Quantum CNOT circuits synthesis for NISQ architectures using the syndrome decoding problem

Timothée Goubault de Brugière[1,3], Marc Baboulin[1], Benoît Valiron[2], Simon Martiel[3], and Cyril Allouche[3]

[1] Université Paris-Saclay, CNRS, Laboratoire de Recherche en Informatique, 91405, Orsay, France
[2] Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire de Recherche en Informatique, 91405, Orsay, France
[3] Atos Quantum Lab, Les Clayes-sous-Bois, France

Abstract. Current proposals for quantum compilers involve the synthesis and optimization of linear reversible circuits and among them CNOT circuits. This class of circuits represents a significant part of the cost of running an entire quantum circuit and therefore we aim at reducing the size of CNOT circuits. In this paper we present a new algorithm for the synthesis of CNOT circuits based on the solution of the syndrome decoding problem. Our method addresses the case of ideal hardware with an all-to-all qubit connectivity and the case of near-term quantum devices with restricted connectivity. Benchmarks show that our algorithm outperforms existing algorithms in both cases of partial and full connectivity.

Keywords: Quantum Circuit Synthesis · CNOT Circuits · Syndrome Decoding · Reversible Computation · Noisy Intermediate Scaled Quantum Computers (NISQ).

1 Introduction

Quantum compilers transform a quantum algorithm into an optimized sequence of instructions (elementary gates) directly executable by the hardware. The most common universal set of gates for this task is the Clifford+T gate set, used in many quantum architectures [7]. With this setup two resources have to be optimized in priority: the T gate and the CNOT gate. The T gate is considered to be the most costly gate to implement and many efforts have been made to reduce their number in quantum circuits [1, 13, 18]. Yet, when implementing complex quantum algorithms, e.g, reversible functions, it is estimated that the total number of CNOT gates increases much more rapidly with the number of qubits than the number of T gates, and it is likely that the CNOT cost will not be negligible on medium sized registers [13, 21].

Circuits consisting solely of CNOT gates, also called linear reversible circuits, represent a class of quantum circuits playing a fundamental role in quantum compilation. They are part of the so-called Clifford circuits and the CNOT+T circuits, two classes of circuits that have shown crucial utility in the design of efficient quantum compilers [1,13] and error correcting codes [6,12]. For instance the Tpar optimizer [1] takes a Clifford+T circuit as input and decomposes it into a series of CNOT+T circuits separated by Hadamard gates. Then each CNOT+T circuit is optimized and re-synthesized by successive syntheses of CNOT circuits and applications of T gates.

Hence the synthesis of CNOT circuits naturally occurs in general quantum compilers and giving efficient algorithms for optimizing CNOT circuits will then be of uttermost importance.

With the current near term quantum devices, also called Noisy Intermediate Scaled Quantum Computers (NISQ) [26], the synthesis of circuits is subject to constraints on the elementary operations available. In this situation, a physical qubit on the hardware can only interact with its neighbors, restricting the 2-qubit gates —such as CNOT— one can apply. Taking into account these constraints is a crucial and difficult task for the design of quantum algorithms and the optimization of the corresponding quantum circuits. In particular, in the literature several works present post-processing techniques to convert with minimum overhead a circuit designed for an ideal hardware to a circuit designed for a specific architecture [8].

Contribution and outline of the paper. In this paper we focus on the size optimization of linear reversible circuits. We present a new method for the synthesis of CNOT circuits relying on solving a well-known cryptographic problem: the syndrome decoding problem. Our algorithm transforms the synthesis problem into a series of syndrome decoding problems and we propose several methods to solve this particular subproblem. This method, initially designed for a full qubit connectivity, is robust enough to be extended to partial connectivity.

The outline of the paper is the following: in Section 2 we present the basic notions and the state of the art in the synthesis of linear reversible circuits. We first present our algorithm in the case of an all-to-all connectivity in Section 3. Then we extend it to the case of restricted connectivity in Section 4. Benchmarks are given at the end of Sections 3 and 4.

2 Background and state of the art

Synthesis of a linear reversible function. Let $\mathbb{F}_2$ be the Galois field of two elements. A linear reversible function f on n qubits applies a linear Boolean function on the inputs to each qubit. Given $x \in \mathbb{F}_2^n$ as inputs, the output of qubit i is

$$f_i(x) = \alpha^i \cdot x = \alpha_1^i x_1 \oplus \alpha_2^i x_2 \oplus ... \oplus \alpha_n^i x_n$$

where $\oplus$ is the bitwise XOR operation and the α^i's are Boolean vectors also called *parities*. The action of f can be represented as an $n \times n$ binary matrix A

with $A[i,:] = \alpha^i$ (using Matlab notation for row selection) and $f(x) = Ax$. In other words each row of A corresponds to the parity held by the corresponding qubit after application of A. By reversibility of f, A is also invertible in $\mathbb{F}_2$. The application of two successive operators A and B is equivalent to the application of the operator product BA.

