An ASP Approach for the Synthesis of CNOT Minimal Quantum Circuits

Carla Piazza[1], Riccardo Romanello[1,*] and Robert Wille[2]

[1]Dipartimento di Scienze Matematiche, Informatiche e Fisiche, Università degli Studi di Udine

[2]Chair for Design Automation Technical University of Munich Munich, Germany Software Competence Center Hagenberg GmbH Hagenberg, Austria

Abstract

In the last year, physical working Quantum Computers have been built and made available for the end users. Such devices, working under the rules of Quantum Mechanics, can only apply a finite set of one/two qubit operations that form a universal set of gates. Single qubit gates are fault-tolerant, while the same cannot be said for two qubit gates. Hence, unitary matrices adopted in Quantum Algorithms must be synthesized in terms of this universal set of operations to obtain a quantum circuit. This synthesis procedure, however, is not constraint-free. In fact, we prefer circuits with minimum number of qubits and with minimum circuit depth. Clifford+T universal set is one of the most adopted in the literature for synthesis. In such set we have 3 single qubit gates and the CNOT, which is a two qubit gate. Many efforts have been directed to devise algorithms that synthesize general unitary matrices into Clifford+T circuits. These algorithms usually tend to optimize circuit depth or eventually the number of T gates. Since two qubit gates are not fault tolerant, in this work we propose an ASP based technique to minimize the number of CNOT gates inside a Clifford+T circuit. We start from a SAT encoding of the problem, and we translate it into an ASP model over a graph, by first working with a generic graph, and then by adopting the structure of a layered DAG. We provide experimental evidence of the scalability of our proposal.

Keywords

Quantum Circuit Synthesis, Answer Set Programming, CNOT Minimization, Clifford+T Synthesis

1. Introduction

In the last few years, a lot of effort has been put into Quantum Computing from both academia and companies. One of the goals is to increase the number of qubits Quantum Computers are able to manage. In fact, the so called *Quantum speed-up* [1] becomes interesting when a reasonably high number of qubits come into play.

Assuming that we have solved the problem of manipulating a reasonable amount of qubits, it remains to consider the difficulty of compiling a quantum algorithm over a real physical device.

CILC'23: 38th Italian Conference on Computational Logic, June 21–23, 2023, Udine, Italy

*Corresponding author.

†This work is partially supported by PRIN project NiRvAna and GNCS-INdAM project "Algoritmi Quantistici su Grafi" CUP_E53C22001930001.

carla.piazza@uniud.it (C. Piazza); riccardo.romanello@uniud.it (R. Romanello); robert.wille@tum.de (R. Wille)
https://users.dimi.uniud.it/~carla.piazza (C. Piazza); https://riccardoromanello.github.io (R. Romanello); https://www.cda.cit.tum.de/team/wille/ (R. Wille)

We take into account Quantum Computers that are built using the superconducting qubits technology. In such architecture, qubits are edited using gates—unitary matrices. Nevertheless, not all unitary operations can be directly implemented in physical hardware. Up to now, only single and two qubit gates can be physically manufactured. The former are more reliable, but may still fail. The latter are error prone in the superconducting qubit environment, while are more reliable in architectures like photonics or Rydberg atoms. The operations that can be physically applied in a Quantum Computer are called *base of gates*. When a generic unitary matrix $U \in \mathbb{C}^{2^n \times 2^n}$ has to be applied to a physical set of qubits, it must be rewritten in terms of given base of gates, i.e., it has to be synthesized. If a base can synthesize correctly any unitary $U \in \mathbb{C}^{2^n \times 2^n}$, then it is called an *universal set of gates*. Once a universal set of gates $\mathcal{B}$ is defined, the problem of rewriting a given unitary in terms of $\mathcal{B}$ is called *synthesis*. This problem has a complexity that is polynomial in the size of the input unitary. Hence, it has a $\Omega(2^n)$ lower bound running time.

Anyway, a lot of effort has been put to create algorithms that synthesize any unitary in a reasonable time and with a low circuit depth and width — often being satisfied with an approximated synthesis. The circuit depth is the length of the circuit, while the width is another term for the number of qubits used. The former has to be kept low since the coherence time of quantum states is not really high, hence circuits that are too long are doomed to fail in their computation. The latter is strictly related to the issue we addressed at the beginning: number of qubits in physical Quantum Computers is currently limited.

