université PARIS-SACLAY

Methods for optimizing the synthesis of quantum circuits

Thèse de doctorat de l'Université Paris-Saclay

École doctorale n° 580, Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat : Informatique
Unité de recherche : Université Paris-Saclay, CNRS, Laboratoire de recherche en informatique, 91405, Orsay, France
Référent : Faculté des sciences d'Orsay

Thèse présentée et soutenue à Gif-sur-Yvette, le 22 Octobre 2020, par

Timothée GOUBAULT DE BRUGIÈRE

Composition du jury :

Pascale Senellart Directrice de Recherche CNRS, C2N/Université Paris-Saclay	Présidente
Mark Asch Professeur des Universités, LAMFA/Université de Picardie Jules Verne	Rapporteur & Examinateur
Michele Mosca Professeur des Universités, IQC/University of Waterloo	Rapporteur & Examinateur
Elham Kashefi Directrice de Recherche CNRS, LIP6/Sorbonne Université	Examinatrice
Cécilia Lancien Chargée de Recherche CNRS, IMT/Université Toulouse 1 Capitole	Examinatrice
Marc Baboulin Professeur des Universités, LRI/Université Paris-Saclay	Directeur de thèse
Benoît Valiron Maître de Conférences, LRI/CentraleSupélec	Co-encadrant de thèse
Cyril Allouche Responsable Quantum R&D, ATOS	Invité

Thèse de doctorat

NNT: 2020UPASG018

Contents

List of Figures

List of Tables

List of Algorithms

Part A

Introduction to Quantum Circuit Synthesis

Chapter 1

Foreword

A possible definition of computer science is that of the field of activity about the encoding and automatic processing of information. The interest of this definition is that it does not place the modern computer as an unsurpassable pinnacle. On the contrary, it puts on the same level any automaton capable of processing information, in any form whatsoever. Whether electronic or mechanical, based on electricity or water, complex or rudimentary, a computer is only a calculating machine. Based on that it is easier to consider quantum computing as only a different way of encoding and processing information. It is not an extension of "classical" computing but a new paradigm, with its new rules governed by quantum mechanics. In quantum computing, the unit of information is the quantum bit, also called a *qubit*, and the information processing operations are *quantum operators*.

In the early 1980s, Richard Feynman and Yuri Manin conceptualized the idea of using a quantum system to process information. Their purpose was to simulate quantum systems that are intractable for classical computers [59, 128]. Until the mid-90s, quantum computing remained a promising theoretical paradigm capable of surpassing a classical computer on contrived problems but with few experimental results or concrete applications. Then the first practical algorithms outclassing the classical computer were found: factorization of integers [178] and simulations of systems [123] with an exponential speedup, research in an unstructured database [71] with a quadratic speedup. The three aforementioned articles have contributed to the development of the field. This is particularly noticeable if we look at the number of papers on quantum computing published every year in Fig 1.1.

Since then, quantum computing has developed in a similar way to classical computing with the creation of compilers [80, 188], the development of programming languages [89, 186, 192], error correction methods [62, 68, 111, 182], the improvement of the hardware [16, 162, 168, 170], the studies of new classes of complexity [1, 21], etc. With more than 70 years of classical computing behind us, the roadmap for the development of quantum computing seems drawn. Yet everything remains to be invented because the rules which govern quantum computing are fundamentally different from classical computing. We can cite the *no-cloning* theorem which prevents duplicating information [207], or the impossibility of "probing" the state of the system during the current calculation otherwise you destructively interfere with the quantum memory. This singularity specific to quantum computing leads for example to consider new complexity classes like the BQP class [21] — where BQP stands for Bounded-error Quantum Polynomial time, the class of problems solvable in polynomial time by a quantum computer — whose inclusion rules are not yet fully known with other classic complexity classes like NP.

In this thesis we focus on the *compilation* problem, i.e., the translation of a high-level algorithm