We are interested in synthesizing general linear reversible Boolean functions into reversible circuits i.e series of elementary reversible gates that can be executed on a suitable hardware. To that end we use the CNOT gate, it performs the following 2-qubit operation:

$$\mathrm{CNOT}(x_1, x_2) = (x_1, x_1 \oplus x_2).$$

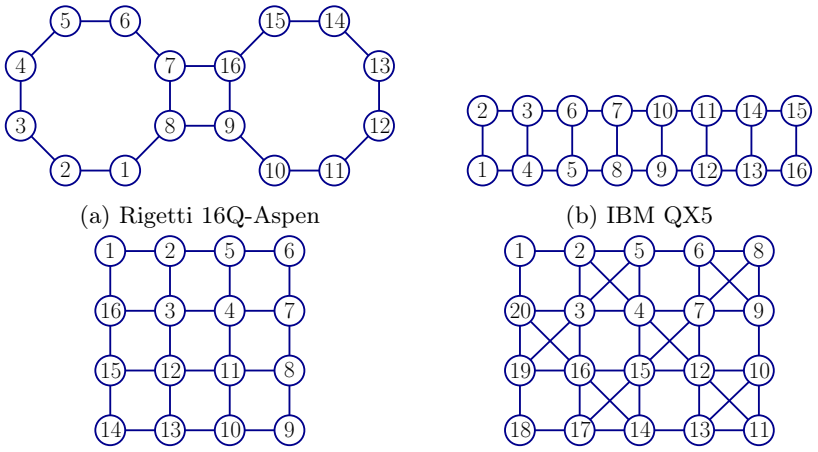
where x_1, resp. x_2, is the parity held by the control qubit, resp. the target qubit. If applied after an operator A, the total operator (A + CNOT) is given from A by adding the row of the control qubit to the row of the target qubit. Such row operations are enough to reduce any invertible Boolean matrix to the identity matrix, so the CNOT gate can be solely used to implement any linear reversible operator. Overall, a CNOT-based circuit can be simulated polynomially: starting from $A = I$ the identity operator, we read sequentially the gates in the circuit and apply the corresponding row operation to A.

We use the size of the circuit, i.e, the number of CNOT gates in it, to evaluate the quality of our synthesis. The size of the circuit gives the total number of instructions the hardware has to perform during its execution. Due to the presence of noise when executing every logical gate, it is of interest to have the shortest circuit possible.

Connectivity constraints. At the current time, for superconducting technologies, full connectivity between the qubits cannot be achieved. The connections between the qubits are given by a connectivity graph, i.e, an undirected, unweighted graph where 2-qubit operations, such as the CNOT gate, can be performed only between neighbors in the graph. Examples of connectivity graphs from current physical architectures are given on Fig. 1.

LU decomposition. Given the matrix representation A of a generic linear reversible operator, we can always perform an LU decomposition [11] such that there exists an upper (resp. lower) triangular matrix U (resp. L) and a permutation matrix P such that $A = PLU$. The invertibility of A ensures that the diagonal elements of L and U are all equal to 1. In the remainder of this paper, the term "triangular operator" stands for an operator whose corresponding matrix is either upper or lower triangular. The LU decomposition is at the core of our synthesis of general linear reversible Boolean operators: synthesizing U, L, P and concatenating the circuits gives an implementation of A.

State of the art. In the unrestricted case the best algorithm reaching an asymptotic optimum is [23, Algo. 1] and produces circuits of size $\mathcal{O}(n^2/\log_2(n))$. This



(a) Rigetti 16Q-Aspen (b) IBM QX5

(c) 16q-Square (d) IBM QX20 Tokyo

Fig. 1: Example of qubit connectivity graphs from existing architectures

algorithm is for instance used in the Tpar and Gray-Synth algorithms [1, 2] so any improvement over [23, Algo. 1] will also improve any quantum compiler that relies on it. In the restricted case the first proposed approach has been to transform the circuits given by an unrestricted algorithm with swap insertion algorithms to match the connectivity constraints [20, 24, 28]. To produce more efficient circuits, two concomitant papers proposed a modification of the Gaussian elimination algorithm [17, 22]. They synthesize the operator column by column similarly to the Gaussian Elimination algorithm but they use Steiner trees to compute the shortest sequence of CNOT gates for the synthesis of one column. In [17] the authors compare their method based on Steiner trees against two compilers: Rigetti Computing's QuilC and Cambridge Quantum Computing's t|ket⟩ that both produced state of the art results on benchmarks published by IBM [9]. The benchmarks show a consequent savings in the total number of CNOT gates in favor of the Steiner tree method, so we consider that the work in [17] is state-of-the-art and we will compare solely to their algorithm.

3 Algorithm for an all-to-all connectivity

In this section we present our algorithm in the case of a complete connectivity between the qubits. We focus on the synthesis of a lower triangular operator $L \in F_2^{n \times n}$. What follows can be straightforwardly extended to the case of upper triangular operators and to general operators using the LU decomposition. With an all-to-all connectivity one can avoid to apply the permutation P by doing a post-processing of the circuit that would transfer the permutation operation directly at the end of the total circuit. This can be done without any overhead in the number of gates.

A circuit implementing L can solely consist of "oriented" CNOTs, whose controlled qubit i and target qubit j satisfy $i < j$. The circuit given by the Gaussian elimination algorithm is an example. For this particular kind of circuits, a CNOT applied to a qubit k does not have any influence on the operations performed on the first $k-1$ qubits: removing such a CNOT will not modify the result of the synthesis of the first $k-1$ parities. We use this property to design a new algorithm where we synthesize L parity by parity and where we reuse all the information acquired during the synthesis of the first k parities to synthesize parity $k+1$.