Most of the synthesis algorithms are devised using the Clifford+T set of universal gates. Such set is composed by 3 single qubit gates and a two qubits gate. Different techniques have been adopted for the synthesis problem. For example, exact techniques with QMDD have been proposed in [2, 3, 4]. Also heuristics have been adopted trying to obtain a good balance between depth and width of the output circuits [5, 6]

Sometimes, it is also reasonable to think about sub-problems that arise from the synthesis of circuits. One idea can be to focus into the minimization of the number of occurrences of one type of gate inside a universal set. In [7], the authors devised an algorithm to minimize the number of T gates into a circuit. A similar problem was tackled in [8] where the authors proposed a SAT approach for the minimization of the number of CNOT gates in a T-optimal circuit — a circuit where the optimal number of T gates was already achieved.

This kind of problem is really important since the CNOT gate is the only two qubits gate in the Clifford+T base set. Two qubits gates cannot be physically implemented without introducing errors. Hence, their minimization could also lead to a reduction of the effort required for Quantum Error Correction [9].

For this reason, we consider the same problem as in [8], but we try to solve it with a different approach. We take as input a T-optimal circuit $\mathcal{C}$ made only of CNOT and T gates. We want to produce a circuit $\mathcal{C}'$ that preserves the behaviour of $\mathcal{C}$ and with the minimum number of CNOT gates. The approach in [8] starts by reducing the problem to a classical one using the notion of phase polynomial representation. Secondly, a satisfiability model is developed to obtain a solution to the following problem: can the circuit $\mathcal{C}$ be written with l CNOT gates?. Hence, the model is queried for each possible value of l starting from 1. When the first solution is found, it is given as output. Starting from this solution, we decided to propose a solution to the same problem with two major differences. The first is to encode the problem in an Answer

Set Programming model instead that using a SAT encoding. The second is to use ASP solvers properties to find the minimum number of CNOT gates, without doing an iteratively approach as in [8].

The paper is structured as follows: Section 2 contains the definition of the problem and its translation to a classical one. Some results about circuit synthesis coming from the literature are presented in Section 3. Section 4 and Section 5 are used to describe the two different encodings we propose to solve the given problem. After that, Section 6 is devoted to experimental evaluation. Some conclusions and ideas for future works are drawn in Section 7. For space reason we omit the introduction of ASP, the reader may refer to [10].

2. Problem Statement

Modern Quantum Computers (like Osprey from IBM) have two important limits:

- Only a finite set of operations can be applied to the qubit
- The quantum coherence can be maintained for a short amount of time

The finite set of operations a Quantum Computer can apply is usually called *Universal Set of Gates*. Quantum Algorithms can be described as a single $2^n \times 2^n$ unitary matrix $\mathcal{U}$.

The process of rewriting $\mathcal{U}$ in terms of gates of a Universal Set of Gates is called *synthesis*.

The most used Universal Set of Gates for synthesis is the Clifford + T, which contains the following single qubit gates:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \qquad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$$

and the two qubits gate CNOT:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

With the following effect:

$$CNOT\,(a, b) = (a, a \oplus b)$$

where a, b are the *control* and the *target* of the CNOT gate, respectively. Its effect is twofold:

- it leaves the control qubit unchanged
- the target is flipped if the control is 1 — a XOR between control and target

Single qubit gates can be implemented in physical quantum computers with no errors. On the other hand, two qubit gates introduce error that must be corrected using Quantum Error Correction (QEC) techniques. Hence, it is reasonable to think about synthesis techniques that aim at minimizing the number of CNOT gates in the resulting circuit.

In this paper, we will start from a circuit made of CNOT and T gates (same problem tackled in [8]), and we will minimize the number of CNOT gates. To do so, we first turn a *Quantum* problem, into a classical one using the following definition from [11]:

Definition 1 (phase polynomial representation). *The action of a {CNOT, T} circuit on the initial state* $|x_1, x_2, \cdots x_n\rangle$ *has the form:*

$$|x_1, x_2, \cdots x_n\rangle \mapsto e^{i\frac{\pi}{4}p(x_1,x_2,\cdots,x_n)} |g(x_1, x_2, \cdots x_n)\rangle$$

with $p(x_1, x_2, \cdots, x_n)$ *defined as:*

$$p(x_1, x_2, \cdots, x_n) = \sum_{i=1}^{k} (c_i \ mod\, 8) f_i(x_1, x_2, \cdots x_n)$$

where $g : \mathbb{B}^n \to \mathbb{B}^n$ *is a linear reversible function and* p *is a linear combination of linear boolean functions* $f_i : \mathbb{B}^n \to \mathbb{B}$.
The coefficients c_i *(reduced modulo 8) measure the number of* $\pi/4$ *rotations applied to each* f_i.
The number k *represents the number of T gates in the circuit.*

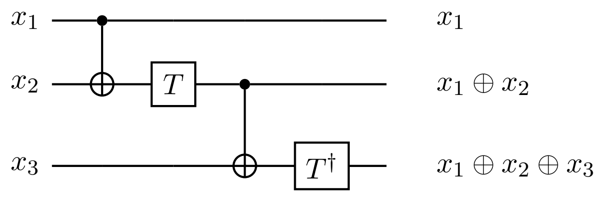
Each circuit has its phase polynomial representation, uniquely defined by

$$g, f_i, c_i \text{ for } i = 1, 2, \ldots k.$$

More than one {CNOT, T} circuit can share the same phase polynomial representation.

