Quantum Information and Computation, Vol. 24, No. 9&10 (2024) 721–733


# LINEAR CIRCUIT SYNTHESIS USING WEIGHTED STEINER TREES

NIR GAVRIELOV
*Racah Institute of Physics, The Hebrew University of Jerusalem, Givat Ram Jerusalem 91904, Israel*

ALEXANDER IVRII, SHELLY GARION
*IBM Quantum, IBM Research Israel, Haifa University Campus, Mount Carmel Haifa 3498825, Israel*

Received May 18, 2024
Revised August 6, 2024

CNOT circuits are a common building block of general quantum circuits. The problem of synthesizing and optimizing such circuits has received a lot of attention in the quantum computing literature. This problem is especially challenging for quantum devices with restricted connectivity, where two-qubit gates can only be placed between adjacent qubits. The state-of-the-art algorithms for optimizing the number of CNOT gates are heuristic algorithms that are based on Gaussian elimination and that use Steiner trees to connect between different subsets of qubits. In this article, we suggest considering *weighted* Steiner trees, and we present a simple low-cost heuristic to compute weights. The simulated evaluation shows that the suggested heuristic is almost always beneficial and reduces the number of CNOT gates by up to 10%.

*Keywords*: linear circuit synthesis, Steiner trees, quantum circuit compilation, quantum information

## 1 Introduction

Quantum circuit synthesis is a process of constructing a quantum circuit that implements a given unitary operator and can be executed on a given quantum device while minimizing the number of gates or the depth of the circuit. Quantum devices differ in terms of *supported gate set* and *connectivity*. For instance, superconducting quantum devices support single-qubit rotations and the two-qubit CNOT gates; moreover, the CNOT gates can only be placed between the "connected" qubits. Hence, placing an entangling two-qubit gate on non-connected qubits requires finding an optimal route between these two qubits and placing SWAP gates between all pairs of adjunct qubits in this route, where each SWAP gate can be implemented using three CNOT gates (1). As the two-qubit gates are significantly noisier than single-qubit gates, the goal of quantum circuit synthesis is usually to optimize the number of CNOT gates or the CNOT depth of the constructed circuit.

CNOT circuits (that is, circuits that only consist of CNOT gates) appear as common building blocks of general quantum circuits. For instance, they are common subcircuits produced by variational-quantum-eigensolver (VQE) algorithms used for quantum chemistry, quantum simulation and quantum optimization (see, e.g. (2)). CNOT circuits also appear when studying Clifford operators, (3) shows that any Clifford operator can be implemented in the form

$-CX-CZ-P-H-P-CZ-CX-$, which includes two $CX$-layers (that is, two CNOT subcircuits). Thus, it should not be surprising that the problem of synthesizing and optimizing CNOT circuits has received a lot of attention in the quantum computing literature.

The problem of synthesizing a CNOT-circuit of width n can be viewed as the problem of reducing a matrix in $GL_2(n)$ to the identity matrix using Gaussian elimination (see section 2 for details), with the additional restriction that row operations can only be performed between rows corresponding to connected qubits. Two algorithms for this task are described in (4–6), which are both heuristic algorithms that aim to minimize the number of CNOT gates. The `SteinerGauss` algorithm (4; 5) is based on reducing first a matrix in $GL_2(n)$ to a lower-triangular matrix and then to the identity matrix. The `RowCol` algorithm (6) is based on eliminating one qubit at a time. Crucially, both works rely on the computation of Steiner trees between various sets of qubits to optimize the number of row operations for each basic step of the algorithm (e.g., eliminating all non-diagonal 1s in the given column of a matrix). In all of these works, Steiner tree optimization aims to minimize the number of edges needed to connect a given subset of nodes. As the problem is NP-hard, in practice approximate Steiner tree computation algorithms are used.

In general, at each step of the algorithm, multiple different optimal Steiner trees can be found, with each tree leading to a different sequence of row operations and hence to a different matrix obtained by executing these row operations. This work aims to improve the above algorithms based on the intuition that it is beneficial to choose the tree that brings the matrix "closer" to the target, the identity matrix. To this extent, we suggest assigning weights to the edges of the connectivity graph and present a heuristic for doing so. This heuristic has a low computational cost and can be easily used with any standard approximate Steiner tree computation tool. Similar intuition and a related heuristic were described in (7) in the context of phase polynomial synthesis. The experimental evaluation shows that the suggested heuristic is beneficial and reduces the number of CNOT gates.

In addition, we improve the CNOT circuit depth compared to the `RowCol` and `SteinerGauss` algorithms mentioned above. We note that for certain connectivity maps between the device qubits, there are better known estimations. For all-to-all connectivity of the $n$ qubits, where every pair of qubits is connected, the depth of the CNOT circuit is bounded by $n+O(\log^2(n))$ (8; 9). For linear nearest neighbor connectivity of the $n$ qubits the depth is bounded by $5n$ (10). In (6) it was shown that for a two-dimensional grid connectivity of the $n$ qubits, the depth is $O(n)$ with $n^2$ ancillas. Recently, (11) extended the approach of (10) to handle blocks of qubits arranged in a line, and in particular proved that the depth of the CNOT circuit on the $n$-qubit grid is bounded by $4n$ (without using ancilla qubits). In some of the current quantum devices, the connectivity between the qubits is a two-dimensional heavy hexagon, into which one can embed a line or a sequence of blocks arranged in a line that includes all or almost all of the qubits, obtaining an effective bound on the CNOT circuit depth. However, such a bound is not known for all possible quantum device connectivity maps, and our results provide a heuristic algorithm that improves the depth in general.