Given $L_{n-1} = L[1{:}n-1, 1{:}n-1]$ (again using Matlab notation), a circuit C implementing the operator $\left(\begin{smallmatrix} L_{n-1} & 0 \\ 0 & 1 \end{smallmatrix}\right)$ and considering that we want to synthesize the operator $L = \left(\begin{smallmatrix} L_{n-1} & 0 \\ s & 1 \end{smallmatrix}\right)$ the core of our algorithm consists in adding a sequence of CNOTs to C such that we also synthesize the parity s of the n-th qubit. During the execution of C, applying a CNOT $i \rightarrow n$ will add the parity currently held by qubit i to the parity of qubit n without impacting the synthesis of the first $n-1$ parities. In other words, if we store in memory all the parities that appeared on all $n-1$ qubits during the execution of the circuit C, we want to find the smallest subset of parities such that their sum is equal to s. Then when a parity belonging to this subset appears during the execution of C, on qubit i for instance, we insert in C a CNOT $i \rightarrow n$. We ultimately have a new circuit C' that implements L.

The problem of finding the smallest subset of parities whose sum equals s can be recast as a classical cryptographic problem. Assuming that $H \in F_2^{n-1 \times m}$ is a Boolean matrix whose columns correspond to the m available parities, any Boolean vector x satisfying $Hx = s^T$ gives a solution to our problem and the Hamming weight of x, $wt(x)$, gives the number of parities to add, i.e, the number of CNOTs to add to C. We are therefore interested in an optimal solution of the problem

$$\begin{aligned} &\underset{x \in F_2^m}{\text{minimize}} && wt(x) \\ &\text{such that} && Hx = s^T. \end{aligned} \tag{1}$$

Problem 1 is an instance of the *syndrome decoding problem*, a well-known problem in cryptography. The link between CNOT circuit synthesis and the syndrome decoding problem has already been established in [2], yet it was used in a different problem for proving complexity results (under the name of Maximum Likelihood Decoding problem) and the authors did not pursue the optimization. The syndrome decoding problem is presented in more details in Section 3.1.

To summarize, we propose the following algorithm to synthesize a triangular operator L. Starting from an empty circuit C, for i from 1 to n perform the three following steps:

1. scan circuit C to compute all the parities available on a single matrix H,
2. solve the syndrome decoding problem $Hx = s$ with s the parity of qubit i,
3. add the relevant CNOT gates to C depending on the solution obtained.

Provided that the size of C remains polynomial in n, which will be the case, then steps 1 and 3 can be performed in polynomial time and in practice in a very

short amount of time. The core of the algorithm, both in terms of computational complexity and final circuit complexity, lies in Step 2.

3.1 Syndrome decoding problem

In its general form, the syndrome decoding problem is known to be NP-Hard [4] and cannot be approximated by a constant factor [3]. A good overview of how difficult the problem is can be found in [27].

We give two methods for solving the syndrome decoding problem. The first one is an optimal one and uses integer programming solvers. The second one is a greedy heuristic for providing sub-optimal results in a short amount of time.

Integer programming formulation. The equality $Hx = s$ is a Boolean equality of n lines. For instance the first line corresponds to

$$H_{1,1}x_1 \oplus H_{1,2}x_2 \oplus \ldots \oplus H_{1,m}x_m = s_1.$$

We transform it into an "integer-like" equality constraint. A standard way to do it is to add an integer variable t and to create the constraint

$$H_{1,1}x_1 + H_{1,2}x_2 + \ldots + H_{1,m}x_m - 2t = s_1.$$

If we write $c = (1, ..., 1, 0, ..., 0)^T \in \mathbb{N}^{m+n}$ and $A = [H| - 2I_n]$ then the syndrome decoding problem is equivalent to the integer linear programming problem

$$\begin{array}{ll} \min\limits_{x \in F_2^m, t \in \mathbb{N}^n} & c^T \cdot [x; t] \\ \text{such that} & A[x; t] = s. \end{array} \tag{2}$$

A cost minimization heuristic. Although the integer programming approach gives optimal results, it is very unlikely that it will scale up to a large number of qubits. Moreover, to our knowledge the other existing algorithms proposed in the literature give exact results, they are complex to implement and their time complexity remains exponential with the size of the problem. We therefore have to consider heuristics to compute an approximate solution in a much shorter amount of time.

We use a simple cost minimization approach: starting with the parity s we choose at each iteration the parity v in H that minimizes the Hamming weight of $v \oplus s$ and we pursue the algorithm with the new parity $v \oplus s$. The presence of the canonical vectors in H (as we start with the identity operator) is essential because they ensure that this method will ultimately converge to a solution.

A simple way to improve our heuristic is to mimic path finding algorithms like Real-Time A* [19]. Instead of directly choosing the parity that minimizes the Hamming weight, we look up to a certain horizon and we make one step in the direction of the most promising path. To control the combinatorial explosion of the number of paths to browse, we only expand the most promising parities

at each level. We set the maximum width to m and the depth to k so that it represents at most m^k paths to explore. With suitable values of m and k we can control the total complexity of the algorithm. A limitation of such a simple approach is that we can store the same path but with different parities order: we decided to ignore this limitation in order to keep a simple implementation.

Lastly, we introduce some randomness by change of basis and we solve the problem $PHx = Ps$ for several change of basis matrices P. Repeating this several times for one syndrome decoding problem increases the chance to find an efficient solution. This technique has been proven to be efficient for a class of cryptographic algorithms called Information Set Decoding [25], even though the complexity of these algorithms remains exponential.

3.2 Benchmarks

All the code is written in Julia and executed on the QLM (Quantum Learning Machine) located at ATOS/BULL. We generate random operators by generating random circuits with randomly placed CNOT gates. When the number of input gates is sufficiently large we empirically note that the operators generated represent the worst case scenario.