Since g is a linear reversible function with n inputs and n outputs, it can be written as a $n \times n$ boolean matrix G. On the other hand, each f_i can be expressed as a boolean row vector F_i.

Example 1. *Consider the following circuit:*



Its phase polynomial representation is the following:

$$G = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \qquad F_1 = \begin{pmatrix} 1 & 1 & 0 \end{pmatrix} \qquad F_2 = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$$

□

Since the number of T gates in the input circuit is supposed to be optimal, the problem we want to solve is the following:

- **INPUT**: $G, F_1, F_2, \cdots F_k$
- **OUTPUT**: a sequence of CNOT gates to be applied such that the final behaviour of the circuit is the one described by G.

The constraints we must fulfill are the following:

- We can only apply CNOT gates
- For each F_i with $i \in \{1, 2, \cdots, k\}$, there must exist a moment during the computation in which a row of G is exactly F_i.

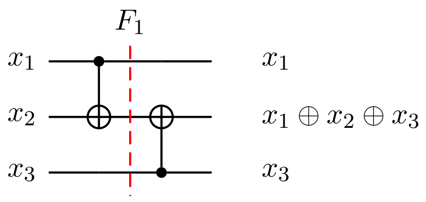
The two constraints allow us to avoid caring about the T gates. We just have to create the *correct* slots for them to be applied, but nothing more.

Example 2. *Let G and F_1 be defined as follows:*

$$G = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \qquad F_1 = (1\,1\,0)$$

We must find a circuit such that x_1 and x_3 are unchanged while x_2 becomes the XOR of all the three variables. Moreover, it must be true that at some point one of the variables must be the XOR between x_1 and x_2 to match the constraint on F_1.

The following circuit, made only of CNOT gates, solves the problem.



We pointed out with the red line the moment in which the constraint on F_1 is satisfied.

3. Related Works

Circuit synthesis has become a more and more prominent problem since the first physically working quantum computers were manufactured. A generic synthesis algorithm takes two inputs: a generic unitary matrix $U \in \mathbb{C}^{2^n \times 2^n}$ and a universal set of gates $\mathcal{B}$. The output must be a circuit made only of gates from $\mathcal{B}$ that implements exactly U. Clearly, we are not interested in any type of circuit. We are interested in keeping the circuit (i) as short as possible, because of the very volatile qubits state, (ii) as thin as possible to use the minimum number of qubits.

Different approaches and algorithms have been proposed. A large branch of them is composed by the ones using Quantum Multivalued Decision Diagrams (QMDD) [2].

QMDDs have been introduced as a means for the efficient representation and manipulation of quantum gates and circuits. The fundamental idea is a recursive partitioning of the respective transformation matrix and the use of edge and vertex weights to represent various complex-valued matrix entries. More precisely, a transformation matrix of dimension $2^n \times 2^n$ is successively partitioned into four sub-matrices of dimension $2^{n-1} \times 2^{n-1}$. This partitioning is represented by a directed acyclic graph - the QMDD.

In [4], the authors exploited the properties of QMDDs to devise a classical algorithm for the synthesis of circuits made of Clifford gates (Hadamard, Phase and CNOT). Such algorithm was extended for the Clifford+T case in [3].

Not only exact algorithms have been defined. In [5], the authors proposed an evolutionary (genetic) algorithm to solve the problem of circuit synthesis. Similar problem is solved in [12]. The optimization technique adopted was a Meet-in-the-Middle one, and in this case the problem of reducing the number of qubits used by synthesized circuit was tackled. An evolutionary algorithm was again used in [6]. In this case the authors proposed a quantum-inspired algorithm. It is worth noticing that in that proposal more than one universal set of gates is considered.

Synthesis problem of Clifford+T circuits is not always addressed as one big problem. A lot of authors also tackled some sub-problems like the minimization with respect to one single gate in the universal set. One example is the minimization of the CNOT gates.

For such problem, an algebraic approach is given in [13] where the authors start by first describing the group structure underlying quantum circuits generated by CNOT gates. Such results are then exploited to create heuristics useful to reduce the number of CNOT in circuits.

A technique based on Steiner Trees has been adopted in [14]. In this case, the authors also took into account the problem of the actual neighbour relation between qubits. When working with physical quantum devices, not every qubit is connected to all the others. Hence, when applying a CNOT between two qubits, we must be sure that these two qubits are linked. Otherwise, we must apply also swap gates, which correspond to 3 CNOT gates.

