constraints of lengths 3 and 4 because constraints of an arbitrary length can be implemented. However, allowing longer constraints increases the search space and can also cost more CNOT gates. In some cases it is possible to avoid using ancillas by allowing longer constraints, e.g., for Eq. (15) no ancillas are needed when constraints of length 5 are directly implemented. There can also be a trade-off between the necessary number of ancillas and circuit depth, because long constraints take more time to execute and are harder to parallelize.

IV. GATE SEQUENCES FOR CONSTRAINTS

Consider a quantum device that can do single-qubit operations on all qubits and an entangling two-qubit gate between some of the qubits such that a universal gate set over all qubits is achieved. The connectivity graph G of the device, where all qubits are nodes and the qubits that can directly interact with each other have an edge, is then a connected graph.

In the following, we show how to efficiently implement a constraint C with the operator

$$\hat{U}_C = \exp\left(i\alpha_C \prod_{m \in C} \hat{\sigma}_z^m\right) \quad (17)$$

in the QAOA for any arrangement of the qubits in C and any angle α_C using a decomposition into CNOT gates and local rotations. Any n -qubit rotation $\exp(i\alpha \hat{\sigma}_z^{\otimes n})$ can be decomposed

into an $n - 1$ -qubit rotation $\exp(i\alpha\hat{\sigma}_z^{\otimes n-1})$ and two CNOT gates with control on the n th qubit and target on any of the $n - 1$ other qubits, applied before and after the rotation (the target must be the same qubit for both CNOT gates) [18]. This can be iteratively applied to decompose any many-body rotation, such as the constraint operator in Eq. (17), into a sequence of CNOT gates followed by a single-body rotation and the reverse sequence of CNOT gates.

The effect of such CNOT sequences can be understood by looking at the effect of a single CNOT gate, collecting the parity of its control and target qubit at the target

$$|a\rangle|b\rangle \xrightarrow{\text{CNOT}} |a\rangle|a \oplus b\rangle, \quad a, b \in 0, 1. \quad (18)$$

Both control and target qubits can already hold the parity of other states, for example, if they were targeted by other CNOT gates before. A Z rotation on a physical qubit which holds the parity of multiple (logical) qubit states then effectively performs a collective rotation on all these logical qubits by adding a phase only to those states whose corresponding multi-qubit parity is odd. After having performed such a rotation, the original CNOT sequence can be applied in reverse order to return to the original representation of qubits.

For a given constraint C , there are many different options for such sequences, as any CNOT sequence which collects the parity of the qubits in C towards a single qubit can be used. The effect on any other qubits is irrelevant as it is undone afterwards by the reversed CNOT sequence. However, as the connectivity of most quantum devices is limited, we are interested in a sequence using only local interactions. We will show that, for any tree in the connectivity graph G which spans the qubits in C , we can find such a sequence using CNOT gates along the edges of that tree. We call such a tree the Steiner tree of C in G . The resulting decomposition has depth $l_{\max} + \mathcal{O}(1)$, where l_{\max} is the largest distance (i.e., the maximal number of edges) between two constraint qubits in the tree.

A. Local constraints

If there is a Steiner tree of a constraint C in the connectivity graph G which only contains the qubits in C , we call this constraint local. For a local constraint, we can find an implementation as follows. First, we choose a root of the tree on which we want to perform the single-body rotation. Then we can obtain the CNOT sequence to be applied after the rotation by traversing the tree from the chosen root towards the leaves and adding a CNOT gate at every edge, controlling the child (outer) node and targeting the parent (inner) node. The CNOT sequence to be applied before the rotation is then the inverse of this. The resulting decomposition is shown for an example tree in Fig. 1. The root of the tree can be any qubit in C , but is usually chosen such that its longest distance to any leaf is minimized, which typically corresponds to the implementation with the smallest circuit depth.¹ We furthermore have the freedom to choose the order of CNOT gates which are on edges entering the same node (i.e., target the

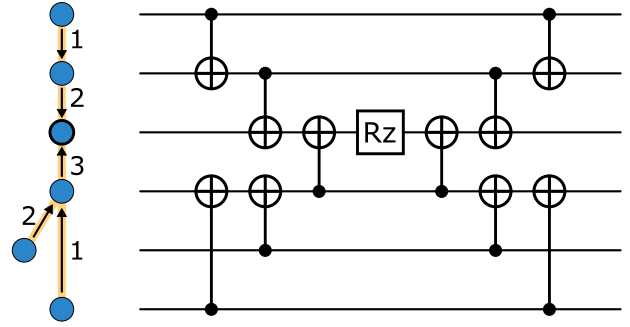


FIG. 1. Implementation of a local constraint on six qubits using the Steiner tree highlighted in yellow. Numbers and arrows indicate the time step and direction (control to target) of a CNOT gate on the given edge before the rotation gate at the root. After the rotation, the CNOT gates are implemented with the same direction, but in opposite order. Note that all CNOT gates are local in the two-dimensional nearest-neighbor lattice, even though they are drawn here in one dimension.

same qubit). Both the choice of the root and the order of CNOT gates with the same target can be used to further optimize the resulting circuit depth, either of the individual constraint, or of a larger circuit implementing multiple constraint operators in parallel.