This paper is organized as follows. In Section 2, we describe the CNOT circuit synthesis problem and the `SteinerGauss` and `RowCol` algorithms. In Section 3, we describe the weight assignment heuristic and illustrate it with an example. Section 4 contains the experimental evaluation. Section 5 concludes the paper.

## 2 Preliminaries

**Linear circuits** Quantum circuits consisting only of CNOT gates are known as *linear circuits* or *linear functions* (12). The CNOT gate performs a controlled-not operation on two qubits: $CNOT(c,t) = (c, t \oplus c)$, with $c$ called the *control* qubit and $t$ called the *target* qubit. Hence, a CNOT gate between two qubits corresponds to a linear reversible function $f : \mathbb{F}_2^2 \to \mathbb{F}_2^2$, which is a $2 \times 2$ binary invertible matrix. The CNOT gate is also universal for linear reversible circuits, therefore any linear reversible function, i.e., any $n \times n$ matrix in the $GL_2(n)$ group, can be implemented using only CNOT gates. A single CNOT circuit gate, controlled by wire $i$ and acting on wire $j \neq i$, can be represented by the elementary matrix $E_{ij}$ (equal to the identity matrix with component $(i,j)$ flipped to 1). Composing a succession of these operations one obtains a $n \times n$ *parity matrix*, representing a n-qubit linear circuit. Row $i$ of the matrix contains the parity output of qubit $i$, sum of those qubits with index $j$ such that the entry $(i,j)$ of the matrix equals 1. Thus, given a random reversible operator $M$ we look for a sequence of $m$ elementary matrices that satisfy

$$\left(\prod_{k=1}^{m} E_{i_k,j_k}\right) M = I_n \Leftrightarrow \prod_{k=m}^{1} E_{i_k,j_k} = M \tag{1}$$

using $E_{ij}^{-1} = E_{ij}$. A circuit containing CNOT operations corresponding to the sequence of elementary matrices $(E_{i_k,j_k})_{k=m}^{1}$ implements the function $M$. For example, the linear circuit in figure 1 can be represented by

$$\overset{G_5}{\begin{pmatrix}1&1&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}}\overset{G_4}{\begin{pmatrix}1&0&0&0\\0&1&1&0\\0&0&1&0\\0&0&0&1\end{pmatrix}}\overset{G_3}{\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&1\\0&0&0&1\end{pmatrix}}\overset{G_2}{\begin{pmatrix}1&0&0&0\\1&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}}\overset{G_1}{\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&1&0&1\end{pmatrix}} = \overset{M}{\begin{pmatrix}0&0&1&1\\1&0&1&1\\0&1&1&1\\0&1&0&1\end{pmatrix}} \tag{2}$$

Since two-qubit gates are prone to hardware errors, a synthesis algorithm will aim to reduce the total amount of operations, known as the circuit size. A second key metric is the resulting circuit depth, corresponding to the number of timestamps required to execute the circuit, assuming that independent gates are performed simultaneously.

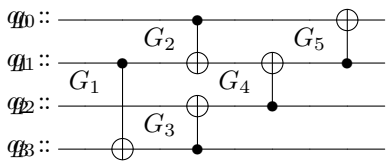



Fig. 1. A linear circuit with depth 4 and 5 CNOT gates, corresponding to the 5 elementary operations in Eq. (2).

**Steiner Trees** A coupling map of a quantum processor is usually represented using an undirected graph $G = (V, E)$ where $V$ is a set of vertices (qubits) and the edges E denote the coupling between adjacent qubits. Such graph has at most one edge connecting each pair of

nodes (not a "multi-graph") and no self-loops, i.e. edges with the same node on both ends. The edges of a graph can be given a numerical weight by some rule, making it a weighted graph. A tree is an undirected graph in which any two vertices are connected by a unique path - a connected graph with no loops. A spanning tree of a connected graph is a tree subgraph that includes all the vertices of $G$. Each graph may contain many spanning trees and those with minimal overall edge weight are called minimal spanning trees (MST). Given a graph $G = (V, E)$ and a set of vertices $S$, a *Steiner Tree* $T = (V_T, E_T)$ is a minimal weight tree subgraph that contains all the vertices in $S$. The nodes in $S$ are often called *terminals* and those in $V_T \setminus S$ are known as *Steiner nodes*. The problem of finding an optimal Steiner tree in an arbitrary graph is known to be NP-hard (13). Our work uses the rustworkx graph package (14), therefore to find a Steiner tree we used their method based on the $(2 - 2/|S|)$ approximate algorithm from (15). A special kind of Steiner tree used is a *decreasing Steiner tree*, in which each node is larger than its children according to a certain input ordering of the vertices.