We first generate an average complexity for different problem sizes: for $n = 1..200$ we generated 20 random operators on n qubits with more than n^2 gates to reach with high probability the worst cases. We run our algorithms on this set of operators in the following cases:

- with the integer programming solver (Coin-or branch and cut solver),
- with the cost minimization heuristic with unlimited width and depth 1,
- with the cost minimization heuristic with width 60 and depth 2,
- with the cost minimization heuristic with width 15 and depth 3,
- with the cost minimization heuristic (width=Inf, depth=1) and 50 random changes of basis, the "ISD" case.

In the case of the ISD experiment, due to its probabilistic nature, one can hope that repeating the complete synthesis several times and keeping the shortest circuit would improve the results. Yet the experiments show that it has a minor influence on the final result.

The results are given on Fig. 2. For clarity, instead of plotting the size of the circuits we plot the ratio between the size of the circuits given by our algorithms and the state of the art algorithm [23, Algo. 1]. We stopped the calculations when the running time was too large for producing benchmarks in several hours.

Overall, for the considered range of qubits and for all versions of our algorithm we outperform [23, Algo. 1]. The integer programming solver gives the best results with with a maximum gain of more than 40% but its scalability is limited: beyond 50 qubits it requires too much computational time. Using commercial softwares for reaching larger problem sizes would be interesting to confirm the tendency toward an increasing gain.

Concerning the cost minimization heuristic, it seems better to increase the depth of search than the width. With depth 3 and width 15 we have the best

results for the range 70-125 with 30% of gain. Surprisingly the ISD based method with 50 random changes of basis works well until 60/70 qubits with more than 35% of gain. Then it seems that the number of random changes is not enough to search efficiently an optimal solution and ultimately after 150 qubits the random changes have no effect at all compared to the simpler heuristic with one try. It is possible to increase this number of random changes but this comes at the price of a longer computational time and the ISD method cannot compete with the other versions of the cost minimization heuristic.

As the number of qubits increases our method performs worse. We ran a few computations for much larger problems and the results are that [23, Algo. 1] produces shorter circuits whenever n goes approximately beyond 400. This raises the question of whether it is due to the method in itself or to the solution of the syndrome decoding that becomes less and less optimal as the problem size increases. We leave this question as future work.

We now look at the performance of the algorithms on a specific number of qubits, here $n = 60$, but for different input circuit sizes. This experiment reveals how close to optimal our algorithm is when we synthesize an operator for which we expect a small output circuit. The results are given on Fig. 3. As the ISD method produces the best results for this size of problem we only plot the results for this method. We also plot the line $y = x$ that shows how far we still are from the optimal solution. Again we outperform the best algorithm in the literature even for small input circuits with more than 50% of savings when the input circuit is of size 100-300 gates, with a maximum saving of 60% for 200 gates.

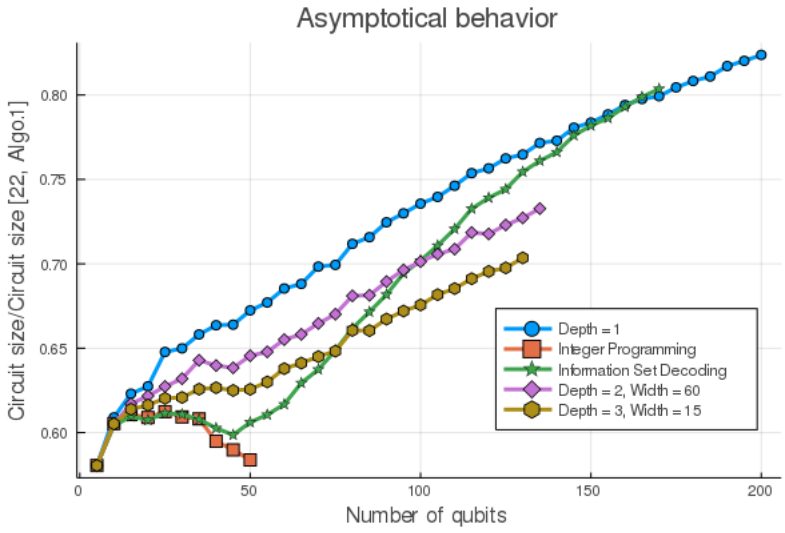


Fig. 2: Average performance of the Syndrome Decoding based algorithms versus the state of the art [23, Algo. 1].

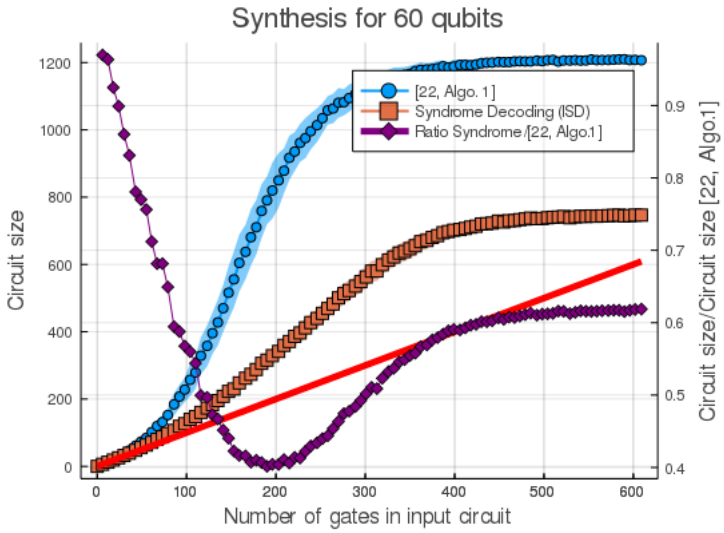


Fig. 3: Performance of Syndrome Decoding based algorithms versus [23, Algo. 1] on 60 qubits for different input circuit sizes.

