Quantum Information and Computation, Vol. 20, No. 7&8 (2020) 581–596

CNOT CIRCUIT EXTRACTION FOR TOPOLOGICALLY-CONSTRAINED QUANTUM MEMORIES

ALEKS KISSINGER[a]
Department of Computer Science, University of Oxford
Wolfson Building, Parks Road. Oxford OX1 3QD, United Kingdom

ARIANNE MEIJER-VAN DE GRIEND[b]
Cambridge Quantum Computing, 9a Bridge Street
Cambridge, CB2 1UB, United Kingdom

Received October 30, 2019
Revised May 22, 2020

Many physical implementations of quantum computers impose stringent memory constraints in which 2-qubit operations can only be performed between qubits which are nearest neighbours in a lattice or graph structure. Hence, before a computation can be run on such a device, it must be mapped onto the physical architecture. That is, logical qubits must be assigned physical locations in the quantum memory, and the circuit must be replaced by an equivalent one containing only operations between nearest neighbours. In this paper, we give a new technique for quantum circuit mapping (a.k.a. routing), based on Gaussian elimination constrained to certain optimal spanning trees called Steiner trees. We give a reference implementation of the technique for CNOT circuits and show that it significantly out-performs general-purpose routines on CNOT circuits. We then comment on how the technique can be extended straightforwardly to the synthesis of CNOT+Rz circuits and as a modification to a recently-proposed circuit simplification/extraction procedure for generic circuits based on the ZX-calculus.

Keywords: quantum circuits, quantum compilation, NISQ, Steiner trees
Communicated by: R Jozsa & M Mosca

1 Introduction

Quantum circuits give a *de facto* standard for representing quantum computations at a low level. They consist of sequences of primitive operations, called quantum gates, applied to a register of quantum bits, or qubits. Increasingly, noisy intermediate-scale quantum (NISQ) computers with 10-80 qubits are becoming a reality. Popular physical realisations such as superconducting quantum circuits [1, 2, 3] and ion traps [4, 5, 6, 7] consist of qubits stored in the physical states of systems arranged in space, where two-qubit operations are typically only possible between pairs of adjacent systems. Hence, when it comes to actually running a quantum computation on these architectures, logical qubits must be mapped to physical memory locations, and the circuit must be modified to only consist of 2-qubit operations between adjacent qubits in the physical architecture. Naïvely, this can be achieved by simply

[a] aleks.kissinger@cs.ox.ac.uk
[b] ariannemeijer@gmail.com

inserting swap gates to move a pair of qubits next to each other before each 2-qubit operation. However, this approach comes with an enormous overhead in terms of 2-qubit operations, each of which introduces a great deal more noise than a single qubit operation on most realistic architectures [5]. More sophisticated approaches incorporate techniques from computer aided design [8] and machine-learning [9] in order to minimise the extra operations needed by making good choices of initial and intermediate memory locations for the qubits involved. Nevertheless, these are simply refinements of the basic 'search and swap' approach. Most approaches only take the topological structure of the circuit into account (i.e. which qubits are being acted upon) rather than semantic structure (i.e. the unitary being implemented), and hence miss out on opportunities for more efficient circuit mapping.

We present a new approach to quantum circuit mapping based on constrained Gaussian elimination, and apply it in the simplest case of mapping CNOT circuits. The main idea is to modify a familiar technique for synthesising CNOT circuits using Guassian elimination in such a way that primitive row operations (i.e. CNOT gates) are only allowed between certain rows corresponding to neighbouring qubits. Hence, non-local row operations must be propagated through intermediate rows. We then give a simple strategy for identifying and using appropriate intermediate rows based on certain minimal spanning trees called Steiner trees. Once we produce such a spanning tree, we use CNOTs to propagate row operations down toward the leaves then ultimately back up toward the root. The end result is a CNOT circuit realising a given parity map involving only nearest-neighbour interactions. To measure the effectiveness of our approach, we produce many random CNOT circuits on 9, 16, and 20 qubits, containing between 3 and 256 CNOT gates, and map them onto 5 different graph topologies: 3×3 and 4×4 square lattices, 16-qubit architectures of the IBM QX-5 and Rigetti Aspen devices, and the 20-qubit IBM Q20 Tokyo architecture. To compare the performance of our technique to general-purpose mapping techniques, we also map these CNOT circuits with the Rigetti QuilC compiler and $\mathsf{t}|\mathsf{ket}\rangle$ by Cambridge Quantum Computing. We chose these tools because they scale well to our larger test circuits and give state of the art results on a large set of benchmark circuits published by IBM [10]. Using these as a baseline, we find an average savings in 2-qubit gates of 48% over QuilC and 36% over $\mathsf{t}|\mathsf{ket}\rangle$.

Since CNOTs and single-qubit operations are universal for quantum computation, this already extends to a routine for mapping generic circuits: simply apply our routine to CNOT-only sub-circuits. However, only circuits containing long sequences of CNOT gates are likely to benefit from this naïve approach. To address this issue, we will discuss in Section 4 how the techniques we describe can be extended to synthesis of more general families of circuits. We note that our technique for CNOT circuits extends straightforwardly to circuits consisting of CNOT and Z-phase gates using the *phase polynomial* representation of these circuits (see e.g. [11]). We also suggest a technique for mapping universal circuits by modifying recently-proposed method by one of the authors [12] based on the ZX-calculus [13]. The extraction method from [12] has been implemented (without topological constraints) in the PyZX [14] circuit optimiser, which has already been very successful in reducing T-count for general Clifford+T circuits [15].

The paper is structured as follows. In Section 2.1 we give a brief background on the theory of CNOT circuits and parity maps and provide the definitions of Steiner trees and descending Steiner trees, which we will use in our synthesis algorithm. The algorithm itself is described in

Section 2.2, for the simpler case where the CNOT connectivity graph contains a Hamiltonian path (as in all five of our benchmark architectures). In Section 3, we give the results of our CNOT mapping procedure in five different architectures and compare performance to the Quilc and t|ket⟩ compilers. We then describe two extensions in Section 4: to general graphs and to general circuits.