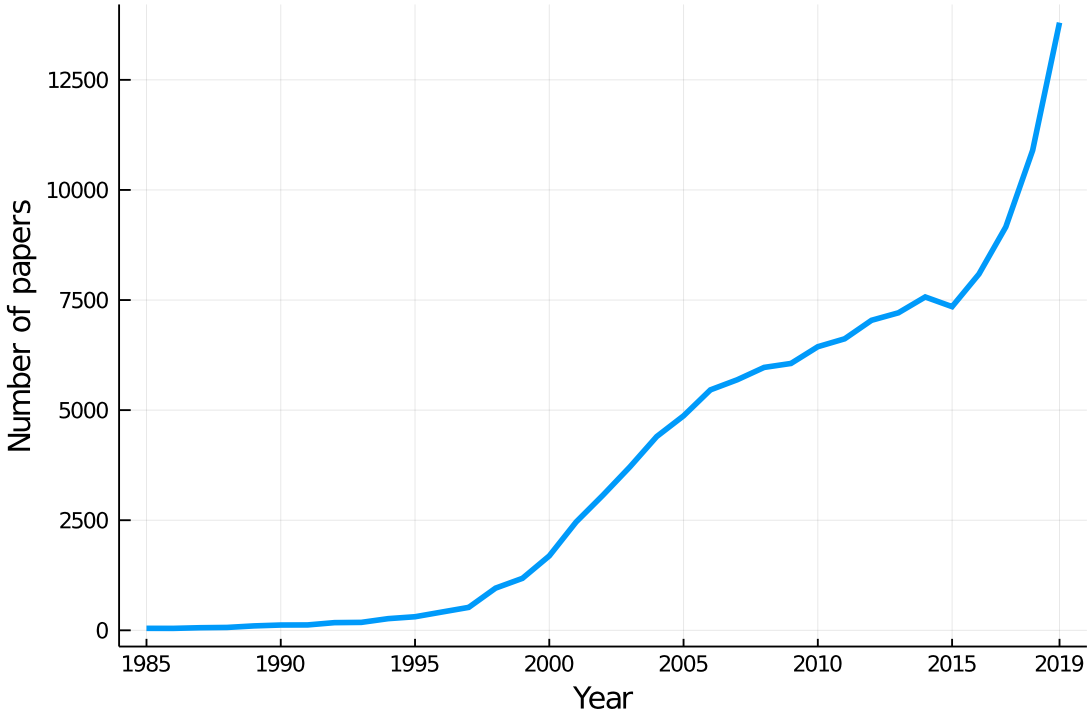


Figure 1.1: Number of papers referenced in Google Scholar using the term "quantum computing" per year.

into a sequence of instructions executable on a given quantum machine, a *quantum circuit*, which is the equivalent of the assembly language on a classical computer. This compilation step is crucial because it determines the quantity of quantum resources necessary for the execution of the algorithm. Yet, quantum resources are currently extremely limited despite the constant progress of experimenters to build a large-scale quantum computer. The number of qubits, the computing time on the quantum machine, the number of instructions, are all precious resources that should be used optimally. Therefore, it is necessary to optimize the compilation process as much as possible to allow the use of practical quantum algorithms despite the hardware constraints. A quantum processor is likely to have the same functionality than other computational modules like GPUs (Graphics Processing Unit) or FPGAs (Field-Programmable Gate Array): it will be a computational core attached to a classical machine — probably via some cloud computing — to speed up some specific tasks. Thus, in the same way that GPUs or FPGAs are very efficient computational modules but with limited resources (memory for a GPU for example), a quantum processor will be able to perform certain tasks much faster but with limited resources that will have to be optimized.

This thesis is part of the quantum computing field but is not reserved to scientists initiated in the domain. For example, it does not suppose any particular understanding of quantum mechanics and its unintuitive mathematical formalism. The laws of quantum mechanics dictate the rules of operation of a quantum computer but, once these rules are accepted, the content of this thesis falls within the fields of optimization, high-performance computing, but ultimately little of what one could imagine as “quantum”. The same goes for the physical details of building a quantum computer: they will impose constraints on our problems but they will impose themselves as axioms rather than necessary steps in reasoning. Chapter 2 of this introductory part will introduce these axioms of the functioning of the quantum computer and will mathematically define the problems

that have been addressed during this Ph.D. thesis.

The manuscript is divided into four parts. Part A introduces the main notions in quantum circuit synthesis and our contributions. The work of this Ph.D. thesis is then developed in Parts B and C, each of them corresponding to the compilation of a quantum circuit from a precise representation of the quantum algorithm. In Part B we focus on algorithms given in the form of a so-called *unitary matrix*. Any algorithm has such a matrix representation and, moreover, this representation is unique. During the compilation, one essential problem is to minimize the quantum resources necessary for the execution of the quantum algorithm. We add the question of the minimization of the classical resources and what tradeoff is possible between these two types of resources. Part C deals with the compilation of a subclass of quantum algorithms, applying so-called *linear reversible operators*. In this part, it will only be a question of minimizing the quantum resources. Part D concludes the manuscript and proposes possible research tracks.

Chapter 2

Background and Contributions

Contents

2.1 From Quantum Mechanics to Quantum Computation

In quantum computing, information is encoded on the quantum state of elementary particles. The basic unit of information in classical computing, the *bit*, is therefore replaced in quantum computing by its quantum version: the *quantum bit*, or *qubit*. For instance, the spin of a photon can be "up" or "down" and encode an elementary 2-level system with values "true" and "false". This system obeys the laws of quantum mechanics and a quantum computer exploits these laws to perform computation. Thereby, by the quantum superposition principle, the value of the spin can be a linear combination of "up" and "down". Any classical operation that would usually take as input either the value true or false will be done “on both values simultaneously” with a quantum computer.

Several models of quantum computation coexist: quantum circuit model [210], adiabatic computation [58], measurement-based [33, 160], *etc.*, and for each model there are several possible technologies for the physical realization of a quantum computer. For example, in the quantum circuit model, we can mention the technologies based on superconducting qubits [16], trapped-ions [74], linear optical [112], etc. Each model corresponds to a specific formalism of computation, with some variations among the different technologies in one model. In this Ph.D. thesis, we focus on the quantum circuit model for quantum computing. Indeed, this model is the dominant model used in, e.g., the description of most quantum algorithms. The purpose of this chapter is to present the tools that formalize this model and alleviate with getting into the details of the physical implementation... up to a certain point. Within the quantum circuit model, we will take a closer look at some candidate technologies and take into account the specificities of the hardware.