**Steiner tree based synthesis of linear circuits** We briefly describe the state-of-the-art algorithms `SteinerGauss` and `RowCol` for the synthesis of linear circuits in quantum devices with restricted connectivity. The algorithms start with a linear invertible binary matrix and aim to optimize the number of row operations (adhering to the connectivity of the device) that reduce the matrix to the identity or to a permutation matrix.

The `SteinerGauss` (4; 5) algorithm first reduces the original matrix to an upper triangular matrix. The matrix $M$ is brought into the upper triangular form column after column. When processing a column $i$, one considers the set $S = \{i\} \cup \{k|\ k > i \text{ and } M_{k,i} = 1\}$ corresponding to the diagonal entry of $M_{i,i}$ together with the column's 1-entries below the diagonal. These entries are then connected using an (approximated) Steiner tree, which is then used to perform a sequence of row operations that result in $M_{i,i} = 1$ and $M_{k,i} = 0$ for all $k > i$. In the second step, the matrix $M$ is reduced to the identity matrix. This is achieved by transposing the matrix (to become lower-triangular) and applying the algorithm from the previous step with one important change required to preserve the lower-triangular form: only *decreasing* Steiner trees can be used.

The `RowCol` algorithm (6) follows a somewhat different matrix simplification strategy. This algorithm is presented as Algorithm 2 (without the gray highlighted text). `RowCol` processes one node of $V$ after another, each time fully simplifying both the relevant column and the relevant row of $M$. Given a node $i \in V$, in lines 3–6 of the algorithm the diagonal element $M_{i,i}$ is turned to 1 and all non-diagonal elements in column $i$ are turned to 0. This is similar to the first step of `SteinerGauss`, except that all the 1 entries in column $i$ are considered. In lines 8–11 all the non-diagonal elements in row $i$ are turned to 0. To do so, the algorithm finds a linear combination of rows that is equal to the target row plus the relevant unit vector. These vertices are added to the target row, using Steiner trees to guide how the rows are added. At this point $M_{i,i} = 1$ and $M_{i,k} = M_{k,i} = 0$ for all $k \neq i$. On line 12 the vertex $i$ is removed from the graph, and the process continues until the graph becomes empty.

**Input :** Integer n, matrix $\mathbf{M} \in \mathbb{F}_2^{n \times n}$, graph $(V, E)$ where $|V| = n$, weight function $w : \mathbb{F}_2^{n \times n} \times E \to \mathbb{R}^{\geq 0}$

**Output:** Row additions to transform $\mathbf{M}$ into $\mathbf{I}$

**for** $i \in V$ *which is not a cut vertex* **do**
  `UpdateEdgeWeight` $(\mathbf{M}, G, w)$
  $S = \{j | \mathbf{M}_{ji} \neq 0\} \cup \{i\}$
  Find a tree T containing $S \subseteq V$ in G
  Postorder traverse T from i. When reaching j with parent k, add row j to row k if $\mathbf{M}_{ji} = 1$ and $\mathbf{M}_{ki} = 0$
  Postorder traverse T from i, add every row to its children when reached
  `UpdateEdgeWeight` $(\mathbf{M}, G, w)$
  Let $S' \subseteq V$ that $\sum_{j \in S'} \mathbf{M}_j = \mathbf{M}_i + e_i$
  Find a tree $T'$ containing $S' \cup \{i\}$
  Preorder traverse $T'$ from i. When reaching $j \notin S'$, add the j-th row to its parent
  Postorder traverse $T'$ from i, add every row to its parent when reached
  Delete i from graph G
**end**

Fig. 2. Weighted RowCol, weighted edges optimization extension of the `RowCol` algorithm. Similar adjustment can be used in `SteinerGauss` or any other Steiner tree based algorithm.

## 3 Method

We propose a heuristic method for weighting the edges of the coupling graph during a synthesis algorithm, which leads to a reduced total number of CNOT gates in the transpiled circuit. In the linear function synthesis task, one aims to transform a general binary matrix to the identity using row operations. Since the atomic operator in the process is the addition of two binary rows of the matrix, we look for a function that given two such vectors assigns a scalar weight. A successful choice of a function of this kind will result in edges representing the number of CNOT gates added upon their choice, and the optimal Steiner tree will include edges that minimize the total number of gates added in the current step of the algorithm. However, such a function is hard to find, as there is no exact quantitative measure connecting the linear function matrix to the number of gates in the final circuit. Furthermore, after obtaining an optimal tree, the mentioned algorithms apply many additions in both directions, that is, parent-to-child and vice versa, actions that have to be taken into account when assigning a weight. Therefore, we turn to heuristic methods. We use the *Hamming distance*, standard measure of difference between binary arrays, defined for $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$ as $h(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} x_i \neq y_i$, with the distance from the zero array called the *Hamming weight*. Intuitively, as our target $\mathbf{I}_n$ is a sparse matrix, we will prefer operations that minimize the Hamming weight of row operations. Even though the addition manifests itself as the binary XOR operation, we explore a wider choice of operations for the weight function and pick the best one as described below. Other operations might yield better results, since after tree selection a complex sequence of row additions is performed. We treat all the qubits symmetrically, therefore every entry in the vector will be calculated by the same rule, limiting the choice to all 2-bit operations. The usage of undirected graphs adds another constraint, 01 and 10 inputs must have the same output.