Related work. Most existing circuit mapping techniques are based on searching for the optimal placement of swap gates and qubits. This can be described mathematically and solved with a general solver or temporal planner [16, 17]. However, the search space for finding optimal swap gates is exponential and these exact techniques will be intractable for larger NISQ devices [18]. Thus, most recent approaches use heuristics to reduce the search space [19, 18, 10]. This includes the IBM-QX contest-winning technique that is based on the A*-search algorithm [8]. Ref. [20] gives an approach for realising arbitrary parity-function oracles, subject to topological constraints, which are a special case of the family of maps we consider. With most of these techniques, the size of the resulting circuit is very sensitive to the original placement of the logical qubits on the device [21]. Although algorithms have been proposed to find an optimal initial placement *a priori* [21], mapping techniques that have the freedom to build the initial placement while routing find a better initial state [8]. Circuits can be routed even better if the initial mapping is adjusted based on the fully routed circuit [19].

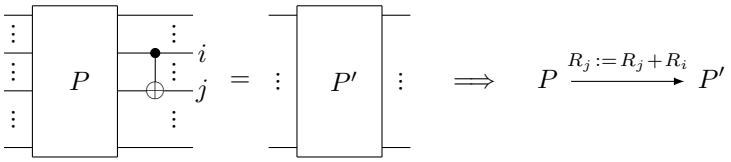
The authors of [22] have recently introduced a technique very similar to ours, which was developed independently. Both their article and a preprint of this paper appeared online within a few days of each other (our preprint [23] on 1/4/2019; [22] on 3/4/2019). The two approaches differ in that we give a slightly different strategy for preserving the upper triangular form based on Hamiltonian paths (which we then generalise to a recursive version based on a depth-first ordering). We also focus primarily on the case of CNOT circuits and sketch extensions to the case of CNOT+Rz and universal circuits using phase polynomials and the ZX-calculus, respectively. Ref. [22] covers the CNOT and CNOT+Rz cases in detail, also relying on phase polynomials for the latter. We additionally optimise over initial qubit locations, which produces signficant improvements over the baseline compilers even for low CNOT counts (e.g. $> 50\%$ improvement over Quilc with 8 CNOTs on IBM Q20). Finally, we compare performance relative to state of the art general-purpose circuit compilers, whereas [22] uses the CNOT circuit synthesis described in [24], along with a naïve mapping procedure, as its baseline.

2 Methods

2.1 Background: Parity maps and Steiner trees

Our approach to CNOT mapping is based on re-synthesising the CNOT circuit from its corresponding parity map. By a *parity map*, we mean any reversible linear map on bitstrings. That is, we mean a bijective mapping from N-bitstrings to N-bitstrings where each bit in the output is a parity (i.e. XOR) of the input bits. It is a well-known fact that such maps exactly correspond to the action of CNOT circuits on computational basis states. It is therefore convenient to represent the action of a CNOT circuit on N qubits as an $N \times N$ matrix over GF(2).

If we consider an arbitrary such parity map, it is straightforward to check that post-composing a CNOT gate with a control on the j-th qubit and the target on the k-th qubit has the overall effect of adding the j-th row to the k-th row:



Hence, there is an evident way to construct a CNOT circuit that realises an arbitrary parity matrix P. Simply perform Gauss-Jordan elimination on P, post-composing CNOTs for each primitive row operation. In the end, we will obtain $CP = 1$, where C is a known CNOT circuit. Then, $P = C^{-1}$, where C^{-1} is obtained from C just by reversing the order of CNOT gates. In other words, in order to synthesise a CNOT circuit which realises parity map P, we simply perform Gauss-Jordan and store the primitive row operations used. Then the CNOT circuit corresponds exactly to that sequence of row operations, in reverse order. This technique, when combined with a simple heuristic for choosing appropriate row operations, is able to obtain asymptotically optimal CNOT realisations of a given parity map [24].

In this paper, we modify the question: how can we construct a CNOT realisation of an arbitrary parity map if only certain CNOT gates are allowed? For example, suppose we have 9 qubits arranged in a 3×3 grid, and we only wish to allow CNOTs between nearest neighbours. Clearly this problem is equivalent to asking: how do we perform Gauss-Jordan elimination on a given GF(2)-matrix, when only certain primitive row operations are allowed?

Clearly, our only recourse is to use *more* row operations in order to allow distant rows to essentially be added together via some intermediate steps. We present a strategy for doing this based on special kinds of spanning trees called Steiner trees.

Note that, by a *graph* we always mean an undirected, simple graph with no self-loops, by a *tree* we mean a graph with no cycles, and by a *rooted tree* we mean a tree with a chosen vertex called the *root*. For rooted trees, we use the standard terminology of *parent* and *child* to denote vertices adjacent to a given vertex which are closer to or farther from the root, respectively.

Definition 1 *For a connected graph G and a subset S of the vertices V_G of G, a Steiner tree T is a minimal subgraph of G such that T is a tree containing all vertices in S.*

Computing Steiner trees is NP-hard in general and the related decision problem is NP-complete [25]. Indeed, the Steiner tree problem can be seen as a generalisation of the travelling salesperson problem, which allows an arbitrary tree to span a set of vertices rather than a single path, which can be seen as a tree with no branching. In practice, we will not need exactly optimal Steiner trees for our strategy to work, and many efficient heuristics exist for computing approximate Steiner trees [26, 27]. For our purposes, we use a very simple heuristic based on a combination of the Floyd-Warshall all-pairs shortest-path algorithm [28] and minimal spanning trees.

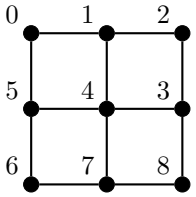
A useful refinement of Steiner trees will be the following notion.

Definition 2 *For a connected graph G, a total ordering $\leq$ of the vertices V_G, and a subset $S \subseteq V_G$, a decreasing Steiner tree T is a minimal rooted subtree of G such that $S \subseteq V_T$ and every vertex in T is larger its children with respect to $\leq$.*

2.2 Constrained CNOT circuit extraction

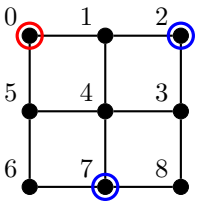
In this section, we will describe the algorithm STEINER-GAUSS, which performs Gauss-Jordan elimination of a parity matrix using only nearest-neighbour row operations for a given graph G. Consequently, this procedure can be used to synthesise a CNOT circuit implementing a given parity map using only nearest-neighbour CNOTs. The algorithm itself consists of two phases, STEINER-DOWN and STEINER-UP, which respectively produce an upper triangular matrix and produce the identity matrix from an upper triangular matrix.