Last but not least, the SAT based approach has been proposed in [8]. Authors aimed at minimizing the number of CNOT gates in a circuit by setting up the solution of a satisfiability problem. Such solution is then iteratively queried until a first feasible model is found and returned as result to the main problem. The SAT solver will eventually be queried for at most n^2 times, where n is the number of qubits, because of this the following result from [15]:

Theorem 1. *Any n-qubit circuit of CNOT gates is equivalent to one composed of at most n^2 gates.*

A SAT encoding is used also in [16]. The authors noticed that synthesis can be related to the problem of finding best boolean chains to represent functions. Hence, they proposed a DAG based approach to generate all possible DAG structures with a fix number of nodes and then query the satisfiability model to check if such structure would solve the problem.

4. Modelling with a Graph

The first model we propose is a graph based one. The circuit from Example 1 (without the T gates) can be interpreted as the labelled graph $\mathcal{G} = (V, E)$ depicted in Figure 1.

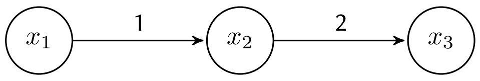

$$\begin{tikzcd} x_1 \arrow[r, "1"] & x_2 \arrow[r, "2"] & x_3 \end{tikzcd}$$

Figure 1: Circuit from Example 1 as a graph.

CNOT gates are encoded as labelled edges. The first CNOT to apply is the one with control x_1 and target x_2, encoded by the edge labelled 1. The second CNOT has control x_2 and target

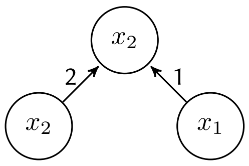


Figure 2: Solution of Example 3 with graph model in 2 time steps.

x_3, and it is encoded by the edge labelled 2.

Edge $e = (x_i, x_j, h)$ means that the h-th CNOT to apply in the circuit has x_i as control and x_j as target. Such edge is represented as:

$$\begin{tikzcd} x_i \arrow[r, "h"] & x_j \end{tikzcd}$$

This idea can be generalized to any CNOT circuit.

We solve the problem introduced in Section 2 by computing $\mathcal{G} = (V, E)$ starting from $G \in \{0,1\}^{n\times n}$ and a set of F_i. The set of nodes V is always $\{x_1, x_2, \cdots x_n\}$. Our aim is to build the set E of minimum size l.
First of all, it must hold that for each time step $t \leq l$, exactly one edge labelled t must exist. After all the l steps, the following has to be true:

$$\textsc{val}_l(x_i) = G_i \qquad \forall i \in \{1, 2, \cdots, n\}$$

where $\textsc{val}$ is defined as follows:

$$\begin{aligned}
&\textsc{val}_0(x_i) = x_i && \forall i \in \{1, 2, \cdots, n\} \\
&\textsc{val}_t(x_i) = \textsc{val}_{t-1}(x_i) && \text{if } \nexists x_k \mid (x_k, x_i, t) \in E \wedge t > 0 \\
&\textsc{val}_t(x_i) = \textsc{val}_{t-1}(x_i) \oplus \textsc{val}_{t-1}(x_k) && \text{if } \exists x_k \mid (x_k, x_i, t) \in E \wedge t > 0
\end{aligned}$$

With the notation $x_j \in \textsc{val}_t(x_i)$, we mean that the XOR operation described by $\textsc{val}_t(x_i)$ contains x_j.

The above constraint was necessary to ensure that the final circuit respects the limitations imposed by G.

As far as F_i is concerned, the following must hold:

$$\forall F_i \, \exists t \leq l \, \exists x_j \mid \textsc{val}_t(x_j) = F_i$$

Example 3. *Suppose G and F_1 are defined as in Example 2. Then the graph depicted in Figure 2 is the one obtained with our model:*

The $\textsc{val}$ *of x_1 and x_3 evolves as follows:*

$$\begin{aligned}
\textsc{val}_0(x_1) = \textsc{val}_1(x_1) = \textsc{val}_2(x_1) = x_1 = (1\,0\,0) \\
\textsc{val}_0(x_3) = \textsc{val}_1(x_3) = \textsc{val}_2(x_3) = x_3 = (0\,0\,1)
\end{aligned}$$

while for x_2 we obtain:

$$\begin{aligned}
\textit{VAL}_0(x_2) &= x_2 \\
\textit{VAL}_1(x_2) &= x_1 \oplus x_2 = (1\,1\,0) \\
\textit{VAL}_1(x_2) &= x_1 \oplus x_2 \oplus x_3 = (1\,1\,1)
\end{aligned}$$

The problem is solved since $\textit{VAL}_2(x_i) = G_i$ *for every* i *and* $\textit{VAL}_1(x_2) = F_1$.