B. Nonlocal constraints

If the Steiner tree of a constraint C contains qubits which are not in C , we can use a similar strategy to implement the constraint. Instead of using SWAP gates to bring the constraint qubits next to each other, we derive a sequence of CNOT gates to effectively bridge the nonconstraint qubits. We engineer the CNOT sequence such that every constraint qubit appears once in the final parity of the rotation qubit, but every bridged qubit along the way appears twice and thus its effect cancels out. For this, we start with the CNOT sequence which implements a constraint on the full tree, including the nonconstraint qubits along the way. We then add CNOT gates in the beginning and end of the constraint circuit, which add the information of each nonconstraint qubit as a parity onto exactly one other qubit, as, for example, is shown in Fig. 2.

Together with the CNOT sequence over the full Steiner tree T , the information of those qubits is collected twice, once from the original qubit and once from the qubit which was targeted with the additional CNOT gate. The final operation which is performed is then

$$\hat{U}_C = \exp\left(i\alpha_C \prod_{m \in T} \hat{\sigma}_z^m \prod_{m \in T \setminus C} \hat{\sigma}_z^m\right) \quad (19)$$

$$= \exp\left(i\alpha_C \prod_{m \in C} \hat{\sigma}_z^m\right), \quad (20)$$

where $T \setminus C$ is the set of qubits which are in the Steiner tree but not in the constraint. Whenever $T \setminus C$ contains more than one qubit, the order of the additional CNOT gates has to be chosen such that they do not duplicate the qubit information more than once. Consider the case of bridging two neighboring qubits i and j along a tree. A gate

¹Note that on trees with many branches, the circuit depth does not only depend on the distance between the root and the leaves but also on the order of the nodes in the tree.

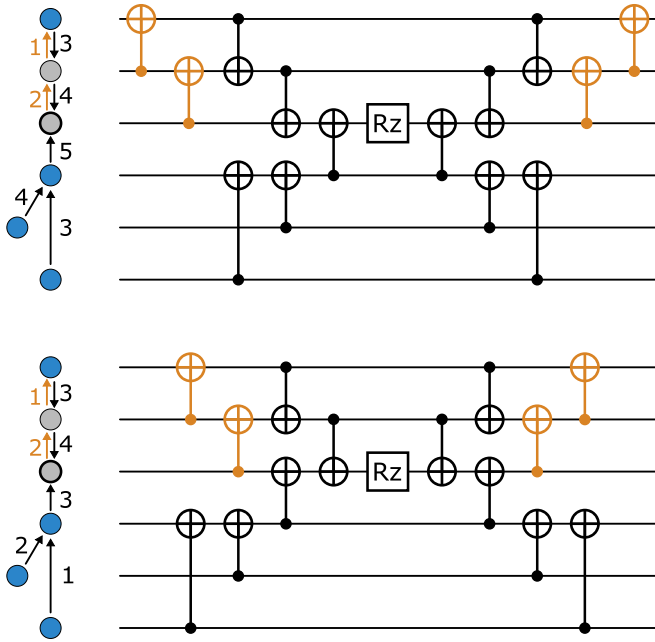


FIG. 2. Implementation of a nonlocal constraint on four qubits (dark blue) with two other qubits (light gray) along the Steiner tree in the same representation as Fig. 1. Orange (light) CNOT gates indicate additional gates that are required to cancel out the effect of the unwanted qubits along the tree. The top circuit shows the result when the optimal-depth CNOT sequence for the full tree is simply extended by the additional gates. In the bottom circuit, the CNOT sequence of the full tree has been rearranged to further reduce the depth of the full circuit. Note that all CNOT gates are local in the two-dimensional nearest-neighbor lattice, even though they are drawn here in one dimension.

CNOT_{ij} followed by CNOT_{jk} , for example, would transform a basis state $|a_i\rangle|a_j\rangle|a_k\rangle$ to the undesired state $|a_i\rangle|a_i \oplus a_j\rangle|a_i \oplus a_j \oplus a_k\rangle$, where qubit a_i now appears three times. The reverse sequence, however, results in the desired state $|a_i\rangle|a_i \oplus a_j\rangle|a_j \oplus a_k\rangle$, duplicating exactly a_i and a_j . In addition to the order of CNOT gates, we also need to choose the target qubit, i.e., on which qubit we duplicate the information. Any qubit neighboring the unwanted qubit in the Steiner tree T is a valid choice for this, no matter if it is a constraint qubit or not. As we want to construct a circuit with minimal depth, we choose to duplicate the information of every unwanted qubit to its child node in the tree, i.e., with a CNOT in the opposite direction of the following full Steiner tree implementation. This way, the additional CNOT gates can be applied almost in parallel to the Steiner tree implementation and result in a maximal depth increase of four CNOT steps compared to the full Steiner tree implementation (see Fig. 3).

The total number of CNOT gates required for a constraint C with Steiner tree T can then be calculated from the gates required for the full Steiner tree implementation and the additional CNOT gates for bridging unwanted qubits as

$$n_{\text{CNOT}} = 2(|T| - 1) + 2(|T| - |C|) = 4|T| - 2|C| - 2, \quad (21)$$

where $|\cdot|$ is the number of qubits in the tree or the constraint.