A quick count shows that there are only $2^3 = 8$ candidates, including the irrelevant constant 0 function (ZERO) and the constant 1 function (ONE), which is equivalent to the unweighted case. More complex heuristic rules, such as bidirectional weights and parameterized scaling, did not affect the overall synthesis.

Eventually, given a parity matrix $M \in GL_2(n)$ and an edge $e = (u, v)$, the weight is given by

$$w_f(M, e = (u, v)) = h(f(M_u, M_v)) \tag{3}$$

where $f$ is one of the functions listed in table 1. Therefore, when we come to use any linear synthesis algorithm, we have several options for the weighting heuristic. In order to compare the varying rules, we "pre-train" to find the ideal one, using a cost function estimating the average CNOT count of an algorithm utilizing a specific rule. Given a set of graphs $\mathcal{G}$ and a weight heuristic $w_f$ calculate

$$cost(f) = \sum_{G \in \mathcal{G}} \frac{1}{|G|^2} \langle CNOT\ count(G, M, w_f) \rangle_{M \in \mathcal{P}_{samp}(G)} \tag{4}$$

where $\mathcal{P}_{samp}(G)$ is a set of random linear functions to synthesize on the graph G and $|G|$ is its cardinality. The function $CNOT\ count(G, M, w_f)$ returns the number of gates in the synthesized circuit and $\langle ... \rangle$ is used to average the count over all input matrices in $\mathcal{P}_{samp}(G)$. The idea is to find the best rule for different architectures, circuit widths, and linear functions. Normalization $|G|^{-2}$ is used to cancel the cost quadratic scaling in $|G|$, to provide a similar contribution from all graph sizes.

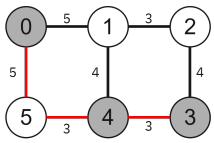
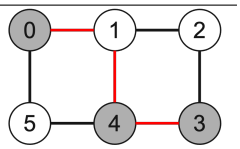
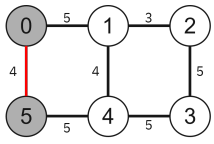
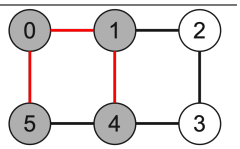
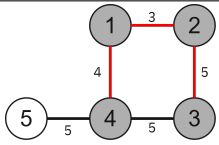
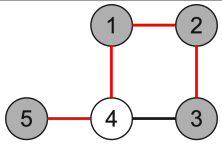
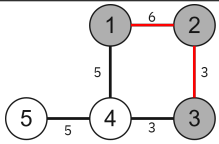
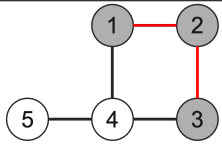
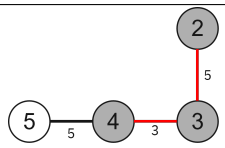
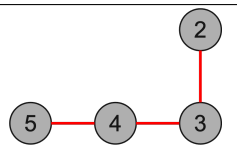
Table 1. Two-bit operations used for the weight function and their corresponding cost estimation. 3 possible different inputs yielding a total of $2^3 = 8$ possible binary functions. The ZERO function assigns zero weight to all edges and is thus irrelevant for our purposes, and the ONE function is equivalent to an unweighted algorithm as all edges will hold the same weight. The cost function from Eq. (4) was evaluated by averaging synthesis over 100 random matrices for each of 6 graphs - grid and all-to-all architectures (low and high connectivity) with 9, 49, and 81 qubits. The uncertainty in the cost evaluation is 0.1 gates per qubit count squared for all entries.

| function / input | ZERO | AND | XOR | OR | NOR | NXOR | NAND | ONE |
|---|---|---|---|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 01/10 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 11 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| `RowCol` cost | - | 4.1 | 3.6 | 3.6 | 3.8 | 3.8 | 3.5 | 3.7 |
| `SteinerGauss` cost | - | 4.1 | 3.7 | 3.7 | 4.0 | 4.0 | 3.8 | 3.8 |

An illustration of the suggested heuristic is depicted in table 2, in which we compare the steps of the augmented `RowCol` algorithm with the NAND heuristic to the steps of the original `RowCol` algorithm for a certain example. A similar weighting method has been proposed in (7) for phase polynomial synthesis, but differs from our work both in the problem it aims to solve and in the specific edge weighting. In their work, for each parity, they create a complete directed graph with the relevant qubits, then weight the edges according to $e = (u, v), M \in \mathbb{F}_2^{n \times m} \rightarrow w(e, M) = h(M_u \oplus M_v) - h(M_v)$, and find a minimal spanning tree.