4 Extension to an arbitrary connectivity

In this section we extend the algorithm to the case where the connectivity is not complete. First we present how to adapt our algorithm based on syndrome decoding for the synthesis of triangular operators, then we extend our method to the synthesis of any general operator.

4.1 Synthesis of a triangular operator

Let G be a qubit connectivity graph and L the lower triangular operator to synthesize. We require an ordering on the nodes of G such that the subgraphs containing only the first k nodes, for $k = 1..n$, are connected. As we need to synthesize both L and U we need in fact this property to be true for an ordering of the qubits and the reverse ordering. An Hamiltonian path in G is enough to have this property so for simplicity we assume that the ordering follows an Hamiltonian path in G.

Even though the native CNOTs in the hardware are CNOTs between neighbor qubits in the connectivity graph, it is possible to perform an arbitrary CNOT gate but this requires more local CNOT gates. Given a target qubit q_t and a qubit control q_c and assuming we have a path $(q_c, q_1, ..., q_k, q_t)$ in the graph connecting the two nodes (such path always exists with the assumption we made above), it is possible to perform the CNOT $q_c \to q_t$ with $\max(1, 4k)$ CNOTs. An example for 4 qubits (with $k = 2$) is given Fig 4.

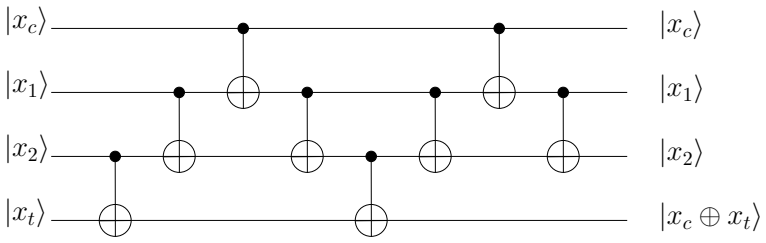


Fig. 4: CNOT in LNN architecture

Hence, it is still possible to perform the synthesis parity by parity but we have to be more careful in the setting and in the solution of the syndrome decoding problem. Not all parities have the same cost, depending on the qubit holding the parity and its position on the hardware. Therefore we have to solve a weighted version of the syndrome decoding problem. Namely once we have a set of parities in a matrix H and a cost vector $c \in \mathbb{N}^m$, we look for the solution of the optimization problem

$$\begin{aligned} &\underset{x \in F_2^m}{\text{minimize}} && c^T \cdot x \\ &\text{such that} && Hx = s^T. \end{aligned} \tag{3}$$

Problem 3 can be recasted again as an integer linear programming problem: we only have to change the value of c. We also propose a greedy heuristic for solving quickly and approximately the problem: we define the "basis cost" of implementing s as the sum of the costs of each canonical vector whose component in s is nonzero. Let $\mathrm{bc}(s)$ be this cost. Our greedy approach consists in finding among the parities of H the parity v (column i of H) that minimizes the cost

$$c[i] + \mathrm{bc}(s \oplus v).$$

This approach gives a good trade-off between zeroing the most costly components of s and applying parities at a very high cost. Again we can repeat the algorithm with random changes of basis to find a better solution. Especially we focused on computing bases for which the canonical vectors have the lowest possible costs.

Nonetheless, compared to the all-to-all case, solving the weighted syndrome decoding problem is not the only computational core for controlling both the quality of the solution and the computational time. Another key task lies in the enumeration of the available parities. As we will see, it is possible to generate more parities for one syndrome decoding problem instance and this increases the chances to get a low-cost solution.

Listing the parities available. Until now we set the weighted syndrome decoding instances by computing the parities appearing during the synthesis and

by using the template in Fig. 4 to estimate their costs. This is in fact inefficient because it ignores some specificities of the problem:

- It is possible to add multiple parities in one shot using the template in Fig. 4.
- There is not necessarily one unique path in G between the control qubit and the target qubit.

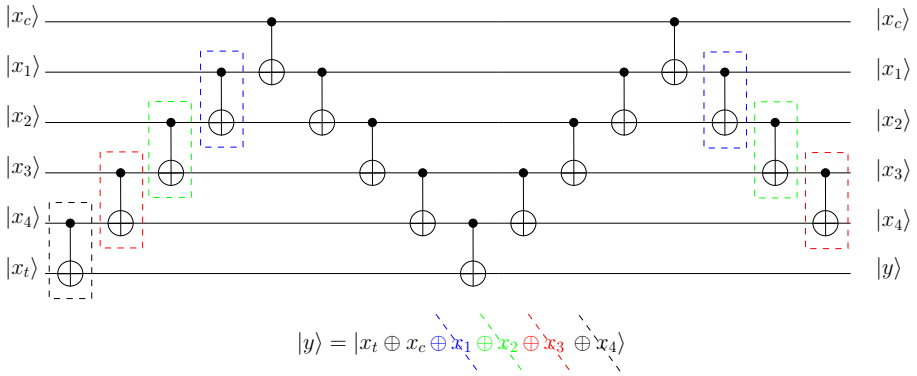


Fig. 5: Fan in CNOT in LNN architecture

More precisely, the template shown in Fig. 4 is the best to our knowledge, in terms of size, to apply solely the parity on qubit q_c to qubit q_t. However it is possible to apply any parity