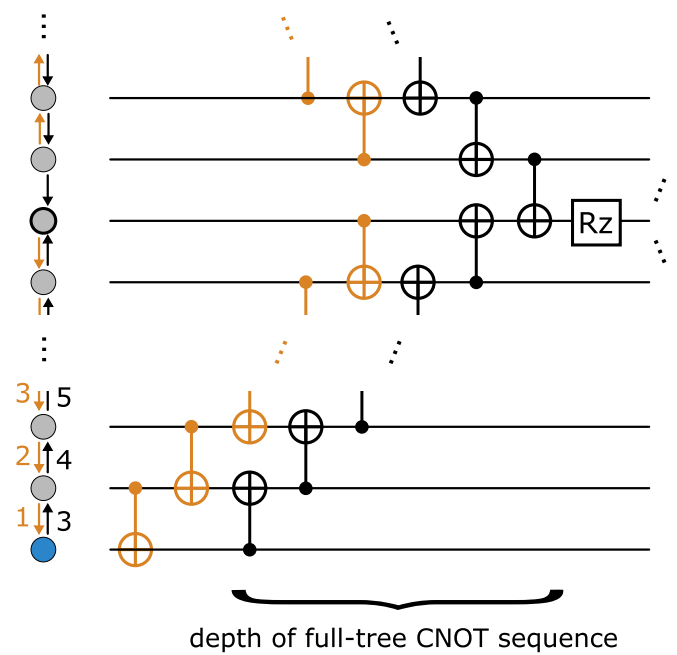


FIG. 3. Illustration of the worst-case circuit depth for nonlocal constraints. On any branch of the Steiner tree, the CNOT gates required for bridging non-constraint qubits will protrude from the CNOT sequence implementing the full Steiner tree by at most two time steps. As the same sequence is repeated in reverse after the rotation in the center (not shown in this figure), the maximal depth increase compared to a constraint on the full Steiner tree is four.

C. Circuit equivalence to SWAP sequence

The presented gate sequence is equivalent in its effect to the conventional method of rearranging the constraint qubits via SWAP gates until the constraint is local. Figure 4 shows how an efficient bridge sequence can be derived from an approach using SWAP gates on a simple example. The derivation also works for more complicated constraint arrangements, and in general can also be applied to other nonlocal interactions whenever the interaction commutes with $\hat{\sigma}_x$ or $\hat{\sigma}_z$ on each of its qubits. Even if the interaction commutes with neither of them, the procedure can still work if the SWAP gate is decomposed into other gates (e.g., controlled- Y gates if the nonlocal interaction commutes with $\hat{\sigma}_y$). The main step in the simplification of the SWAP sequence lies in the cancellation of gates which appear again in the reverse SWAP sequence and commute with everything in-between. If the implemented interaction commutes with $\hat{\sigma}_z$ (and thus with the controls of CNOT gates), the SWAP gates must be decomposed such that they begin and end with a CNOT gate controlling the qubits on the inside (where the desired interaction is implemented). Similarly, if the interaction commutes with another operator, the SWAP gates must be decomposed accordingly. CNOT gates resulting from a SWAP decomposition further away can still be commuted towards the inside by taking into account the other SWAP gates on the way (see first step in Fig. 4). With this, the total cost can be reduced from three to two gates per SWAP gate. Finally, rearranging the gates, taking into account the different possibilities to implement constraint operators, can further improve the circuit depth. For longer constraints