In order to update the weights, after each iteration, one should traverse all edges of the coupling map and perform an operation with a time complexity of $\mathcal{O}(|V|)$. Consequently,

Table 2. Illustration of the method. Operation of the RowCol algorithm on the same linear function and coupling map, with (left) and without (right) the heuristic. In each step of the process, the state of the parity matrix and the graph is presented, the terminal nodes are in gray and the chosen Steiner tree edges are colored in red. After eliminating the first 2 columns and rows, it is seen that the weighted algorithm has made larger progress, in terms of hamming distance from $\mathbb{I}_6$, and used smaller trees. The weighted algorithm synthesis ended with a circuit containing 18 CNOTs, 9 less than the unweighted one. For brevity, the remaining steps are shown in the appendix.

| Step | Weighted | | Unweighted | |
|---|---|---|---|---|
| Col 0 | $\begin{pmatrix}1&0&0&1&1&0\\0&0&1&0&1&1\\0&1&1&1&1&1\\1&0&0&0&1&1\\1&1&0&0&1&1\\0&1&0&0&1&1\end{pmatrix}$ |  | $\begin{pmatrix}1&0&0&1&1&0\\0&0&1&0&1&1\\0&1&1&1&1&1\\1&0&0&0&1&1\\1&1&0&0&1&1\\0&1&0&0&1&1\end{pmatrix}$ |  |
| Row 0 | $\begin{pmatrix}1&0&0&1&1&0\\0&0&1&0&1&1\\0&1&1&1&1&1\\0&1&0&0&0&0\\0&1&0&0&1&1\\0&0&0&1&1&0\end{pmatrix}$ |  | $\begin{pmatrix}1&0&0&1&1&0\\0&1&1&1&1&0\\0&1&1&1&1&1\\0&1&0&0&0&0\\0&0&1&0&1&1\\0&1&0&0&1&1\end{pmatrix}$ |  |
| Col 1 | $\begin{pmatrix}1&0&0&0&0&0\\0&0&1&0&1&1\\0&1&1&1&1&1\\0&1&0&0&0&0\\0&1&0&0&1&1\\0&0&0&1&1&0\end{pmatrix}$ |  | $\begin{pmatrix}1&0&0&0&0&0\\0&1&0&1&0&1\\0&1&1&1&1&1\\0&1&0&0&0&0\\0&0&1&0&1&1\\0&1&0&0&1&1\end{pmatrix}$ |  |
| Row 1 | $\begin{pmatrix}1&0&0&0&0&0\\0&1&1&0&0&0\\0&0&0&1&1&1\\0&0&1&1&1&1\\0&0&1&0&1&1\\0&0&0&1&1&0\end{pmatrix}$ |  | $\begin{pmatrix}1&0&0&0&0&0\\0&1&1&0&0&0\\0&0&0&1&1&1\\0&0&1&1&1&1\\0&0&1&0&1&1\\0&0&0&1&1&0\end{pmatrix}$ |  |
| Col 2 | $\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&1&1&1&1\\0&0&1&0&1&1\\0&0&0&1&1&0\end{pmatrix}$ |  | $\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&0&1&0&1\\0&0&1&1&1&1\\0&0&1&1&0&1\\0&0&1&0&1&1\end{pmatrix}$ |  |

the resulting complexity overhead of the heuristic is a polynomial addition of $(|V| \cdot |E|)$ for a coupling graph with $|V|$ nodes and $|E|$ edges. To continue with the `RowCol` example, for an n-qubit circuit, the algorithm performs n iterations, the computationally heaviest operation in each being the Steiner tree search. Given that the approximate tree search complexity is $\mathcal{O}\left(|S|\left(|V|\log|V|+|E|\right)\right)$(15), the enhanced program asymptotic runtime is

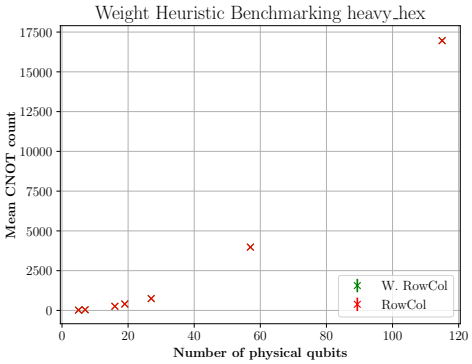
$$\mathcal{O}\left[\underbrace{|V|}_{\#iterations}\cdot\left(\underbrace{|V|\cdot|E|}_{wt.track}+\underbrace{|S|\left(|V|\log|V|+|E|\right)}_{SteinerTree}\right)\right]=\mathcal{O}\left(|V|^4\right) \tag{5}$$

which is equal to the complexity of the original `RowCol` algorithm.