$$q_t \leftarrow q_t \oplus q_c \oplus_{i=1}^{k} \alpha_i q_i$$

with $\alpha_i \in \{0, 1\}$ using less CNOTs than required for applying only q_c. In fact the less costly linear combination of parities is the complete combination $q_c \oplus q_1 \oplus ... \oplus q_k$, showing that $2k+1$ CNOTs are enough. Removing any parity from this combination requires 2 additional CNOTs per parity except for the qubit q_k that needs only one extra CNOT. An explanatory template on 6 qubits ($k = 4$) is given on Fig. 5. For any parity at a distance k of the target qubit, there is at most 2^{k-1} different linear combinations possible and just as many new parities to consider. Moreover the path between the control qubit and the target qubit matters as a different path will result in different linear combinations of parities. A slight modification of the A* algorithm is enough to compute all the shortest paths between two nodes in a graph.

Even for a small number of qubits the number of parities becomes quickly intractable. The number of linear combinations along a path increases exponentially with the length of the path as the number of paths for most of the architectures — a grid for instance. In practice we control the total number of parities by favoring paths over the choices in the linear combinations. This option is empirically justified but a more detailed analysis could be made. For one

path we only consider the less costly linear combination, i.e, the one that adds all the parities on the way. On the other hand if possible we go through all the shortest paths between one control qubit and one target qubit. We introduce a parameter $P_{\max}$ equal to the maximum number of shortest paths we consider between two qubits.

4.2 Synthesis of a general operator

The extension of the synthesis from triangular to general operator is not as straightforward as in the all-to-all connectivity case. We cannot simply write $A = PLU$ and concatenate the circuits synthesizing L and U and ultimately permuting the qubits. If we want to use this algorithm as a sub-task of a global circuit optimizer for NISQ architectures we cannot afford to swap the qubits because it could break the optimizations done in the rest of the circuit.

To avoid the permutation of the qubits we have to transform the matrix A by applying a pre-circuit C such that $CA = LU$. Then the concatenation of C^{-1} and the circuits synthesizing L and U gives a valid implementation of A.

Computation of C. If A is invertible, which is always the case, then it admits an LU factorization if and only if all its leading principal minors are nonzero. We propose an algorithm for computing C exploiting this property while trying to optimize the final size of C. We successively transform A such that every submatrix A[1:i, 1:i] is invertible. By construction when trying to make $A[1 : k, 1 : k]$ invertible for some k we have $A[1{:}k-1, 1{:}k-1]$ invertible. If $A[1 : k, 1 : k]$ is invertible then we do nothing, otherwise we look in the parities $A[k+1{:}n, 1{:}k]$ those who, added to $A[k, 1 : k]$, make $A[1 : k, 1 : k]$ invertible. By assumption A is invertible so there is at least one such row that verifies this property. Then among the valid parities we choose the closest one to qubit k in G. We can add all the parities along the path because by assumption they belong to the span of the first $k-1$ rows of $A[1 : k, 1 : k]$ so it has no effect on the rank of $A[1 : k, 1 : k]$.

Choice of the qubit ordering. A last optimization can be performed by changing the qubits ordering. The algorithm we have presented for synthesizing a triangular operator is still valid up to row and column permutations. Thus, given a permutation P of the qubits, one can synthesize $P^{-1}LP$ by applying our algorithm with the order given by P. Then, instead of computing a circuit C such that $CA = LU$ we search for a circuit C satisfying $P^{-1}CAP = LU$ and

$$CA = PLP^{-1}PUP^{-1} = L'U'$$

where L' and U' can be synthesized using our algorithm. Searching for such C can be done using our algorithm on $A[P, P]$ (in Matlab notation, i.e. the reordering of A along the vector P).

This means that we can choose P such that the synthesis of L and U will yield shorter circuits. Empirically we noticed that when synthesizing the k^{th}

parity of L it is preferable to have access to the parities appearing on qubits $k-1, k-2$ etc. in priority for two reasons: first because they can modify more bits on the k^{th} parity and secondly because it is likely that there will be much more parities available, increasing the chance to have an inexpensive solution to the weighted syndrome decoding problem. Intuitively we want the ordering of the qubits to follow at least an Hamiltonian path in $G = (V, E)$ which would match the previous restriction on the ordering we formulated at the beginning of the section. We formulate the best ordering $\pi : V \to [\![1, n]\!]$ as a solution of the Minimum Linear Arrangement problem

$$\underset{\pi}{\text{minimize}} \quad \sum_{(u,v)\in E} w_{uv} \, |\pi(u) - \pi(v)| \tag{4}$$

where w_{uv} is the weight of the edge connecting u and v in the graph. Here we want to give priority to neighbors in the hardware: the nodes must be as close as possible in the hardware if their "numbers" are also close. A way to do so is to solve the MinLA problem, not in the hardware graph, but in the complete graph with suitable weights. Namely w_{ij} must be large when i, j are neighbors in the hardware and w_{ij} must be smaller if i, j are at distance 2 etc. The MinLA problem has already been used for qubit routing [24] and the problem is in general NP-Hard [10]. In our case we did not use any heuristic for solving this problem: all the architectures chosen to test our method have Hamiltonian paths and we simply chose manually a suitable ordering of the qubits. Fig. 6 features the choices we made for some architectures: nodes are labeled with their position in the linear arrangement. We leave as a future work the inclusion of the solution of the MinLA problem in our algorithm.