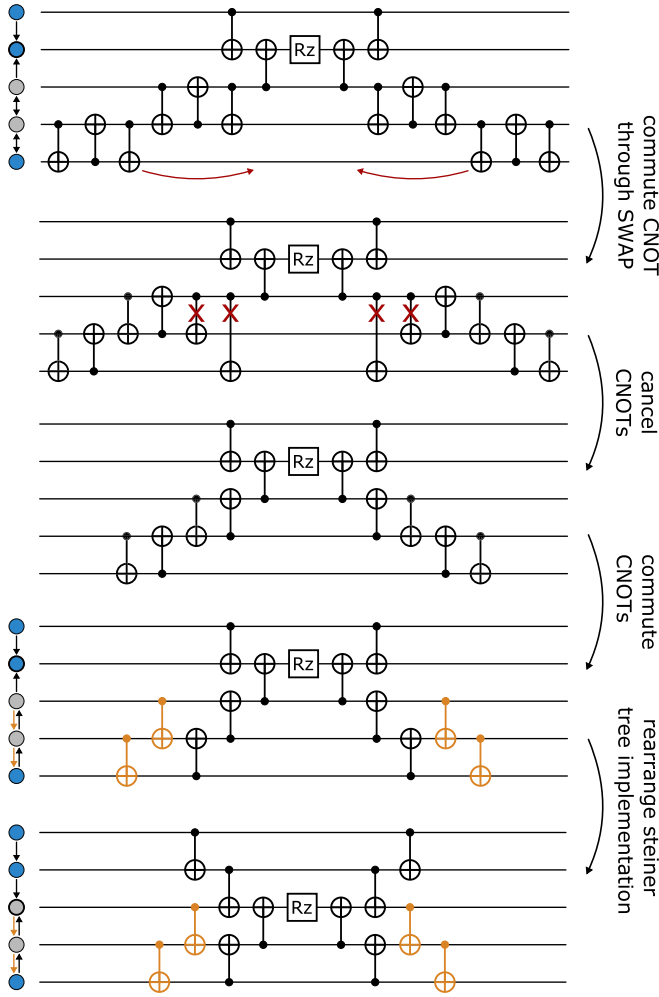


FIG. 4. Derivation of a bridging sequence of a non-local constraint operator $\exp(i\alpha\hat{\sigma}_z\hat{\sigma}_z\hat{\sigma}_z)$ on the qubits drawn in blue (darker) from an equivalent implementation via SWAP gates. The same derivation also works for other operators as long as there exists a SWAP decomposition whose last gate commutes with the target operator.

along more complex trees, the same idea still holds, although it becomes less straightforward.

D. Scaling comparison

To compare our solution to an implementation using SWAP gates, we consider a simple constraint of two qubits of distance l along the connectivity graph. Such a constraint requires

$$n_{\text{CNOT}}^{(\text{bridge})} = 2 + 4(l - 1) \quad (22)$$

CNOT gates when using the bridging method, whereas it would take

$$n_{\text{CNOT}}^{(\text{SWAP})} = 2 + 6(l - 1), \quad (23)$$

when using SWAP gates (implemented by three CNOT gates per SWAP gate). The corresponding circuit depths of this constraint implementation are

$$d^{(\text{bridge})} \leq 2 \left\lceil \frac{l+1}{2} \right\rceil + 4, \quad (24)$$

where the linear term is the depth of implementing a constraint operator on the full Steiner tree, and the constant is due to the additional bridging gates, and

$$d^{(\text{SWAP})} = 6 \left\lceil \frac{l+1}{2} \right\rceil - 4 \quad (25)$$

when assuming a SWAP sequence without further optimization. Note that the constant overhead of four additional steps in the bridge implementation does not apply for small l , where either no bridging is necessary ($l = 1$), or only two additional steps are necessary ($1 < l < 5$). While the existing transpilers [19,20] are likely able to cancel some of the CNOT gates of the occurring SWAP sequences, we stress that the main advantage of the construction via bridging lies in the rearrangement of the gates into a minimal-depth circuit which is not always straightforward using local circuit optimization strategies. The construction via parity, however, provides simple rules and allows flexibility for optimization. For longer constraints, the scaling of the circuit depth mainly depends on the largest distance of two constraint qubits in the Steiner tree, while the gate count is determined by the total length of the tree.

V. MINIMAL STEINER TREES FOR NONLOCAL CONSTRAINTS

Given that the cost of implementing constraint operators (both gate count and depth) scale linearly with the size of the corresponding Steiner tree, we now want to find the smallest such Steiner tree. The minimal tree that spans a subset of vertices (terminals) of the connectivity graph is well known from graph theory and called the minimal Steiner tree. We define the size of a tree as its number of edges.

There are many (approximate) algorithms to find minimal Steiner trees [21] in arbitrary graphs and also more specific algorithms that use the geometry of the graph.

For rectangular devices with nearest-neighbor connections, we get a rectilinear Steiner tree problem, first studied by Hanan [22]. He introduced the Hanan grid, which is obtained by drawing horizontal and vertical lines through all terminals. There always exists a minimal Steiner tree on the Hanan grid [23]. The rectilinear Steiner tree problem is also well studied in the presence of obstacles [24], which is relevant because realistic devices may have obstacles like defect qubits and missing qubit connections.

For short constraints up to a certain size (number of constraint qubits, independent of their position or distance in the connectivity graph), there is a fast way to find rectilinear Steiner trees. Here, we will show how to efficiently find minimal Steiner trees for constraints of lengths 3 and 4. The minimal Steiner tree can be found by sorting the terminals in the x direction and in the y direction and only keeping the component that was sorted for, such that $x_1 \leq x_2 \leq x_3 (\leq x_4)$ and $y_1 \leq y_2 \leq y_3 (\leq y_4)$ for three-qubit (four-qubit) constraints. For three-qubit constraints, the smallest tree size S_3 is

$$S_3 = (x_3 - x_1) + (y_3 - y_1), \quad (26)$$

which is the largest difference between vertices in the horizontal and vertical direction. For four-qubit constraints (see Fig. 5), a minimal Steiner tree can be found by first