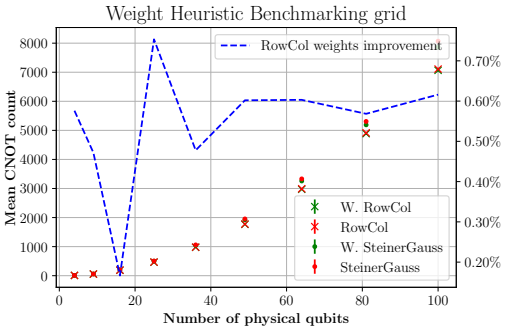
## 4 Results

We now turn to present the performance results of the method for the two different algorithms `RowCol` and `SteinerGauss`. As described in the Method section, we first looked for the optimal weight rule by minimizing the cost function in (4). In this "pre-training" process, we found the NAND rule optimal for `RowCol` with an average cost of 3.5 gates per qubit number squared, and for the `SteinerGauss` algorithm the OR heuristic yielded the optimal result, as can be seen in the bottom part of table 1. Intuitively, since in the `RowCol` algorithm, one performs row eliminations by linear combinations of other rows, reducing the number of dependent rows will lead to better results. Therefore, the NAND operation, which favors cancellation of the overlapping entries 1, was found to perform best. In figure 4 we depict the average CNOT count of the algorithms, for various device connectivity architectures and sizes, showing a substantial improvement for the weighted algorithms.

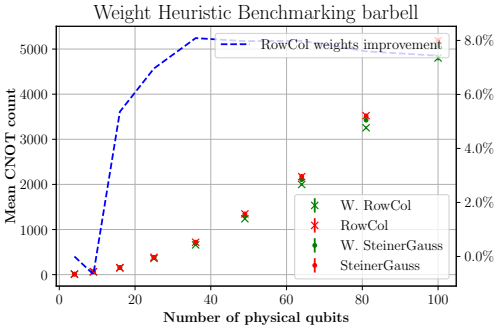
Since the method requires enough Steiner tree possibilities within the graph to make a difference, the more connected the device, the larger the circuit size reduction. Our results show a negligible change for the heavy-hex architecture which is within the error bounds, a slight 1% boost given a grid of qubits, and up to more than 10% smaller circuits on complete graphs. To quantify this claim, in figure 5a we present the synthesis performance as a function of the connectivity of the graph. On a (fixed) 25-qubit device initially containing only chain nearest neighbor edges, we compare the synthesis of the same matrices as we add randomly selected connecting edges. Clearly, the weighted algorithm uses the higher connectivity more intelligently. The improvement is even greater when we look at the depth metric within the simulation mentioned, as seen in figure 5b. We explain this gap by the fact that during the unweighted algorithm the lowest (by index) Steiner nodes are chosen repeatedly, while the weight analysis forces a more diverse choice, which opens the possibility for parallel gates and lower overall synthesis depth. Naturally, the performance of the algorithms depends on the input linear circuit properties as well. We have trialed random input circuits with an increasing number of CNOTs and analyzed the properties of the re-synthesized circuits. From figure 7a (7b) we infer that the more CNOTs in the input circuits, the larger the reduction in the size (depth) of the output circuits. This gain reaches saturation since the $GL_2(n)$ group contains a finite number of unique operations, many of the two-qubit operations in input circuits of large size cancel each other out, and the average synthesis cost approaches a constant value.



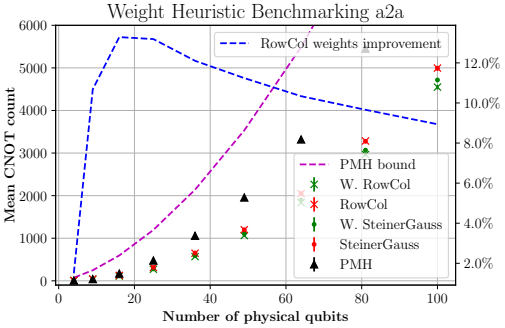

(a) heavy-hex architecture. A decreasing Steiner tree is not always feasible in these graphs, therefore `SteinerGauss` is not evaluated. In this case, the weighted and unweighted points are indistinguishable.




(b) 2D grid




(c) barbell, a graph containing 2 complete subgraphs connected by a path of 1 or 2 edges

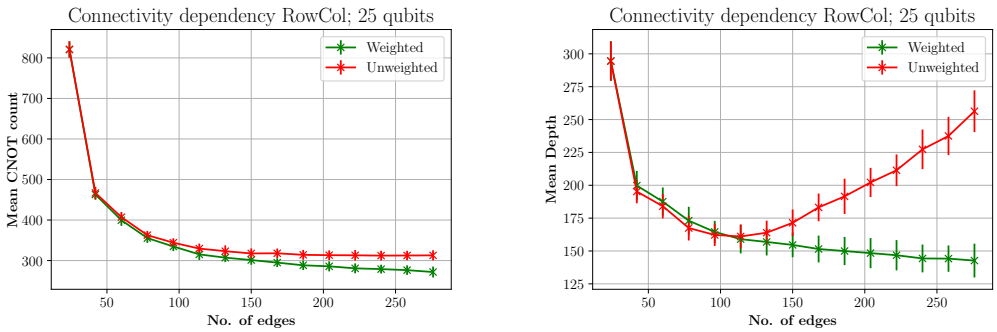



(d) All-to-All connectivity, with Patel-Markov-Hayes(PMH)(12) results as comparison. PMH result for 100 qubits is 8175 ± 27, and has been omitted for clarity

Fig. 4. Benchmarking. Each marker represents an average result over one hundred reversible matrices of the same size, standard deviations are around tens of gates thus not visible. Markers are grouped by algorithm (symbol) and weights (color). The weighted `RowCol` algorithm uses the NAND heuristic, while the `SteinerGauss` results shown are with the OR rule. The blue dashed line together with the right axis represents the weighted version improvement, calculated by $1 - \left\langle \frac{CNOT\text{count}(G,M,w_{\text{NAND}})}{CNOT\text{count}(G,M,w_{\text{const}})} \right\rangle$.