4.3 Benchmarks

We compare our method against the best algorithm in the literature [17] whose source code is available on the PyZX Github repository [16]. For each architecture considered in their implementation we generate a set of 100 random operators and perform the synthesis using the Steiner trees. Their algorithm provides an optimization using genetic algorithms but this implements the circuit up to a permutation of the qubits. As we focus on implementing exactly the operator we considered their algorithm without this extra optimization.

Our own algorithm is implemented in Julia. The experiments have been carried out on one node of the QLM (Quantum Learning Machine) located at ATOS/BULL. We set a time limit of 10 minutes for the synthesis of an operator. We recall that $P_{\max}$ is the maximum number of shortest paths considered between two qubits. We also set $N_{\text{iter\_syndrome}}$ to be the number of iterations for the solution of a decoding syndrome and N_{iter} the number of times that the synthesis has been repeated. The values of the parameters for the different problem sizes are the following:

- $n < 36$, $P_{\max} = \text{Inf}$, $N_{\text{iter}} = 100$ and $N_{\text{iter\_syndrome}} = 1$,

Architecture	#	Steiner [17]	Syndrome	Saving				t_{St} (s)	t_{Sy} (s)
				Mean	Min.	Max.	Positive		
9q Square	9	60	56	6%	-25.5%	40.6%	66%	0.01	0.16
Rigetti 16q	16	272	245	10%	-6%	23.1%	97%	0.022	1.6
IBM QX 5	16	245	195	20.2%	10%	28.7%	100%	0.019	2
16q Square	16	205	183	10.7%	-7.1%	33%	93%	0.02	1.6
19q Line	19	455	470	-6.7%	-19.4%	7.9%	6%	0.045	4.5
IBM Q20 Tokyo	20	292	239	18.1%	8.7%	26.9%	100%	0.025	1.8
25q Square	25	512	458	10.6%	3.5%	19.1%	100%	0.04	19
25q Sq. + diag.	25	410	324	21%	10.7%	28.3%	100%	0.035	8
36q Square	36	1067	891	16.5%	11%	22.4%	100%	0.1	67
36q Sq. + diag.	36	861	667	22.5%	18.2%	26.6%	100%	0.09	22
49q Square	49	1981	1662	16%	11.9%	20.7%	100%	0.2	420
49q Sq. + diag.	49	1607	1246	22.4%	19%	25.2%	100%	0.19	114
64q Square	64	3374	2812	16.6%	13.9%	18.9%	100%	0.54	79
81q Square	81	5363	4447	17%	14.4%	19.2%	100%	1.04	192
100q Square	100	8148	6666	18.2%	16.8%	19.8%	100%	2.1	449

Table 1: Performance of our Syndrome Decoding based algorithm vs Steiner trees algorithm [17] for several architectures

- $n = 36,\ P_{\max} = \text{Inf},\ N_{\text{iter}} = 20$ and $N_{\text{iter\_syndrome}} = 1$,
- $n = 49,\ P_{\max} = 10,\ N_{\text{iter}} = 100$ and $N_{\text{iter\_syndrome}} = 1$,
- $n > 49,\ P_{\max} = 1,\ N_{\text{iter}} = 100$ and $N_{\text{iter\_syndrome}} = 1$.

The results are summarized in Table 1. Columns 3 and 4 give the average size of the generated circuits for the method using Steiner trees in [17] and our algorithm based on syndrome decoding. The next columns detail the savings: the mean saving, the minimum saving (negative saving means that our algorithm performs worse), the maximum saving and the proportion of operators for which our circuit is actually shorter than the one provided by the state-of-the-art method. The last two columns give the average time required to perform the synthesis of one operator (all iterations included for our algorithm).

We can expect our algorithm to behave better if there are more connections between the qubits. When the connectivity is as limited as possible, for instance with an LNN architecture, our algorithm does not outperform the algorithm based on Steiner trees. Except for 6% of the operators where we have a slight gain (less than 8%) we provide circuits with more gates, up to 19%. For other architectures the results are more promising. In the case of the 9-qubit square there is a lot of variance in the results: depending on the operator we can have a gain of 40% or a loss of 25%. Overall we still manage to produce a shorter circuit

66% of the time. For larger square architectures, we outperform the state-of-the-art algorithm consistently with increasing savings, between 10% for the 25 qubits square to 18% for the 100 qubits square. When adding diagonal connections in the square architectures the results are even better. This shows that improving just slightly the connectivity can improve consistently the results of our algorithm compared to the state of the art method. Finally on specific architectures we also provide better results. The results for Rigetti's chip are not as good as for IBM's chips essentially because the connectivity is still close to a straight line, otherwise we manage to have a saving of around 20% for both IBM-QX5 and IBM-Tokyo chips.

5 Conclusion

We have presented a new framework for the synthesis of linear reversible circuits. We exploit the specific structure of triangular operators to transform the synthesis into a series of syndrome decoding problems, which are well-known problems in cryptography. Using an LU decomposition we can synthesize any quantum operator in the case of an all-to-all connectivity. Benchmarks show that we outperform a state-of-the-art algorithm for intermediate sized problems ($n < 400$). Our heuristics for solving the syndrome decoding problem are efficient but could be still improved, both in circuit size and computational time. For instance, some quantum algorithms have been proposed for solving the syndrome decoding problem via the Information Set Decoding algorithm [5, 14, 15], which gives the possibility of designing a hybrid quantum/classical compiler for this particular synthesis problem.