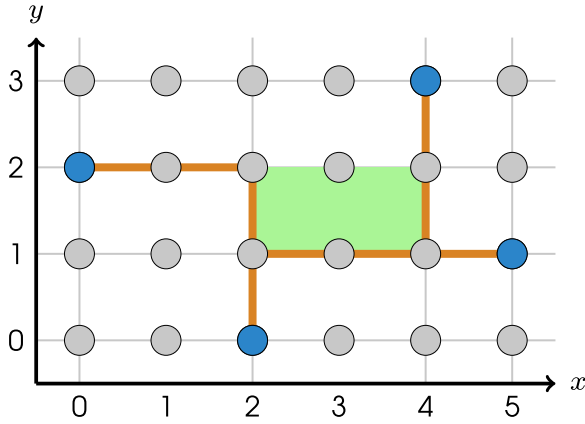


FIG. 5. Example of a minimal Steiner tree for the four blue (darker) nodes (terminals) on a rectangular lattice. There are many minimum Steiner trees of size 9, one of which is drawn with orange thick lines. The gray narrow lines represent the Hanan grid, constructed by drawing horizontal and vertical lines through all terminals. There always exists a minimal Steiner tree on the Hanan grid for any choice of terminals. The central rectangle that can be used to find optimal Steiner trees for four terminals is highlighted in green.

connecting all terminals to the closest corner of the central rectangle defined by its bottom left corner (x_2, y_2) and top right corner (x_3, y_3) , and then connecting the corners of the central rectangle. If the terminals connect to all four corners, this requires one connection on the longer side and two connections on the shorter side, otherwise a single connection in each direction suffices. The size of the minimal Steiner tree S_4 is then

$$S_4 \leq (x_4 - x_1) + (y_4 - y_1) + \min(y_3 - y_2, x_3 - x_2), \quad (27)$$

where the additional term is the length of the shorter side of the central rectangle, which can contribute twice to the total length.

In Fig. 5 we have $(x_1, x_2, x_3, x_4) = (0, 2, 4, 5)$ and $(y_1, y_2, y_3, y_4) = (0, 1, 2, 3)$, which results in the tree size $S_4 = 5 + 3 + \min(1, 2) = 9$.

VI. APPLICATION: 1D CONNECTIVITY

In this section, we look at a small example and compare to a direct implementation using the standard gate model. The Hamiltonian we will look at has the form

$$\begin{aligned} \hat{H} = & J_{12} \hat{\sigma}_z^{(1)} \hat{\sigma}_z^{(2)} + J_{13} \hat{\sigma}_z^{(1)} \hat{\sigma}_z^{(3)} + J_{14} \hat{\sigma}_z^{(1)} \hat{\sigma}_z^{(4)} \\ & + J_{23} \hat{\sigma}_z^{(2)} \hat{\sigma}_z^{(3)} + J_{123} \hat{\sigma}_z^{(1)} \hat{\sigma}_z^{(2)} \hat{\sigma}_z^{(3)} + J_{124} \hat{\sigma}_z^{(1)} \hat{\sigma}_z^{(2)} \hat{\sigma}_z^{(4)} \\ & + J_{134} \hat{\sigma}_z^{(1)} \hat{\sigma}_z^{(3)} \hat{\sigma}_z^{(4)} + J_{234} \hat{\sigma}_z^{(2)} \hat{\sigma}_z^{(3)} \hat{\sigma}_z^{(4)}, \end{aligned} \quad (28)$$

and the goal is to compile this problem on a one-dimensional device with nearest-neighbor interactions only. Under these restrictions, it is impossible to compile with only local constraints, so we need bridging to successfully compile it. We will then compare the gate count and circuit depth to a standard implementation of the gate model.

In general, any choice of the constraint basis and layout of qubits can be implemented using bridging. However, it is possible to optimize the choice of constraint basis and

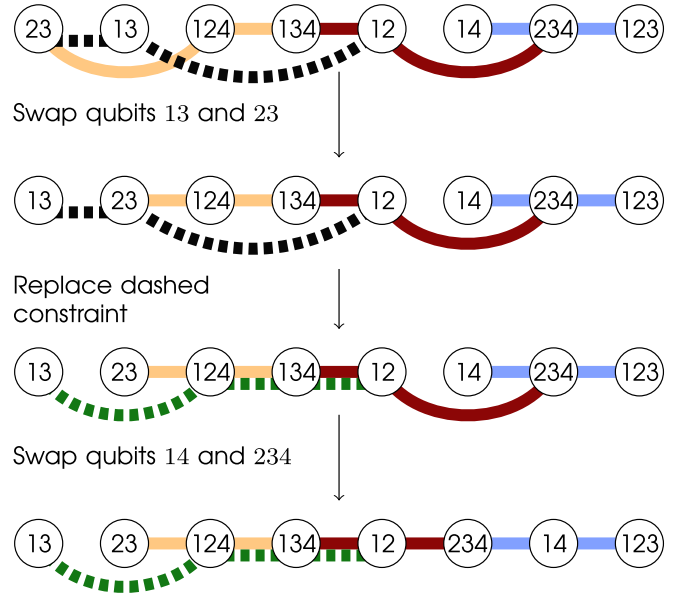


FIG. 6. Process of parity compilation with bridging, optimizing gate count, and depth. Qubits are marked by their logical indices (e.g., 23 means that a $\hat{\sigma}_z$ on this qubit corresponds to the logical operation $\hat{\sigma}_z^{(2)} \hat{\sigma}_z^{(3)}$) and each colored line connects the qubits of a parity constraint. The two possibilities for optimization are highlighted: a qubit swap and a change of the constraint basis. In each step of the process, the constraints become more local and thus cheaper to implement. In the first step the yellow (light line on the left) constraint is made local by swapping qubits 13 and 23. Then the black dashed constraint is replaced by the product of itself with the yellow constraint, resulting in the green dashed constraint. The green dashed constraint is slightly cheaper because it only has to bridge a single qubit. Finally, the red (darker solid line) constraint is made local by swapping qubits 14 and 234. All curved nonlocal interactions can be implemented with only nearest-neighbor CNOT gates using the introduced bridge sequence.