Initially, we will consider only graphs G which have a Hamiltonian path, i.e. graphs G which contain a connected path P that visits each of the vertices in G exactly once. For example, the 3×3 grid:

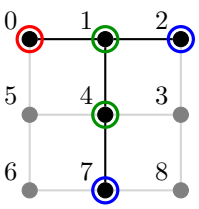
$G :=$ 

has a Hamiltonian path given by $[0, 1, 2, \ldots, 8]$. We will assume this path also provides a total ordering $\leq$ on the vertices of G. We will remove the assumption of a Hamiltonian path in the next section by providing a recursive algorithm capable of handling arbitrary graphs.

We begin by labelling the rows of our parity map P by the vertices of the constraint graph G. The first stage of our algorithm, STEINER-DOWN, computes an upper triangular matrix. To do this, we wish to remove the non-zero elements below the diagonal. We do this one column at a time, starting with column $k := 0$ and proceeding left to right. Let S be the set containing k itself, as well as all of the vertices j such that $j > k$ and $P_{jk} = 1$. That is, S contains the diagonal element and all of the rows which contain 1s below the diagonal. For example, in the following parity map, $S = \{0, 2, 7\}$:

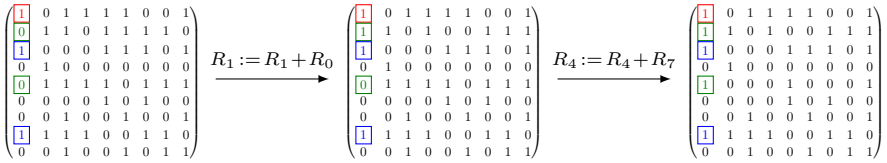
$$P = \begin{pmatrix} \boxed{1} & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ \boxed{1} & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ \boxed{1} & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} \qquad G := \quad 

These vertices are not adjacent in the graph, hence when we compute the Steiner tree T containing S, we get some extra vertices, corresponding to rows that have 0s below the diagonal:

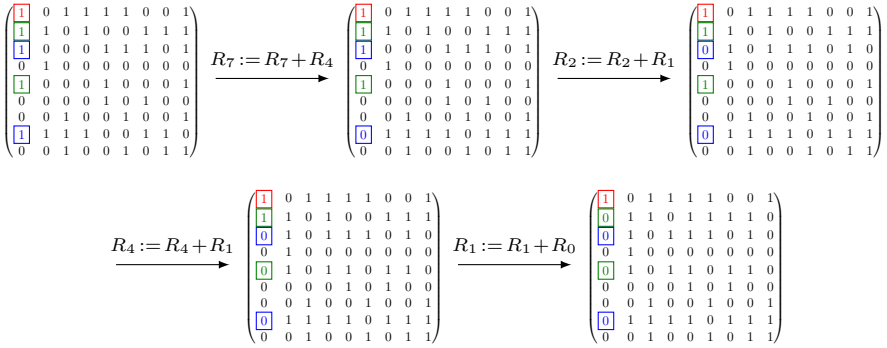
$$P = \begin{pmatrix} \boxed{1} & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ \boxed{0} & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ \boxed{1} & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \boxed{0} & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ \boxed{1} & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} \qquad G := \quad 

(1)

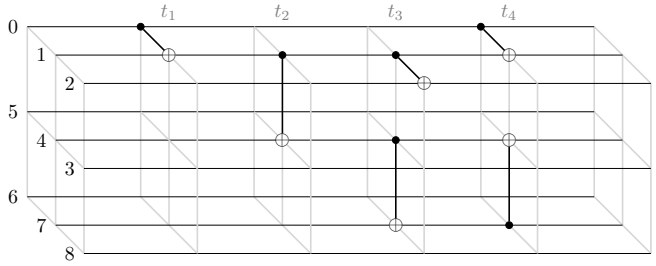
These extra vertices which need to be added to get a spanning tree are sometimes called *Steiner points*. We consider the numbers in boxes above as decorating the corresponding vertices of the Steiner tree T. Initially there are some 0s in the Steiner tree corresponding to Steiner points (and possibly the diagonal element), so we first 'fill' the Steiner tree. That is, we add a row with a 1 to any neighbouring row in T with a 0. Since the tree is connected, after finitely many iterations this will propagate 1s into every location in T:



After that, we can 'empty' the Steiner tree by setting every location except for the diagonal to zero. We do this by regarding the diagonal as the root of the tree. For each leaf v in T with parent w, perform the row operation $R_v := R_v + R_w$, then remove v from T. This terminates when there is only one vertex left in T, the root.



Since we only perform row operations along edges of T, which are a subset of the edges of G, the corresponding CNOTs in the circuit we synthesise will only be between neighbouring qubits. For example, the six row operations above (read from right to left) yield the following part of a CNOT circuit:



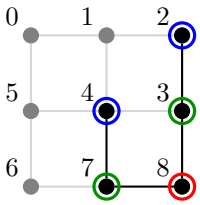
Note that in this phase, we have some freedom to choose which row operations to perform. Here we have taken a greedy strategy for maximising the number of row operations that can be done in parallel.

Having put the first column in upper triangular form, we delete the corresponding root vertice from G and then proceed to the next column, building up our CNOT circuit from

right-to-left. Since we proceed in order along a Hamiltonian path, the graph never becomes disconnected and always has a Hamiltonian path. Hence it is always possible to find a Steiner tree for any combination of points. STEINER-DOWN terminates after we remove the last vertex from G with a matrix in upper triangular form.