4.1. Encoding in CLINGO

We now briefly describe how we encoded the aforementioned solution in the CLINGO solver.

The input has been encoded in a very simple way. We provide the model with n (the size of G) and k, the number of different F. Then, with a predicate of the form $\text{G}(i, j, b)$ we describe the matrix G, with the following rules:

$$\begin{aligned}
G_{i,j} &= 1 \leftrightarrow \text{G}(i, j, 1) \\
G_{i,j} &= 0 \leftrightarrow \text{G}(i, j, 0)
\end{aligned}$$

where $i, j \in \{1, 2, \cdots, n\}$

The same relation holds between predicate $\text{F}(i, j, b)$ and F. The only difference is that the domain of variable i is $\{1, 2, \cdots, k\}$.

The variables we adopted to solve the problem have two types:

- NODE that describes a node of the graph. Given an $n \times n$ boolean matrix, we will have exactly n NODES with labels $1, 2, \cdots, n$. The mapping from nodes to the variables x_i is straightforward: node i describes the variable x_i—the variable that will have to match G_i.

- STEP, which describes the labelling of the edges.

Notice that we encoded this solution as a *satisfiability* problem. We provide the model with a *candidate* number of CNOT gates l, and it tries to compute a graph with exactly l edges. Therefore, the domain of the STEP variable is $1, 2, \cdots, l$. The predicate describing an edge is:

$$\text{EDGE}(X, Y, I)$$

which holds if and only if the edge at STEP I goes from NODE X to NODE Y. Of course, we fixed all the constraints to make sure that for each STEP I there exists exactly one edge with label I.

For what concerns the VAL of a node, we introduced the predicate

$$\text{VALUE}(X, I, Y)$$

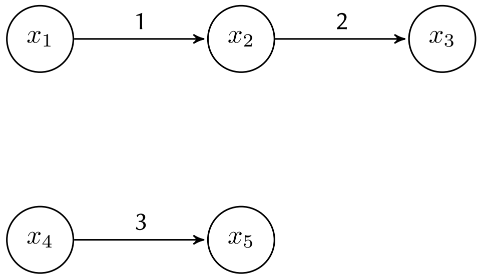


Figure 3: One possible solution.

where X, Y are NODES and I is a STEP.

VALUE(X, I, Y) holds if and only if $x_Y \in \text{VAL}_I(x_X)$. We then encoded the update of VALUE using the recursive definition introduced above.

We used the predicate VALUE to check the final solution of the model at the l-th step. For all $i, j \in \{1, 2, \cdots, n\}$ the following must hold:

$$\text{G}(i, j, 1) \leftrightarrow \text{VALUE}(I, l, J)$$
$$\text{G}(i, j, 0) \leftrightarrow \neg\text{VALUE}(I, l, J)$$

Forcing the model to search for graphs that satisfy the total behaviour imposed by G.

Something similar has been done for the constraint on F, with the only difference that F_i can match the VAL of a NODE at any time. Hence, for all F_i there must exists a $t \leq l$ and a J such that for all $k \in \{1, 2, \cdots, n\}$:

$$\text{F}(i, k, 1) \leftrightarrow \text{VALUE}(J, t, K)$$
$$\text{F}(i, k, 0) \leftrightarrow \neg\text{VALUE}(J, t, K)$$

4.2. Symmetry Breaking

symmetries in this problem play a key role to obtain good running times. Because of the model we proposed, symmetries are strictly related to connected components of the resulting graph. Take as an example the following matrix:

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

The minimum number of CNOT required for this matrix is 3. Using the graph model, one possible solution is the one shown in Figure 3

Nevertheless, the only true constraint that must be satisfied, is that the edge between x_1 and x_2 must have a label smaller than the one of the edge between x_2 and x_3. Hence, the labelling $(x_1, x_2, 1), (x_2, x_3, 3), (x_4, x_5, 2)$ is correct too. This is clearly an example of symmetry that has to be broken. In particular, it concerns edges in different connected components.

In order to break this kind of symmetry, we forced the model to avoid leaving *holes* in the labelling in a single connected components. Suppose we have two connected components C_1 and C_2. And let m_i, M_i be the minimum and the maximum edge label in C_i. Then it must be true that all the labels between m_i and M_i belong to C_i.

This kind of constraints avoid any kind of symmetry between connected components with the same size. In case that C_1 and C_2 contains the same number of edges, let m_1, m_2 be the minimum edge labels of C_1 and C_2, respectively. Let X be the source node of the edge with label m_1 and Y be the source node of the edge with label m_2. Then it must be true that $m_1 < m_2 \rightarrow X < Y$. Coping with symmetries introduced in different connected components clearly reduces the search space for the ASP solver. Nevertheless, symmetries in the same connected component also have to be taken into account.