qubit layout to obtain constraints that are as local as possible. Both the constraint basis and qubit layout can be incrementally changed to improve the quality, as for example shown in Fig. 6. Optimization cannot only be done for individual constraints, but also to parallelize multiple constraints, which requires a minimization of the constraint overlap during optimization. Often, these two considerations agree, which means that making individual constraints as local as possible will also result in constraints that have small overlap and are therefore easier to parallelize.

The required amount of gates to implement the three local three-qubit constraints and the nonlocal four-qubit constraint in the last line of Fig. 6 can now be calculated. A local three-qubit constraint requires four CNOT gates, so the three constraints require $3 \times 4 = 12$ CNOT gates. The four-qubit constraint needs to bridge one qubit and requires ten CNOT gates. This means that in total we need 22 CNOT gates to implement the entire circuit.

For a standard implementation for the gate model on a one-dimensional device, a possible good layout is

$$4 - -1 - -2 - -3. \quad (29)$$

No matter how the qubits 1, 2, 3, and 4 are arranged, there will always be two local three-qubit terms and two nonqubit

three-body terms. The local three-qubit terms require four CNOT gates and the nonlocal terms eight CNOT gates. So in total the three-qubit terms require $2 \times 4 + 2 \times 8 = 24$ CNOT gates. For the two-qubit interactions, only three of them can be local and the last one must be nonlocal. In Eq. (29) we have three local two-qubit terms and one term which has to bridge a qubit. For the local terms we require two CNOT gates and for the last term six CNOT gates, which means that in total we require $3 \times 2 + 6 = 12$ CNOT gates for the two-qubit terms.² Overall, the standard gate model implementation requires $24 + 12 = 36$ CNOT gates.

In summary, this example shows that bridging allows us to use parity compilation even on sparsely connected devices and can give an advantage over the standard gate model implementation. Specifically, problems with higher-order interactions are natural to solve with the parity approach, as recent benchmarks indicate [10].

VII. APPLICATION: CONSTRAINED OPTIMIZATION PROBLEMS

The parity architecture allows us to solve different types of constrained optimization problems without overhead. In Ref. [9] it was shown that problem constraints of product form (e.g., $\hat{\sigma}_z^{(3)}\hat{\sigma}_z^{(6)} = \hat{I}$) can directly be included when calculating the parity check matrix. Since each product constraint represents a single-parity qubit that is fixed to a certain value, this parity qubit can be left out and implicitly implemented using the parity constraints. These product constraints can therefore lead to a reduction in qubit and gate count, whereas in other implementations they can lead to further overhead. In Ref. [11], more general (polynomial) problem constraints were investigated. The main idea there is to use the Hamiltonian dynamics of the driver term to satisfy the problem constraints. In the initial state the spins are put in a state that satisfies the constraint and the driver only allows a specific kind of exchange between spins, which makes sure that all constraints stay satisfied during the algorithm. For all parity qubits which are part of such a polynomial constraint, the corresponding single-body $\hat{\sigma}_x$ term is removed from the driver Hamiltonian, and instead an exchange Hamiltonian of the form

$$H_{\text{exch.}} = \sum_{\langle i,j \rangle} \tilde{\sigma}_+^{(i)} \tilde{\sigma}_-^{(j)} + \text{H.c.} \quad (30)$$

is added, where $\langle i, j \rangle$ are pairs of neighboring parity qubits that span the polynomial constraint. Such an exchange interaction is available on different hardware platforms like for example neutral atoms [25,26] or superconducting qubits [27–29]. However, since it is typically only available locally, the qubits forming a polynomial constraint must be placed directly next to each other on the physical hardware. This sets a limit to the flexibility of qubit placement. Apart from the placement restrictions, the compilation process is not affected by polynomial constraints. The parity check matrix is