The second stage of our algorithm, STEINER-UP, starts with the original graph G and a parity map P in upper triangular form and removes any 1s that are above the diagonal, yielding the reduced echelon form (which in our case is always an identity matrix). It works in almost the same way, except that some care must be taken not to destroy the existing upper triangular structure. To do this, we must always perform *decreasing* row operations. That is, we must perform operations of the form $R_j := R_j + R_i$ only when $j < i$. This is where Definition 2 comes in. Starting with the last column, let the set S consist of the diagonal element and the rows which contain 1s above the diagonal. We then compute a decreasing Steiner tree for S whose root is the diagonal element. A decreasing Steiner tree can be easily created by giving the edges in graph G a direction from larger nodes to smaller ones. Such a descreasing Steiner tree always exists, because in the worse case we can just take the Hamiltonian path for T, but in general we can take shortcuts. For example:

$$P = \begin{pmatrix} 1&0&1&1&1&1&0&0&0\\ 0&1&1&0&1&1&1&1&0\\ 0&0&1&1&0&0&1&0&1\\ 0&0&0&1&1&1&0&1&0\\ 0&0&0&0&1&0&0&1&1\\ 0&0&0&0&0&1&0&1&0\\ 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1 \end{pmatrix} \qquad G :=$$



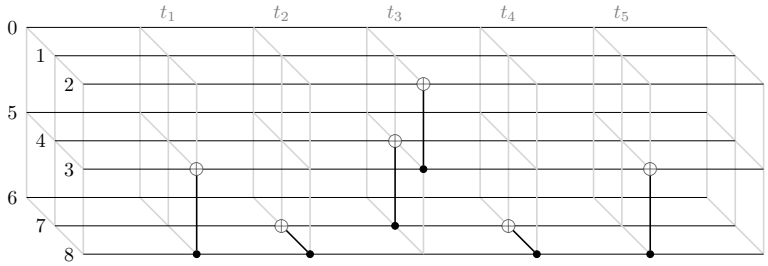
Here, we have $S = \{2, 4, 8\}$ and T has vertices $\{2, 3, 4, 7, 8\}$. Note there is indeed a smaller Steiner tree which does not contain vertex 7, but in that case 4 would need to be a child of 3, hence it is not a decreasing Steiner tree.

Once we have a decreasing Steiner tree, we can 'fill' the tree with 1s by decreasing row operations. This is always possible since at this stage, the root always contains a 1 and all edges from parents to children are decreasing. We can then 'empty' the tree again just as we did in STEINER-DOWN, noting that every row operation at this stage is applied from a parent to its child (and hence is decreasing). The resulting row operations in the above example are thus:

$$\begin{pmatrix} 1&0&1&1&1&1&0&0&0\\ 0&1&1&0&1&1&1&1&0\\ 0&0&1&1&0&0&1&0&1\\ 0&0&0&1&1&1&0&1&0\\ 0&0&0&0&1&0&0&1&1\\ 0&0&0&0&0&1&0&1&0\\ 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1 \end{pmatrix} \xrightarrow{R_3 := R_3 + R_8} \xrightarrow{R_7 := R_7 + R_8} \xrightarrow{R_2 := R_2 + R_3} \cdots$$

$$\cdots \xrightarrow{R_4 := R_4 + R_7} \xrightarrow{R_7 := R_7 + R_8} \xrightarrow{R_3 := R_3 + R_8} \begin{pmatrix} 1&0&1&1&1&1&0&0&0\\ 0&1&1&0&1&1&1&1&0\\ 0&0&1&0&1&1&1&1&0\\ 0&0&0&1&1&1&0&1&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&1&0\\ 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1 \end{pmatrix}$$

We can therefore prepend the following section to our CNOT circuit:



Once a column is done, we can delete it from G and take the next highest column. Again, since we traverse along a Hamiltonian path (backwards this time), the graph G never becomes disconnected and always has a Hamiltonian path. STEINER-UP then terminates with P equal to the identity matrix. The corresponding CNOT circuit then implements the original parity map using only nearest-neighbour CNOT gates.

Much like other approaches, the size of the final circuit is very sensitive to the initial layout of the logical qubits within the physical architecture, especially for relatively small CNOT circuits. For us, a good choice of qubit positions means smaller Steiner trees, which in turn means that fewer extra CNOTs are added. The placement of qubits on the architecture is equivalent to permuting the rows and columns of the parity map before applying the algorithm described above. Hence, we use a genetic algorithm [29] to find an optimal permutation such that the resulting circuit contains as little gates as possible.

3 Results

We work with a fixed set of randomly-generated CNOT circuits on 9, 16, and 20 qubits. The 9-qubit CNOT circuits have either 3, 5, 10, 20 or 30 gates, whereas the other circuits have either 4, 8, 16, 32, 64, 128 or 256 gates. For each of these gate counts, we generated 20 different random circuits, yielding a test set of 380 random circuits.

We compared our algorithm to the QuilC [30] and $\mathsf{t}|\mathsf{ket}\rangle$ [10] compilers. The former uses CZ gates as basic two-qubit gates instead of CNOT gates. Since a CNOT gate can be formed by conjugating the target bit of a CNOT gate with Hadamards, the amount of CZ gates can be compared directly to the amount of CNOT gates.

As architectures, we used a 9-qubit square grid, the 20-qubit IBM Q20 Tokyo, a 16-qubit square grid, the IBM QX5 and the Rigetti 16Q Aspen architectures. Their respective gate counts and percentage of added gates (overhead) can be found in Table 1 [c]. For the genetic algorithm, we used different parameters depending on the size of the architecture. For the 9-qubit architecture, we used a population of 30 and 15 iterations. For the 16-qubit architectures, we used a population of 50 and 100 iterations. And for the 20-qubit architecture, we used a population of 100 and 100 iterations. The crossover and mutation probability had a constant value of 0.8 and 0.2, respectively. The values population and iteration were simply found by trial and error, and could probably be tuned further. A larger population and more iterations improves the chances of finding a better permutation, but increases the time that

[c] All circuits can be found in QASM format at:
https://github.com/Quantomatic/pyzx/tree/steiner_decomp/circuits/steiner