In the same connected component, we want to find which pairs of edges can be *swapped* without modifying the solution's correctness. Hence, we want to find pairs of edges $\langle k, h\rangle$ such that if we switch label between edge k and edge h the computed $\textsc{value}$ function is unchanged.

To do so, we supposed to generate all the candidate swaps of the form $\langle k, h\rangle$, with $k < h$ to avoid duplicate pairs. Then, we introduced a predicate $\textsc{fake\_value}(I, T, J, K, H)$ which has to be interpreted as a *fake* version of $\textsc{value}$ in case that the swap $\langle k, h\rangle$ was performed. Hence, $\textsc{fake\_value}(I, T, J, K, H)$ holds if and only if $x_J \in \textsc{val}_T(x_I)$ with edges k and h swapped.

The update of such predicate follows the rules of $\textsc{val}$, taking care of swapping source and destination of edges k and h. Clearly we are not interested in computing $\textsc{fake\_value}$ for all the time slots, but just for the t between k and h. In fact, it is easy to see that if the behaviour is preserved in the time range $k, k+1, \cdots, h$, the same will hold also for steps after time h.

A candidate swap $\langle k, h\rangle$ is considered a valid swap if the following holds:

$$\begin{gathered}\forall I, J \in \{1, 2, \cdots, n\}, \forall t \in \{K, K+1, \cdots, H\} \\ \textsc{fake\_value}(I, T, J, K, H) \leftrightarrow \textsc{value}(I, T, J)\end{gathered}$$

Once valid swaps have been identified, we can use them as symmetry breakers.In particular, we split symmetries in 3 cases:

- A valid swap $\langle k, h\rangle$ between edges of the form $(X, Y, K), (X, Z, H)$. Then it must be true that $Y < Z$.
- A valid swap $\langle k, h\rangle$ between edges of the form $(X, V, K), (Y, V, H)$. Then it must be true that $X < Y$.
- A valid swap $\langle k, h\rangle$ between edges of the form $(X, V, K), (Y, W, H)$. Then it must be true that $V < W$.

Roughly speaking, we fixed a unique choice for the swappable edges depending on their sources / destinations.

5. Modelling with a DAG

In the previous model, the graph had no constraint in its structure. Hence, it could contain cycles that makes the detection of swappable edges harder than expected.

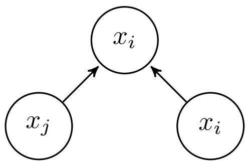


Figure 4: Generic DAG node describing CNOT(x_j, x_i).

On top of that, a graph with exactly one edge per time step, did not exploit the possibility of doing parallel CNOTs. For example, two CNOTs that share neither the control nor the target, can eventually be done in the same time step, since they do not interfere with each other.

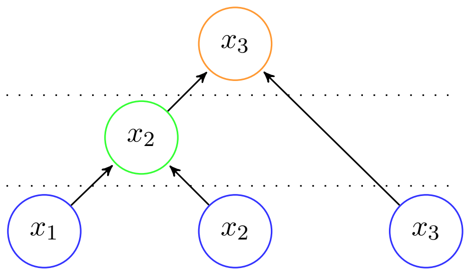
For this reason, we decided to translate our general graph model into a DAG based one. The leaves of the DAG represent the problem variables. Hence, we will have n leaves labelled $x_1, x_2, \cdots, x_n$. A CNOT with control x_j and target x_i is represented by a node x_i with two incoming edges. One edge comes from the closer node labelled x_i. The other from the closer node labelled x_j. The generic structure of a node is depicted in Figure 4.

The DAG structure induces a *layering* of the nodes. Leafs are at layer 0. A node at layer j, can only have incoming edges from nodes in smaller layers. One layer can contain more than a single node. Nevertheless, any layer l can contain at most one node labelled x_i, for any $i \in \{1, 2, \cdots, n\}$.

Example 4. *Consider the matrix*

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

The generated DAG $\mathcal{G}$, with the minimum number of node is the following:



where blue nodes are in layer 0, the green node composes layer 1, and layer 2 is composed by the single orange node. The circuit induced by $\mathcal{G}$ applies CNOT(x_1, x_2) followed by CNOT(x_2, x_3).

As for the case with the model of Section 4, we must ensure that the DAG we build induces a circuit that implements G. Given a DAG $\mathcal{G}$, the circuit it induces can be found through a visit from layer 1 up to layer l exploiting the following rule: each internal node describes a CNOT gate and all the gates at layer i must be executed before the gates at layer $i + 1$, for all $i \in \{1, 2, \cdots, l\}$.