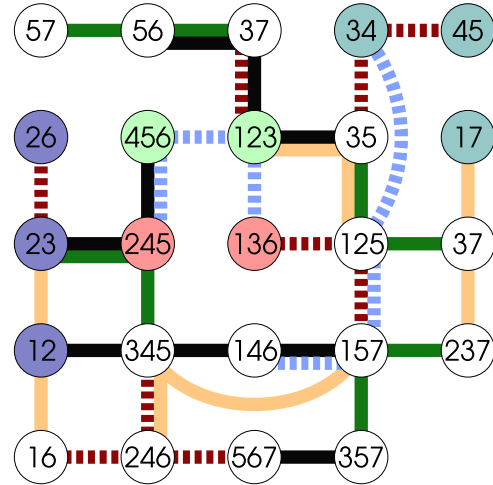


FIG. 7. Compilation of a constrained optimization problem with two product constraints and four general polynomial constraints, requiring connectivity only on a subset of the edges of a square lattice. The colored qubits are in polynomial constraints, where each of the four colors represents a different constraint, e.g., the blue (darkest) qubits implement a constraint $(\hat{\sigma}_z^{(1)}\hat{\sigma}_z^{(2)} + \hat{\sigma}_z^{(2)}\hat{\sigma}_z^{(3)} + \hat{\sigma}_z^{(2)}\hat{\sigma}_z^{(6)})|\psi\rangle = a|\psi\rangle$, where $a \in \{-3, -1, 1, 3\}$. The two product constraints $\hat{\sigma}_z^{(3)}\hat{\sigma}_z^{(6)}|\psi\rangle = |\psi\rangle$ and $\hat{\sigma}_z^{(1)}\hat{\sigma}_z^{(2)}\hat{\sigma}_z^{(7)}|\psi\rangle = |\psi\rangle$ are directly incorporated in the parity constraints, represented by solid and dashed colored lines. For example, the condition $\hat{\sigma}_z^{(3,6)} = \hat{I}$ is used in the black line in the bottom right to implement $\hat{\sigma}_z^{(5,6,7)}\hat{\sigma}_z^{(3,5,7)}\hat{\sigma}_z^{(3,6)}|\psi\rangle = \hat{\sigma}_z^{(5,6,7)}\hat{\sigma}_z^{(3,5,7)}|\psi\rangle = |\psi\rangle$. The curved lines show the two places where bridging is required to decompose the non-local interactions into nearest-neighbor CNOT gates.

independent of these constraints, as they are enforced exclusively with the driver Hamiltonian.

In Fig. 7 we assume a quantum device that can implement such exchange Hamiltonians and show an example of an optimization problem with both product constraints and polynomial constraints. Note that in this example not all connections between qubits are required. In cases where the qubits in a constraint can be connected in multiple ways, this reduced connectivity requirement may make it possible to route around missing links without overhead.

Due to the restrictions of local polynomial constraints, bridging is very useful when solving these kinds of problems. When optimizing the layout of qubits to make the parity constraints as local as possible, the polynomial constraints should be kept strictly local. This is achieved by only allowing optimization moves that preserve locality of the polynomial constraints.

VIII. DISCUSSION AND CONCLUSION

In this work, we presented methods to compile constrained optimization problems in the parity architecture, which also allows us to use the parity mapping on devices that are irregularly or sparsely connected. This is especially important for practical near-term systems, where the quality of qubits and connections is not guaranteed. We showed examples where our approach required a lower gate count than a standard direct implementation. This is an important result for near-term

²Note that this already assumes further optimization of the circuit; a naive implementation with swap gates would take 14 CNOT gates.

devices, where each gate introduces a significant amount of noise. The parity mapping always requires a qubit overhead, as the number of needed qubits is equal to the number of terms in the original Hamiltonian. This can be a limiting factor for practical devices. However, in many cases the number of qubits is not the main limiting factor and the lower gate count and decreased circuit depth can make it feasible to run larger and more complicated problems.

Finding a valid layout mapping where enough parity constraints are present to complete the mapping is now straightforward because even nonlocal constraints can be implemented without SWAP gates. The introduced construction via bridging gives clear rules while providing a lot of freedom in the implementation and thus room for optimization in the required number of gates and circuit depth. For example, solving the problem of finding minimal Steiner trees for arbitrary device connectivity remains a computationally difficult problem, but approximate algorithms can already lead to good results. The scaling of the circuit depth with problem and device size will depend on the specific problem and connectivity of the device. However, we note that this technique in general does not yield an upper bound to the circuit depth. If a lower circuit depth is desired, one can instead use plaquette-compilation methods like in Ref. [30], which have a higher overhead in qubit numbers but allow constant circuit depth, independent of the specific problem.

Improving the connectivity of the device will allow more local constraints, which in turn allows for more parallelization. The presented approach can thus be used both to improve the circuit implementation on a given device and to design future devices in a way that keeps the circuit depth manageable for relevant problems.

While, in this work, we demonstrated the power of the bridge technique to implement parity constraints, its application goes beyond the parity architecture and can also improve the implementation of many other nonlocal operators.

ACKNOWLEDGMENTS

The authors would like to thank Valentin Stauber for crucial feedback and discussions about minimal Steiner tree constructions, Michael Fellner for his detailed review and support with quantum circuit transpilers, and Sagar Kale for feedback about the manuscript. This study was supported by the Austrian Research Promotion Agency (FFG Project No. 884444, QFTE 2020). Work at the University of Innsbruck was supported by the Austrian Science Fund (FWF) through a START grant under Projects No. Y1067-N27 and No. I 6011. This research was funded in whole, or in part, by the Austrian Science Fund (FWF) SFB BeyondC Project No. F7108-N38. This project was funded within the QuantERA II Program that received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 101017733.