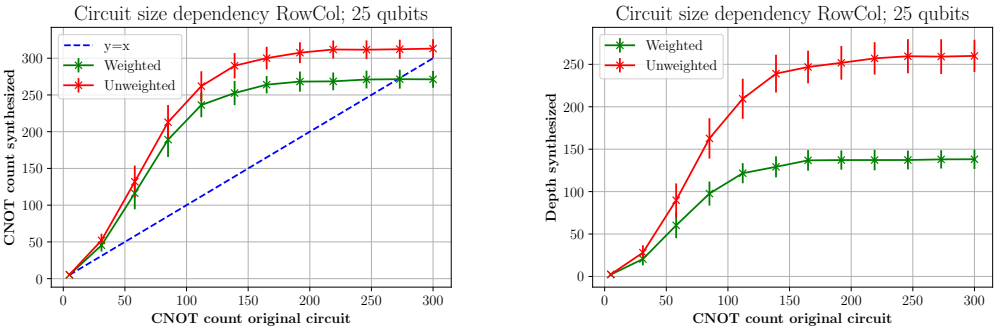
## 5 Conclusion

In this paper, we present a general heuristic to improve Steiner tree algorithms. The proposed approach has been tested on the `RowCol` and `SteinerGauss` algorithms, improving their performance on varying architectures and linear functions. Our benchmarking showed that the heuristic is most useful in cases where the graphs are highly connected or where large circuits need to be synthesized. In addition, the complexity overhead of the heuristic is minimal, making it suitable for real-world applications. This work lays the foundation for future algorithms that can be built upon the proposed heuristic, adjusting the assignment of




(a) size (b) depth

Fig. 6. synthesis performance as a function of the coupling map connectivity




(a) size (b) depth

Fig. 8. input circuit CNOT count dependency. In all of these simulations, the coupling map is fixed to be a complete graph with 25 qubits. Each point represents the average result over 100 random parity matrices with the same amount of row operations (sometimes annulling) on the identity.

the weights to new optimization tasks and methods in the field.

## Acknowledgements

The authors thank Lev Bishop, Eli Arbel, Simon Martiel, Ali Javadi-Abhari, and Dmitri Maslov from IBM Quantum for fruitful discussions and support. We also thank Arianne Meijer - van de Griend for the code and explanation of the `SteinerGauss` algorithm.

## References

[1] Gushu Li, Yufei Ding, and Yuan Xie, Tackling the qubit mapping problem for nisq-era quantum devices, *ASPLOS '19*, page 10011014, New York, NY, USA, 2019. Association for Computing Machinery.

[2] Sukin Sim, Peter D. Johnson, and Aln Aspuru-Guzik, Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms, *Advanced Quantum Technologies*, 2(12):1900070, 2019.

[3] Dmitri Maslov and Martin Roetteler Shorter stabilizer circuits via bruhat decomposition and quantum circuit transformations, CoRR, abs/1705.09176, 2017.

[4] Beatrice Nash, Vlad Gheorghiu, and Michele Mosca. Quantum circuit optimizations for nisq architectures Quantum Science and Technology, 5(2):025010, mar 2020.

[5] Aleks Kissinger and Arianne Meijer, Cnot circuit extraction for topologically-constrained quantum memories, *Quantum Information and Computation*, 20:581–596, 06 2020.

[6] Bujiao Wu, Xiaoyu He, Shuai Yang, Lifu Shou, Guojing Tian, Jialin Zhang, and Xiaoming Sun, Optimization of cnot circuits on limited-connectivity architecture, *Phys. Rev. Res.*, 5:013065, Jan 2023.

[7] Vivien Vandaele, Simon Martiel, and Timothée Goubault de Brugière, Phase polynomials synthesis algorithms for nisq architectures and beyond, *Quantum Science and Technology*, 7(4):045027, sep 2022.

[8] Dmitri Maslov and Ben Zindorf, Depth optimization of cz, cnot, and clifford circuits, *IEEE Transactions on Quantum Engineering*, 3:1–8, 2022.

[9] Timothée Goubault de Brugière, Marc Baboulin, Benot Valiron, Simon Martiel, and Cyril Allouche, Reducing the depth of linear reversible quantum circuits, *IEEE Transactions on Quantum Engineering*, 2:1–22, 2021.

[10] Samuel A. Kutin, David Petrie Moulton, and Lawren Smithline, Computation at a distance, *Chic. J. Theor. Comput. Sci.*, 2007.