Architecture	**#**	**QuilC**		**t|ket⟩**		**Steiner**		**<QuilC**	**<t|ket⟩**
9q-square	3	3.8	(27%)	3.6	(20%)	3	(**0**%)	21%	17%
9q-square	5	10.82	(116%)	6.4	(28%)	5.2	(4%)	52%	19%
9q-square	10	20.08	(101%)	16.95	(70%)	11.6	(16%)	42%	32%
9q-square	20	46.24	(131%)	40.75	(104%)	25.85	(29%)	44%	37%
9q-square	30	72.89	(143%)	66.15	(121%)	35.55	(19%)	51%	46%
16q-square	4	6.14	(54%)	5.8	(45%)	4.44	(11%)	28%	23%
16q-square	8	19.68	(146%)	12.95	(62%)	12.41	(55%)	37%	4%
16q-square	16	48.13	(201%)	36.2	(126%)	33.08	(107%)	31%	9%
16q-square	32	106.75	(234%)	94.45	(195%)	82.95	(159%)	22%	12%
16q-square	64	225.69	(253%)	203.75	(218%)	147.38	(130%)	35%	28%
16q-square	128	457.35	(257%)	436.25	(241%)	168.12	(31%)	63%	61%
16q-square	256	925.85	(262%)	922.65	(260%)	169.28	(**-34**%)	82%	82%
rigetti-16q-aspen	4	7.05	(76%)	7.15	(79%)	4.15	(4%)	41%	42%
rigetti-16q-aspen	8	28.2	(253%)	17.2	(115%)	11.22	(40%)	60%	35%
rigetti-16q-aspen	16	69.15	(332%)	52	(225%)	33.95	(112%)	51%	35%
rigetti-16q-aspen	32	147.3	(360%)	144.95	(353%)	101.75	(218%)	31%	29%
rigetti-16q-aspen	64	324.6	(407%)	322.85	(404%)	189.15	(196%)	42%	41%
rigetti-16q-aspen	128	664.65	(419%)	666.15	(420%)	220.75	(72%)	67%	66%
rigetti-16q-aspen	256	1367.89	(434%)	1361.6	(432%)	222.15	(**-13**%)	84%	83%
ibm-qx5	4	6.75	(69%)	4.3	(8%)	4	(**0**%)	41%	7%
ibm-qx5	8	23.7	(196%)	14.75	(84%)	8.95	(12%)	62%	39%
ibm-qx5	16	60.5	(278%)	47.5	(197%)	26.55	(66%)	56%	44%
ibm-qx5	32	140.05	(338%)	122.95	(284%)	84.4	(164%)	40%	31%
ibm-qx5	64	301.05	(370%)	278.7	(335%)	152.65	(139%)	49%	45%
ibm-qx5	128	600.9	(369%)	597.65	(367%)	188.25	(47%)	69%	69%
ibm-qx5	256	1247.8	(387%)	1258.8	(392%)	193.8	(**-24**%)	84%	85%
ibm-q20-tokyo	4	5.5	(38%)	6.05	(51%)	4	(**0**%)	27%	34%
ibm-q20-tokyo	8	17.3	(116%)	12	(50%)	7.69	(**-4**%)	56%	36%
ibm-q20-tokyo	16	43.83	(174%)	29.05	(82%)	20.44	(28%)	53%	30%
ibm-q20-tokyo	32	93.58	(192%)	78.15	(144%)	66.93	(109%)	28%	14%
ibm-q20-tokyo	64	215.9	(237%)	181.25	(183%)	165.6	(159%)	23%	9%
ibm-q20-tokyo	128	432.65	(238%)	391.85	(206%)	237.64	(86%)	45%	39%
ibm-q20-tokyo	256	860.74	(236%)	789.3	(208%)	245.84	(**-4**%)	71%	69%

Table 1. The average number of CNOTs needed to map random circuits containing '#' CNOT gates. The first column shows the architecture mapped to and the second column the original number of CNOT gates. The remaining columns show the average 2-qubit gate count after mapping 20 random circuits and percent improvement in total CNOT count of our approach vs. QuilC and t|ket⟩. The percentages in parentheses show mapping overheads (i.e. percentage of the original gate count that was added during mapping), where negative overheads indicate a mapped circuit that is smaller than the original.

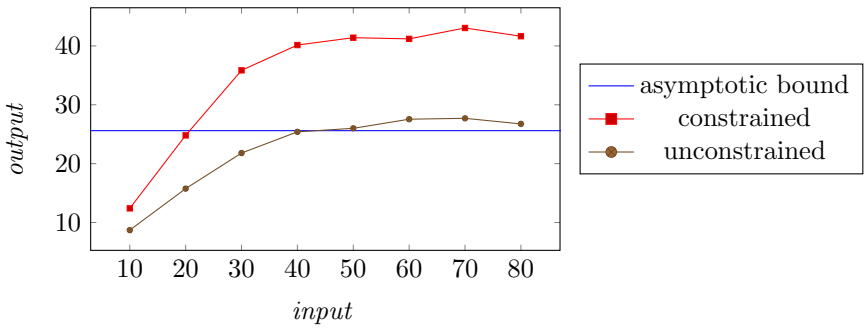


Fig. 1. A plot of input vs. output CNOT gates when mapping on to a 9-qubit square grid, when computing and re-synthesising the circuit, both in the unconstrained case using the method described in [24] and respecting nearest-neighbour constraints using our method.

the algorithm needs to run. Our running times range from a few seconds to about a minute on a laptop with a core i7 8550, 1.8 GHz. CPU and 16 GB of RAM.[d]

Note that the overheads of our mapping process are sometimes negative. This is because the process we use computes the parity map associated to a CNOT circuit and re-synthesises the circuit using Gaussian elimination, as described in Sections 2.1 and 2.2. In the unconstrained case, Patel *et al.* [24] gave a heuristic for (re-)synthesising generic CNOT circuits on n qubits with $O(n^2/\log(n))$ CNOTs. Indeed, using their algorithm for 9-qubit circuits, we see in Fig. 1 that the average CNOT count stabilises around the asymptotic bound of $9^2/\log_2(9) \approx 25.6$. Our approach also stabilises, but at a higher number of CNOTs (~ 42 for 9 qubits).

As a point of comparison, a naïve mapping procedure, such as the baseline procedure described in [22], introduces an overhead of $4(d-1)$ CNOT gates per gate introduced by the Patel *et al.* synthesis algorithm, where d is the length of the shortest path between two qubits. Since the average Manhattan distance on a 9-qubit square is $d = 2$, we expect the naïve method to stabilise at $25.6 \cdot 4 \cdot (2-1) \approx 102$ CNOT gates on the 9-qubit square.