Each layer of this model can be interpreted as the aggregation of more time steps from the general graph model.

In order to ensure that the circuit induced actually implements G, we use the notion of VALUE from the previous section, tweaked for the layered DAG case.

The function VAL is now defined in terms of layer: given a layer t and variable x_i, its value is described by $\textsc{Val}_t(x_i)$. The recursive definition of VAL remains unchanged.

Hence, the correctness of the generated DAG is checked by imposing that in the last layer l the following holds:

$$\textsc{Val}_l(x_i) = G_i$$

5.1. Encoding in CLINGO

We encoded all the above properties in CLINGO. We adopted the NODE type as for the graph encoding. We translated the STEP variables into LAYER variables.

Inner nodes of the graph are described by a predicate of the form XOR_NODE(I, J, L) which has to be interpreted as follows: at layer L, there is a node labelled x_I which is the result of the XOR between x_I and x_J – CNOT(x_j, x_i). Given a variable X and a layer L, we imposed that a single XOR_NODE with target X can exist at layer L. Roughly speaking, XOR_NODE is the equivalent of the EDGE in the graph case.

The predicate VALUE is handled in the same way as for the graph case. The only edit required is to interpret time steps in terms of layers.

Once VALUE is defined, also the final constraints on G and F_i are defined as in case of generic graph encoding.

In this model, we found much less symmetries than in the one from Section 4. The only one we could actually came up with is the following. Suppose XOR_NODE(I, J, L) and XOR_NODE(I, H, K) holds with $L < K$. Moreover, suppose all the variables edited between layer L and layer K are different from x_I, x_H and x_J. Then it is true that CNOT(x_J, x_I) at layer L and CNOT(x_H, x_I) at layer K can be swapped, preserving the circuit overall behaviour.

In case that pairs of nodes of this kind were found in the solution, we forced the model the have the CNOT with the smaller target, at the lower layer. Formally speaking, if XOR_NODE(I, J, L) and XOR_NODE(I, H, K) can be swapped, then it must hold that $x_J < x_H \rightarrow L < K$.

6. Results

We tested the proposed models with the following procedure. For each $n \in \{4, 5, 6, 7, 8\}$ we generated 10 different random tests. For each test we created an $n \times n$ matrix, representing a linear reversible function, that will serve as G – in the quantum case, n is the number of qubits, that evolve through $2^n \times 2^n$ unitary matrices. Moreover, for each test, we picked a number k between 1 and n, that will be the number of different F_i we give as input to each test. We then run the two different models with an iterative procedure that works as follow. Let G, $\{F_i\}$ be the input for the current test case. We initialized a counter l to 1. We then run the graph model with input G, $\{F_i\}$ and l, to see if the problem was solvable in l steps. If true, then we stored the running time and move to next sample. Otherwise, we increased l and queried the model again. Notice that, by Theorem 1, l will at most be n^2.

For what concerns the DAG model, we applied the same procedure but expecting a different result from the model: the DAG model uses l as number of layers, and then it minimizes the number of CNOT within the layers. Hence, when the model is satisfiable, we also have the computing time to get the optimal result for the given l.

In Table 1 we report the obtained results for the graph method, while in Table 2 the results with the DAG approach.

In both tables, the first column is the quantity n, the number of qubits. The second one is the average running time (in seconds) in the 10 tests. For each test, the time we measured is the overall time for the iterative procedure to find the solution, and not only the time for the satisfiable instance — even if we noticed that the unsatisfiability is found in less than 0.01 seconds in most of the cases. The tests have been run in a 2021 MacBookPro, with a 2 GHz Intel Core i5 quad-core CPU and 16GB of LPDDR4X Ram.

n	avg time (seconds)
4	0.524
5	10.140
6	243.938
7	835.358
8	»1000

Table 1
Results with the graph model.

n	avg time (seconds)
4	0.023
5	0.702
6	20.212
7	45.548
8	98.721

Table 2
Results with the DAG model.

We can see from Table 1 and Table 2 that the DAG model is faster than the graph one. The generic graph model becomes useless with sizes bigger than 6. On the other hand, the DAG model is promising even if the running time is doubling every time the size increases. We tried to compare our results with the one presented in [8], but their procedure contains parts the we got for *granted*. For example, they start from an actual circuit, they extract the phase polynomial decomposition and just then try to minimize the circuit. In our case, we focused in the minimization part to see if we could somehow inject our solution into their algorithm. Since they tested their method with 6×6 matrices with running times of order of hundreds of seconds (in average), we are pretty confident that our DAG method would bring a significant speed up in the overall process.