[11] Timothée Goubault de Brugière, and Simon Martiel, Shallower CNOT circuits on realistic quantum hardware, arXi2303.07302, 2023.

[12] Ketan N. Patel, Igor L. Markov, and John P. Hayes, Optimal synthesis of linear reversible circuits, *Quantum Info. Comput.*, 8(3):282294, mar 2008.

[13] Richard M. Karp, *Reducibility among Combinatorial Problems*, pages 85–103, Springer US, Boston, MA, 1972.

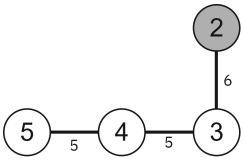
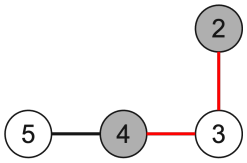
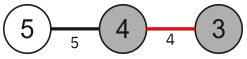
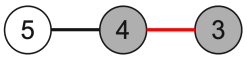
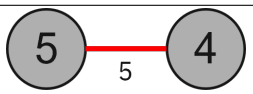
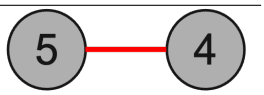
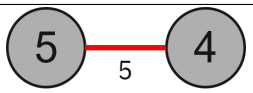
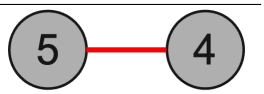
[14] Matthew Treinish, Ivan Carvalho, Georgios Tsilimigkounakis, and Nahum Sá, rustworkx: A high-performance graph library for python. *Journal of Open Source Software*, 7(79):3968, nov 2022.

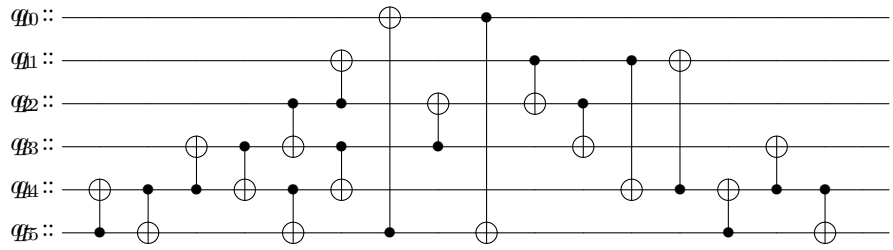
[15] Kurt Mehlhorn, A faster approximation algorithm for the steiner problem in graphs, *Information Processing Letters*, 27(3):125–128, 1988.

## Appendix A End of Illustration

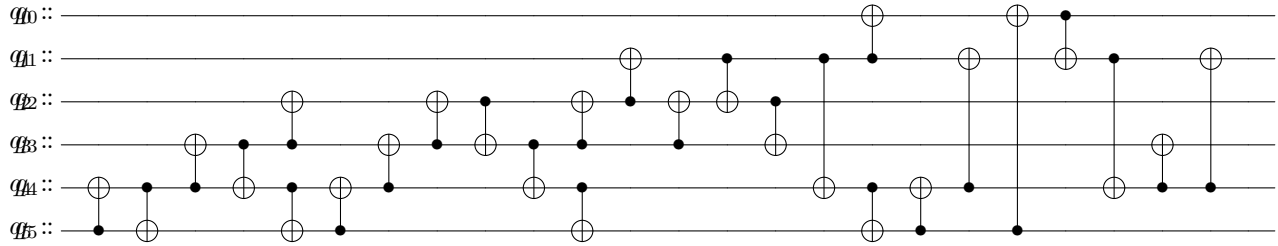
In this appendix, we present the remaining elimination steps of the example presented in the main text, depicted in table A.1. The resulting circuits of this synthesis example appear in figure A.2

Table A.1. Remaining steps of the example from table 2

| Step | Weighted | | Unweighted | |
|---|---|---|---|---|
| Row 2 | $\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&1&1\\0&0&0&1&0&0\\0&0&0&1&1&0\end{pmatrix}$ |  | $\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&1&0\\0&0&0&1&0&1\\0&0&0&0&1&0\\0&0&0&1&1&0\end{pmatrix}$ |  |
| Col 3 | $\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&1&1\\0&0&0&1&0&0\\0&0&0&1&1&0\end{pmatrix}$ |  | $\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&1&1\\0&0&0&0&1&0\\0&0&0&1&1&0\end{pmatrix}$ |  |
| Row 3 | $\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&1&1\\0&0&0&0&1&1\\0&0&0&0&1&0\end{pmatrix}$ |  | $\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&1&1\\0&0&0&0&1&1\\0&0&0&0&1&0\end{pmatrix}$ |  |
| Col 4 | $\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&1\\0&0&0&0&1&0\end{pmatrix}$ |  | $\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&1\\0&0&0&0&1&0\end{pmatrix}$ |  |
| Row 4 | $\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&1\\0&0&0&0&0&1\end{pmatrix}$ |  | $\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&1\\0&0&0&0&0&1\end{pmatrix}$ |  |



(a) weighted



(b) unweighted

Fig. A.2. Resulting circuits