Also note that the QuilC and t|ket⟩ baselines also attempt to optimise the given circuit while routing, but these optimisation are mainly local. Without a semantic description of the circuit, it is difficult to find global optimisations automatically. Similarly, without attempting to find an optimal qubit placement, our method and unconstrained Gauss-Jordan might increase the CNOT count for very sparse circuits [24].

4 Extensions

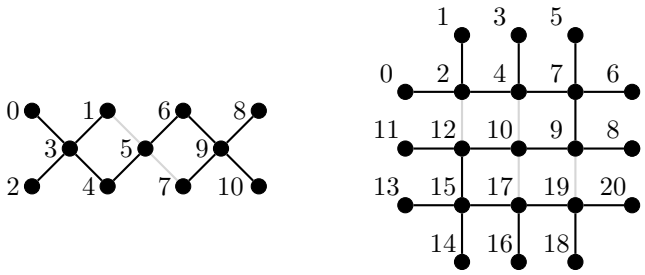
4.1 Mapping to general graphs

We can extend our algorithm, described in Section 2.2, to graphs that do not contain a Hamiltonian path using a recursive version. For this, we will define parametrised versions, STEINER-DOWN(k, V) and STEINER-UP(k, V, W), of the procedures from Section 2.2. In both cases, the procedures now take a parameter k which gives the current vertex/column on which to perform the downward or upward Gaussian elimination, respectively. They are also given a set $V \subseteq V_G$ over which all operations are restricted. Hence, rather than progressively deleting

[d]The python implementation of our algorithm can be found at:
https://github.com/Quantomatic/pyzx/tree/steiner_decomp/pyzx/routing

vertices from G, we start with $V := V_G$ and progressively remove vertices from V. The second procedure, STEINER-UP, also has a parameter $W \subseteq V$ where the Steiner tree and row operations are allowed to be non-descending. That is, all edges in the rooted Steiner tree T must be descending unless both vertices adjacent to that edge are in W. Similarly, we require that row operations should be descending unless the rows corresponding to both vertices are in W.

Let G be an undirected graph and R a spanning tree. Fix some leaf of R and number the vertices consecutively by depth-first traversal (DFT), ensuring that each vertex is labelled *after* its children (i.e. post-ordering). We then choose vertex 0 as the root of R. Such a numbering has the property that removing vertices in ascending order never results in a disconnected graph. Here are some examples:



Note that, since we use a post-ordering, the starting point of the DFT will *not* end up as the root of R. In the graphs above, the starting points for the DFT end up labelled 10 and 20, respectively, whereas the root of R is always taken to be 0.

For a parity map P whose rows are labelled by the vertices of G, the recursive version of the previous procedure, called STEINER-GAUSS-REC, is defined as follows:

1. For each $k \in G$ (in ascending order), apply STEINER-DOWN(k, V_R).
2. If R is empty, we are done. Otherwise pick the maximal vertex $k \in R$ and maximal leaf $k' \in R$.
3. Let W be the set of vertices in the shortest path from k' to k (inclusive). Apply STEINER-UP(k', V_R, W).
4. Apply STEINER-GAUSS-REC on the subgraph of G restricted to W.
5. Remove k' from R and go to 2.

Note that, when the spanning tree R is actually a Hamiltonian path, it will always be the case that $k = k'$, hence the recursive part is trivial and it reduces to the algorithm described in Section 2.2.

We now argue that this recursive algorithm terminates with P reduced to the identity. This is clearly true if R is empty and P is already the identity matrix, so assume for the sake of induction this holds for all R with fewer than n nodes.

Now, assume we have a parity matrix P and R has n nodes. After step 1, P is in upper-triangular form. Note, we only ever delete leaves of the spanning tree R, so our choice of numbering guarantees that there is a decreasing path from the maximal vertex in R to any

other vertex in R. Hence, there is a path from the maximal leaf k' to any node in R that is only increasing on the shortest path from k' to k, which we fix as W in step 2. Applying STEINER-UP(k', V_R, W) therefore succeeds in eliminating all of the 1s above the node k'.

Since STEINER-UP only allows downward row additions on rows corresponding to vertices in W, the rest of P stay in upper triangular form. That is, if we restrict to 'unfinished' rows and columns (i.e. those in R), P looks like this:

$$P = \begin{pmatrix} \boxed{P'} & \\ \boxed{0} & \boxed{B} \end{pmatrix}$$

where P' is upper triangular and B consists of the rows and columns in W. We then recurse on to W, which necessarily has fewer than n nodes (otherwise the maximal vertex k would have been a leaf). By induction hypothesis, the block B is reduced to the identity. Hence k' has zeros on its entire row and column, and a 1 on the diagonal. Thus k' is 'finished', so we remove it from R and iterate. R now has fewer than n nodes, so we can invoke the induction hypothesis again to finish the proof.

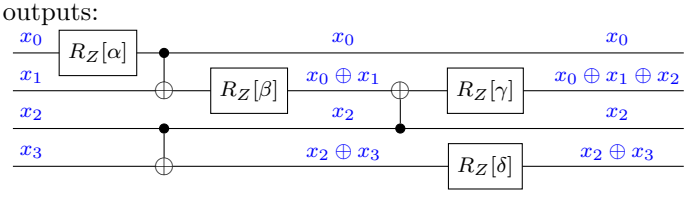
4.2 Mapping from general circuits

The technique we proposed for CNOT circuits extends in one obvious way to general circuits. If we decompose circuits into a gate set where CNOT is the only 2-qubit gate, we can simply do constrained synthesis of all of the CNOT sub-circuits. However, by sub-circuits independently, we might lose out on global structure that can be exploited to obtain more efficient output circuits. In this section, we briefly explain two generalisations that enable us to directly synthesise more general families of circuits: namely CNOT+Rz circuits and even generic Clifford+Rz circuits. We first explain the simpler case, as it does not require going outside of the circuit model. By Rz gates, we mean a Z-phase rotation by a generic phase α:

$$R_Z[\alpha] := \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix}$$

Certain special cases have common names in quantum circuit literature, e.g. $S := R_Z[\frac{\pi}{2}]$ and $T := R_Z[\frac{\pi}{4}]$. As explained in e.g. [11], circuits consisting of CNOT gates and arbitrary Z-phase rotations can be described efficiently in two parts: a GF(2)-linear map describing the action of the circuit on basis states and a set of pairs of the form (α, v) where $\alpha \in [0, 2\pi)$ is an angle and v is a vector in GF(2) describing the *parity* of input states on which that angle is applied.