7. Conclusions and Future Works

In this paper we tackled the problem of minimizing the number of CNOT gates in a Quantum Circuits. We solved the problem with two different ASP models. The results we obtained show that the generic graph model is not very useful. On the other hand, the second DAG approach is promising. Further improvements are possible. For example, the search for the first l for which the problem is satisfiable can be done through a binary search instead of trying all possible l's. Deep investigations should be carried out also about the effectiveness of constraint programming in solving the same problem we tackled with ASP.

Despite that, it seems that Answer Set Programming can be fruitfully applied to the synthesis problem of quantum circuits in general, and not only in the CNOT case. Moreover, the solutions to this problem and solutions to problems such as the minimization of the number of T gates, paves down the road to the idea that the synthesis problem is not a monolith. Fast and not optimal techniques may be adopted to obtain a first and rough synthesis of some unitary $\mathcal{U}$. Starting from such synthesis, algorithms can be applied to systematically minimize parts of the circuits to obtain a reduction that is close to the optimal one. In order to benchmark the results of such approach, a great amount of test cases can be found in [17, 18].

References

[1] L. K. Grover, A fast quantum mechanical algorithm for database search, 1996. arXiv:quant-ph/9605043.

[2] D. Miller, M. Thornton, Qmdd: A decision diagram structure for reversible and quantum circuits, in: 36th International Symposium on Multiple-Valued Logic (ISMVL'06), 2006, pp. 30–30. doi:10.1109/ISMVL.2006.35.

[3] P. Niemann, R. Wille, R. Drechsler, Advanced exact synthesis of clifford+t circuits, Quantum Information Processing 19 (2020). doi:10.1007/s11128-020-02816-0.

[4] P. Niemann, R. Wille, R. Drechsler, Efficient synthesis of quantum circuits implementing clifford group operations, in: 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), 2014, pp. 483–488. doi:10.1109/ASPDAC.2014.6742938.

[5] C. Ruican, M. Udrescu, L. Prodan, M. Vladutiu, A genetic algorithm framework applied to quantum circuit synthesis, in: Nature Inspired Cooperative Strategies for Optimization, 2007.

[6] Y.-H. Chou, S.-Y. Kuo, Y.-C. Jiang, C.-H. Wu, J.-Y. Shen, C.-Y. Hua, P.-S. Huang, Y.-T. Lai, Y. F. Tong, M.-H. Chang, A novel quantum-inspired evolutionary computation-based quantum circuit synthesis for various universal gate libraries, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 2182–2189. doi:10.1145/3520304.3533956.

[7] M. Amy, D. Maslov, M. 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 (2014) 1476–1489. doi:10.1109/tcad.2014.2341953.

[8] G. Meuli, M. Soeken, G. D. Micheli, Sat-based {CNOT, T} quantum circuit synthesis, in: International Workshop on Reversible Computation, 2018.

[9] J. Roffe, Quantum error correction: an introductory guide, Contemporary Physics 60 (2019) 226–245. doi:10.1080/00107514.2019.1667078.

[10] W. Faber, An Introduction to Answer Set Programming and Some of Its Extensions, Springer International Publishing, Cham, 2020, pp. 149–185. doi:10.1007/978-3-030-60067-9_6.

[11] M. Amy, D. Maslov, M. Mosca, M. Roetteler, A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2013).

[12] M. Amy, D. Maslov, M. Mosca, M. Roetteler, A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 32 (2013) 818–830. doi:10.1109/tcad.2013.2244643.

[13] M. Bataille, Quantum circuits of CNOT gates: optimization and entanglement, Quantum Information Processing 21 (2022) 269. doi:10.1007/s11128-022-03577-8.

[14] V. Gheorghiu, J. Huang, S. M. Li, M. Mosca, P. Mukhopadhyay, Reducing the cnot count for clifford+t circuits on nisq architectures, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2022) 1–1. doi:10.1109/tcad.2022.3213210.

[15] M. Bataille, Quantum circuits of cnot gates, 2020. arXiv:2009.13247.

[16] W. Haaswijk, A. Mishchenko, M. Soeken, G. De Micheli, Sat based exact synthesis using dag topology families, in: Proceedings of the 55th Annual Design Automation Conference, DAC '18, Association for Computing Machinery, New York, NY, USA, 2018. doi:10.1145/3195970.3196111.

[17] T. Tomesh, P. Gokhale, V. Omole, G. S. Ravi, K. N. Smith, J. Viszlai, X.-C. Wu, N. Hardavellas, M. R. Martonosi, F. T. Chong, Supermarq: A scalable quantum benchmark suite, 2022. arXiv:2202.11045.

[18] A. Li, S. Stein, S. Krishnamoorthy, J. Ang, Qasmbench: A low-level qasm benchmark suite for nisq evaluation and simulation, 2022. arXiv:2005.13018.