-
- [1] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell *et al.*, *Nature (London)* **574**, 505 (2019).
 - [2] A. D. King, S. Suzuki, J. Raymond, A. Zucca, T. Lanting, F. Altomare, A. J. Berkley, S. Ejtemaee, E. Hoskinson, S. Huang *et al.*, *Nat. Phys.* **18**, 1324 (2022).
 - [3] K.-N. Schymik, B. Ximenez, E. Bloch, D. Dreon, A. Signoles, F. Nogrette, D. Barredo, A. Browaeys, and T. Lahaye, *Phys. Rev. A* **106**, 022611 (2022).
 - [4] M. Sarovar, T. Proctor, K. Rudinger, K. Young, E. Nielsen, and R. Blume-Kohout, *Quantum* **4**, 321 (2020).
 - [5] P. Murali, D. C. McKay, M. Martonosi, and A. Javadi-Abhari, in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '20* (Association for Computing Machinery, New York, 2020), pp. 1001–1016.
 - [6] Z. Cai, R. Babbush, S. Benjamin, S. Endo, W. J. Huggins, J. R. McClean, Y. Li, and T. E. O'Brien, [arXiv:2210.00921](https://arxiv.org/abs/2210.00921).
 - [7] J. B. Hertzberg, E. J. Zhang, S. Rosenblatt, E. Magesan, J. A. Smolin, J.-B. Yau, V. P. Adiga, M. Sandberg, M. Brink, J. M. Chow, and J. S. Orcutt, *npj Quantum Inf.* **7**, 129 (2021).
 - [8] W. Lechner, P. Hauke, and P. Zoller, *Sci. Adv.* **1**, e1500838 (2015).
 - [9] K. Ender, R. ter Hoeven, B. E. Niehoff, M. Drieb-Schön, and W. Lechner, *Quantum* **7**, 950 (2023).
 - [10] M. Fellner, K. Ender, R. ter Hoeven, and W. Lechner, *Quantum* **7**, 952 (2023).
 - [11] M. Drieb-Schön, K. Ender, Y. Javanmard, and W. Lechner, *Quantum* **7**, 951 (2023).
 - [12] M. Fellner, A. Messinger, K. Ender, and W. Lechner, *Phys. Rev. Lett.* **129**, 180503 (2022).
 - [13] C. Chamberland, G. Zhu, T. J. Yoder, J. B. Hertzberg, and A. W. Cross, *Phys. Rev. X* **10**, 011022 (2020).
 - [14] A. C. Y. Li, M. S. Alam, T. Iadecola, A. Jahin, J. Job, D. M. Kurkcuoglu, R. Li, P. P. Orth, A. B. Özgüler, G. N. Perdue, and N. M. Tubman, *Phys. Rev. Res.* **5**, 033071 (2023).
 - [15] E. Farhi, J. Goldstone, and S. Gutmann, [arXiv:1411.4028](https://arxiv.org/abs/1411.4028).
 - [16] S. Hadfield, Z. Wang, B. O'Gorman, E. G. Rieffel, D. Venturelli, and R. Biswas, *Algorithms* **12**, 34 (2019).
 - [17] W. Lechner, *IEEE Trans. Quantum Eng.* **1**, 3102206 (2020).
 - [18] J. Unger, A. Messinger, B. E. Niehoff, M. Fellner, and W. Lechner, [arXiv:2211.11287](https://arxiv.org/abs/2211.11287).
 - [19] Qiskit contributors, :An open-source framework for quantum computing.
 - [20] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, *Quantum Sci. Technol.* **6**, 014003 (2020).
 - [21] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità, in *Proceedings of the Forty-Second ACM Symposium on Theory of Computing, STOC '10* (Association for Computing Machinery, New York, 2010), pp. 583–592.
 - [22] M. Hanan, *SIAM J. Appl. Math.* **14**, 255 (1966).
 - [23] M. Zachariasen, *Networks* **38**, 76 (2001).
 - [24] T. Huang and E. F. Y. Young, in *Proceedings of the 48th Design Automation Conference, DAC '11* (Association for Computing Machinery, New York, 2011), pp. 164–169.
 - [25] M. Anderlini, P. J. Lee, B. L. Brown, J. Sebby-Strabley, W. D. Phillips, and J. V. Porto, *Nature (London)* **448**, 452 (2007).

- [26] A. W. Glaetzle, M. Dalmonte, R. Nath, C. Gross, I. Bloch, and P. Zoller, *Phys. Rev. Lett.* **114**, 173002 (2015).
- [27] J. Koch, T. M. Yu, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, *Phys. Rev. A* **76**, 042319 (2007).
- [28] L. DiCarlo, J. M. Chow, J. M. Gambetta, L. S. Bishop, B. R. Johnson, D. Schuster, J. Majer, A. Blais, L. Frunzio, S. Girvin *et al.*, *Nature (London)* **460**, 240 (2009).
- [29] Y. Chen, C. Neill, P. Roushan, N. Leung, M. Fang, R. Barends, J. Kelly, B. Campbell, Z. Chen, B. Chiaro, A. Dunsworth, E. Jeffrey, A. Megrant, J. Y. Mutus, P. J. J. O'Malley, C. M. Quintana, D. Sank, A. Vainsencher, J. Wenner, T. C. White *et al.*, *Phys. Rev. Lett.* **113**, 220502 (2014).
- [30] R. ter Hoeven, B. E. Niehoff, S. S. Kale, and W. Lechner, [arXiv:2307.10626](https://arxiv.org/abs/2307.10626).