First, note that it is straightforward to efficiently calculate the behaviour of CNOT+Rz circuits on computational basis states using *annotated circuits* [31]. One labels the input wires by variables $(x_0, \ldots, x_{n-1})$ and propagates these labels from left-to-right using the rule that $R_Z[\alpha]$ does not change the labels, whereas CNOT sends labels (x, y) on its inputs to labels $(x, x \oplus y)$ on its outputs:



We can then read off the output basis state from the final labels on the wires and the parities associated with each phase from the wire that the phase gate is on. For example, the circuit above acts as follows on computational basis states:

$$|x_0, x_1, x_2, x_3\rangle \mapsto e^{i[\alpha\cdot x_0+\beta\cdot(x_0\oplus x_1)+\gamma\cdot(x_0\oplus x_1\oplus x_2)+\delta(x_2\oplus x_3)]}|x_0, x_0\oplus x_1\oplus x_2, x_2, x_2\oplus x_3\rangle \quad (2)$$

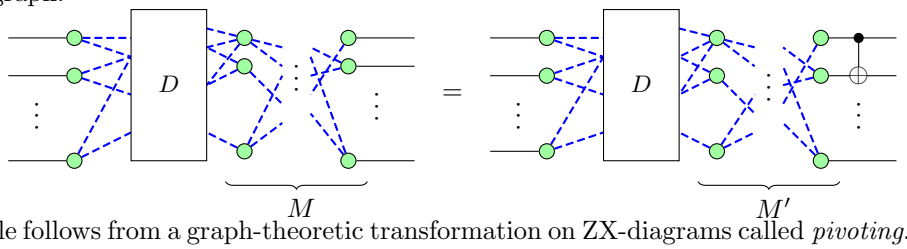
for all $x_i \in \{0,1\}$. The expression ϕ in $e^{i\phi}$ above dictates how the phase depends on the input basis state, and is sometimes referred to as a *phase polynomial*. Clearly the relevant data for the overall unitary is the (GF(2)-linear) action on basis states as well as this phase polynomial. We can then represent a parity of input variables as a bitstring, e.g. $x_0\oplus x_2\oplus x_3$ is the bitstring $(1,0,1,1)$, indicating this parity depends on the first, third, and forth input variables. Hence the data associated with the map (2) is a parity matrix P acting on basis states and a set of angle/vector pairs $\mathcal{P}$ giving the terms of the phase polynomial:

$$\left(P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}, \quad \mathcal{P} = \left\{ \alpha\cdot\begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \ \beta\cdot\begin{pmatrix}1\\1\\0\\0\end{pmatrix}, \ \gamma\cdot\begin{pmatrix}1\\1\\1\\0\end{pmatrix}, \ \delta\cdot\begin{pmatrix}0\\0\\1\\1\end{pmatrix} \right\} \right)$$

Much like the CNOT case, there is an evident circuit extraction procedure based on Gaussian elimination for CNOT+Rz maps. Since applying a CNOT gate changes the labels on wires, it can be seen as applying a primitive row operation simultaneously to all of the vectors in $\mathcal{P}$. Hence, following [32], we can paritition vectors in $\mathcal{P}$ into independent sets $I_1, \ldots I_m$. For each set I_k, we can apply Gaussian elimination to reduce the parity vectors in I_k to unit vectors then apply $R_Z[\alpha]$ on the i-th qubit for each pair $\alpha\cdot e_i \in I_k$. If this Gaussian elimination is performed using the STEINER-GAUSS procedure, these primitive row operations, and hence the synthesised circuit, will respect nearest-neighbour constraints, just as in the CNOT case.

An alternative approach to synthesising CNOT+Rz circuits was proposed by [22], which uses Steiner trees to build a connectivity-restricted version of the 'Gray-synth' algorithm proposed in [32].

A very similar procedure to (unconstrained) CNOT+Rz circuit synthesis using Gaussian elimination was described by one of the authors in [12] as a means of extracting a quantum circuit from a simplified tensor-network representation called a *ZX-diagram*. The extraction phase proceeds from right-to-left on a directed acyclic graph, and exploits the fact that post-composing CNOT gates is able to perform primitive row operations on the adjacency matrix of the graph:



This rule follows from a graph-theoretic transformation on ZX-diagrams called *pivoting*. While the details of how this actually works are not relevant here, an important observation is that this procedure makes use of Gaussian elimination to produce CNOT gates, so substituting the STEINER-GAUSS algorithm immediately gives a circuit extraction procedure that respects nearest-neighbour constraints of the architecture.

Since the technique described in [12], and the corresponding PyZX circuit optimisation tool [14], take an arbitrary quantum circuit as input, this will give a general-purpose, optimising routine for circuit mapping. We leave a detailed exposition (and implementation) of this technique for future work.

5 Conclusions and Future Work

We have demonstrated a CNOT circuit mapping procedure that significantly out-performs existing compilers on memory architectures whose graphs contain a Hamiltonian path. We have also outlined extensions of this technique to arbitrary graphs and to arbitrary circuits. In addition to implementing the extensions outlined in the previous section, an interesting direction for future work is to focus not only on decreasing two-qubit gate count, but also in decreasing gate depth and/or overall fidelity loss due to gate errors. With the incredibly short coherence times characteristic of superconducting hardware, gate depth is likely to play an even more important role than gate count in practical realisation of quantum computations in the coming years [10]. It also comes with a unique set of challenges, as parallel gates can interfere with each other if they are mapped to neighbouring locations on some architectures [33]. Another notable feature of current hardware is not all qubits are created equal: performing 2-qubit gates between certain pairs of qubits can be done with much higher fidelities than others, depending on implementation details or even random variance in the manufacturing processes of superconducting chips [34]. While the optimisation techniques used in this paper are very simple, a topic of future work is to apply much more powerful machine learning and/or constraint satisfaction techniques in order to take these factors into account.

Acknowledgements.