Then we have highlighted the robustness of our framework by extending it to an arbitrary connectivity graph having a Hamiltonian path. With a suitable pre-processing of the matrix we transform the problem into a series of weighted syndrome decoding problems. Except for the LNN architecture whose connectivity is too sparse, we consistently outperform existing algorithms by a percentage that increases with the number of qubits. As a future work, we can study how to extend our method to the case where the connectivity graph does not have a Hamiltonian path, similarly to [17].

Acknowledgment

This work was supported in part by the French National Research Agency (ANR) under the research project SoftQPRO ANR-17-CE25-0009-02, and by the DGE of the French Ministry of Industry under the research project PIA-GDN/QuantEx P163746-484124. We thank Bertrand Marchand for comments on the manuscript.

References

1. Amy, M., Maslov, D., Mosca, M.: 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)
2. Amy, M., Azimzadeh, P., Mosca, M.: On the controlled-NOT complexity of controlled-NOT–phase circuits. Quantum Science and Technology **4**(1), 015002 (2018)
3. Arora, S., Babai, L., Stern, J., Sweedyk, Z.: The hardness of approximate optima in lattices, codes, and systems of linear equations. Journal of Computer and System Sciences **54**(2), 317–331 (1997)
4. Berlekamp, E., McEliece, R., Van Tilborg, H.: On the inherent intractability of certain coding problems. IEEE Transactions on Information Theory **24**(3), 384–386 (1978)
5. Bernstein, D.J.: Grover vs. McEliece. In: International Workshop on Post-Quantum Cryptography. pp. 73–80. Springer (2010)
6. Campbell, E.T., Anwar, H., Browne, D.E.: Magic-state distillation in all prime dimensions using quantum Reed-Muller codes. Physical Review X **2**(4), 041021 (2012)
7. Campbell, E.T., Terhal, B.M., Vuillot, C.: Roads towards fault-tolerant universal quantum computation. Nature **549**(7671), 172–179 (2017)
8. Childs, A.M., Schoute, E., Unsal, C.M.: Circuit transformations for quantum architectures. In: 14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019). LIPIcs, vol. 135, pp. 3:1–3:24 (2019)
9. Cowtan, A., Dilkes, S., Duncan, R., Krajenbrink, A., Simmons, W., Sivarajah, S.: On the qubit routing problem. In: 14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019). LIPIcs, vol. 135, pp. 5:1–5:32 (2019)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co (1979)
11. Golub, G.H., Van Loan, C.F.: Matrix Computations. The Johns Hopkins University Press, Baltimore (1996), third edition
12. Gottesman, D.: Stabilizer Codes and Quantum Error Correction. Ph.D. thesis, Caltech (1997)
13. Heyfron, L.E., Campbell, E.T.: An efficient quantum compiler that reduces T count. Quantum Science and Technology **4**(1), 015004 (2019)
14. Kachigar, G., Tillich, J.P.: Quantum information set decoding algorithms. In: International Workshop on Post-Quantum Cryptography. pp. 69–89. Springer (2017)
15. Kirshanova, E.: Improved quantum information set decoding. In: International Conference on Post-Quantum Cryptography. pp. 507–527. Springer (2018)
16. Kissinger, A.: PyZX. `https://github.com/Quantomatic/pyzx`
17. Kissinger, A., van de Griend, A.M.: CNOT circuit extraction for topologically-constrained quantum memories (2019), draft available as arXiv:1904.00633.
18. Kissinger, A., van de Wetering, J.: Reducing T-count with the ZX-calculus (2019), draft available as arXiv:1903.10477.
19. Korf, R.E.: Real-time heuristic search. Artificial intelligence **42**(2-3), 189–211 (1990)
20. Li, G., Ding, Y., Xie, Y.: Tackling the qubit mapping problem for NISQ-era quantum devices. In: International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 1001–1014 (2019)

21. Maslov, D.: Optimal and asymptotically optimal NCT reversible circuits by the gate types. Quantum Information & Computation **16**(13-14), 1096–1112 (2016)
22. Nash, B., Gheorghiu, V., Mosca, M.: Quantum circuit optimizations for NISQ architectures (2019), draft available as arXiv:1904.01972.
23. Patel, K.N., Markov, I.L., Hayes, J.P.: Optimal synthesis of linear reversible circuits. Quantum Information & Computation **8**(3), 282–294 (2008)
24. Pedram, M., Shafaei, A.: Layout optimization for quantum circuits with linear nearest neighbor architectures. IEEE Circuits and Systems Magazine **16**(2), 62–74 (2016)
25. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962)
26. Preskill, J.: Quantum computing in the NISQ era and beyond. Quantum **2**, 79 (2018)
27. Vardy, A.: Algorithmic complexity in coding theory and the minimum distance problem. In: Symposium on Theory of Computing. pp. 92–109. ACM (1997)
28. Wille, R., Keszocze, O., Walter, M., Rohrs, P., Chattopadhyay, A., Drechsler, R.: Look-ahead schemes for nearest neighbor optimization of 1D and 2D quantum circuits. In: Asia and South Pacific Design Automation Conference. pp. 292–297. IEEE (2016)