We gratefully acknowledge support from the Unitary Fund (`http://unitary.fund`) for this work. We would also like to thank Will Zeng, Ross Duncan, and John van de Wetering for fruitful discussions about circuit mapping for NISQ as well as the authors of [22] for clarifying some points about their approach.

References

1. Matthew Reagor, Christopher B Osborn, Nikolas Tezak, Alexa Staley, Guenevere Prawiroatmodjo, Michael Scheer, Nasser Alidoust, Eyob A Sete, Nicolas Didier, Marcus P da Silva, et al. Demonstration of universal parametric entangling gates on a multi-qubit lattice. *Science advances*, 4(2):eaao3603, 2018.
2. IBM. IBM unveils world's first integrated quantum computing system for commercial use. https://newsroom.ibm.com/2019-01-08-IBM-Unveils-Worlds-First-Integrated-Quantum-Computing-System-for-Commercial-Use, 2019. Accessed: 2019-03-28.
3. R Versluis, S Poletto, N Khammassi, N Haider, DJ Michalak, A Bruno, K Bertels, and L DiCarlo. Scalable quantum circuit and control for a superconducting surface code. *arXiv:1612.08208*, 2016.
4. Joseph W Britton, Brian C Sawyer, Adam C Keith, C-C Joseph Wang, James K Freericks, Hermann Uys, Michael J Biercuk, and John J Bollinger. Engineered two-dimensional ising interactions in a trapped-ion quantum simulator with hundreds of spins. *Nature*, 484(7395):489, 2012.
5. CJ Ballance, TP Harty, NM Linke, MA Sepiol, and DM Lucas. High-fidelity quantum logic gates using trapped-ion hyperfine qubits. *Physical Review Letters*, 117(6):060504, 2016.
6. JP Gaebler, TR Tan, Y Lin, Y Wan, R Bowler, AC Keith, S Glancy, K Coakley, E Knill, D Leibfried, et al. High-fidelity universal gate set for be 9+ ion qubits. *Physical Review Letters*, 117(6):060505, 2016.

7. WK Hensinger, S Olmschenk, D Stick, D Hucul, M Yeo, M Acton, L Deslauriers, C Monroe, and J Rabchuk. T-junction ion trap array for two-dimensional ion shuttling, storage, and manipulation. *Applied Physics Letters*, 88(3):034101, 2006.
8. Alwin Zulehner, Alexandru Paler, and Robert Wille. An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.
9. Steven Herbert and Akash Sengupta. Using reinforcement learning to find efficient qubit routing policies for deployment in near-term quantum computers. *arXiv:1812.11619*, 2018.
10. Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. On the qubit routing problem. *arXiv:1902.08091*, 2019.
11. Matthew Amy, Dmitri Maslov, and Michele Mosca. Polynomial-time T-depth optimization of Clifford+ T circuits via matroid partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1476–1489, 2014.
12. Ross Duncan, Alex Kissinger, Simon Perdrix, and John van de Wetering. Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus. *https://arxiv.org/abs/1902.03178*, 2019.
13. B. Coecke and R. Duncan. Interacting quantum observables. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science, 2008.
14. Aleks Kissinger and John van de Wetering. PyZX: A circuit optimisation tool based on the ZX-caculus. http://github.com/Quantomatic/pyzx.
15. Aleks Kissinger and John van de Wetering. Reducing t-count with the zx-calculus. *arXiv:1903.10477*, 2019.
16. Prakash Murali, Ali Javadi-Abhari, Frederic T Chong, and Margaret Martonosi. Formal constraint-based compilation for noisy intermediate-scale quantum systems. *Microprocessors and Microsystems*, 2019.
17. Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank. Compiling quantum circuits to realistic hardware architectures using temporal planners. *Quantum Science and Technology*, 3(2):025004, 2018.
18. Alexandru Paler, Alwin Zulehner, and Robert Wille. NISQ circuit compilers: search space structure and heuristics. *arXiv:1806.07241*, 2018.
19. Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for NISQ-era quantum devices. *arXiv:1809.02573*, 2018.
20. Davide Ferrari and Michele Amoretti. Demonstration of envariance and parity learning on the IBM 16 qubit processor. *arXiv:1801.02363*, 2018.
21. Alexandru Paler. On the influence of initial qubit placement during NISQ circuit compilation. *arXiv:1811.08985*, 2018.
22. Beatrice Nash, Vlad Gheorghiu, and Michele Mosca. Quantum circuit optimizations for NISQ architectures. *arXiv preprint arXiv:1904.01972*, 2019.
23. Aleks Kissinger and Arianne Meijer-van de Griend. CNOT circuit extraction for topologically-constrained quantum memories. *arXiv preprint arXiv:1904.00633*, 2019.
24. Ketan N. Patel, Igor L. Markov, and John P. Hayes. Optimal synthesis of linear reversible circuits. *Quantum Info. Comput.*, 8(3):282–294, March 2008.
25. Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
26. Gabriel Robins and Alexander Zelikovsky. Improved steiner tree approximation in graphs. In *SODA*, pages 770–779. Citeseer, 2000.
27. Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An Improved LP-based Approximation for Steiner Tree. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 583–592, New York, NY, USA, 2010. ACM.
28. Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.
29. David E Golberg. Genetic algorithms in search, optimization, and machine learning. *Addion*

wesley, 1989(102):36, 1989.
30. Robert S Smith, Michael J Curtis, and William J Zeng. A practical quantum instruction set architecture. *arXiv:1608.03355*, 2016.
31. Christopher M Dawson, Andrew P Hines, Duncan Mortimer, Henry L Haselgrove, Michael A Nielsen, and Tobias J Osborne. Quantum computing and polynomial equations over the finite field F_2. *Quantum Information & Computation*, 5(2):102–112, 2005.
32. Matthew Amy, Parsiad Azimzadeh, and Michele Mosca. On the controlled-not complexity of controlled-not–phase circuits. *Quantum Science and Technology*, 4(1):015002, 2018.
33. An Update on Google's Quantum Computing Initiative. Slides for presantation at FOSDEM 2019. https://fosdem.org/2019/schedule/event/google_qc/attachments/slides/3082/export/events/attachments/google_qc/slides/3082/GoogleQuantumCirq.. Accessed 1 May, 2019.
34. Performance parameters for Rigetti QPUs. https://www.rigetti.com/qpu. Accessed 1 May, 2019.