We start this chapter by presenting the basic notions about quantum computation and the quantum circuit model:

- In Section 2.2 we first present the way quantum states are represented and how they are modified by operators.
- In Section 2.3 we present the quantum circuit model.

Then we will focus on the main theme of the thesis, namely the synthesis of quantum circuits:

- In Section 2.4 we formally define the quantum circuit synthesis problem.
- In Section 2.5 we present the main results in the state-of-the-art regarding quantum circuits synthesis.
- In Section 2.6 we finally present the different problems we address during this thesis.

Section 2.7 specifies some notations that will be used throughout this manuscript.

2.2 Quantum States and Quantum Operators

2.2.1 Quantum States

In classical computation, the value of a bit is binary: true or false, 1 or 0. In quantum computation, the value of a qubit is a linear, complex, and normalized superposition of true and false. We use the Dirac notation and we write

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \; ; \; |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

to represent the basis states of computation. Then any qubit is in a *quantum state*

$$|x\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \in \mathbb{C}^2$$

with $|\alpha|^2 + |\beta|^2 = 1$. In other words, we identify the states of a qubit with the normalized complex vectors of size 2 modulo a global phase $e^{i\theta}$ for some real angle θ.

Similarly, the state $|x\rangle$ of an n-qubit system is a linear, complex, and normalized superposition of the 2^n classical states of an n-bit system. We write

$$|x\rangle = \sum_{x_1,\ldots,x_n \in \{0,1\}} \alpha_{x_1 x_2 \ldots x_n} \, |x_1 x_2 \ldots x_n\rangle \; ; \sum_{x_1,\ldots,x_n \in \{0,1\}} |\alpha_{x_1 x_2 \ldots x_n}|^2 = 1$$

where $x_1x_2...x_n$ stands for the string of bits made of the concatenation of $x_1, ..., x_n$. Again we identify the states of an n-qubit system with the normalized vectors of size 2^n. The usual ordering of the basis states corresponds to the lexicographic order. For example, in the case of two qubits, the basis states are

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \ |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \ |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \ |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Given one n-qubit system in a state $|\psi\rangle$ and one m-qubit system in a state $|\varphi\rangle$, the $n+m$-qubit system given by the combination of the two systems is in the state $|\psi\rangle \otimes |\varphi\rangle$ where $\otimes$ is the *Kronecker product*, or *tensor product*, and is defined by

$$|\psi\rangle \otimes |\varphi\rangle = \begin{pmatrix} \psi_1 |\varphi\rangle \\ \psi_2 |\varphi\rangle \\ \vdots \\ \psi_{2^n} |\varphi\rangle \end{pmatrix} \in \mathbb{C}^{2^{n+m}}.$$

One can check that the basis states of the n-qubit systems are the tensor product of the basis states of n 1-qubit systems combined together. In our example with two qubits, the basis states are

$$|00\rangle = |0\rangle \otimes |0\rangle, \ |01\rangle = |0\rangle \otimes |1\rangle, \ |10\rangle = |1\rangle \otimes |0\rangle, \ |11\rangle = |1\rangle \otimes |1\rangle.$$

We simplify the notations by identifying a classical state on n qubits with its integer representation. We write indifferently

$$|x\rangle = \sum_{x_1,...,x_n \in \{0,1\}} \alpha_{x_1x_2...x_n} |x_1x_2...x_n\rangle$$

or

$$|x\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle$$

depending on the context.

If two independent systems can always be combined into one global state using the tensor product — the state is then said to be *separable* —, it is not always possible to separate the state of a combined system into two independent states. When a quantum state cannot be written as the tensor product of two smaller states the state is said to be *entangled*. The Bell states are simple examples of entangled states on two qubits. One of them is defined by

$$|\Phi\rangle = \frac{1}{\sqrt{2}} (|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle)$$

and one can check that it cannot be expressed as the tensor product of two one-qubit states. Entanglement is believed to be a key aspect in the quantum supremacy over classical computation [92] and research is still ongoing to better understand its role, for example by giving a measure of how entangled a state is [198].

2.2.2 Quantum Operators

Given a quantum state $|\phi\rangle$ on n qubits, the allowed transformations one can perform on $|\phi\rangle$ can be derived from the Schrödinger equation. They are of the form

$$|\phi_{t+1}\rangle = e^{iH} |\phi_t\rangle$$

with i the standard unit imaginary number and H a Hermitian matrix, i.e., it satisfies the relation $H = H^\dagger$ where $\dagger$ stands for the conjugate transpose operator. The set

$$\{e^{iH} \mid H \in \mathbb{C}^{2^n}, H = H^\dagger\}$$

is mathematically equivalent to the set of unitary matrices

$$\mathcal{U}(2^n) = \{U \in \mathbb{C}^{2^n} \mid UU^\dagger = I\}.$$

Therefore the evolution of a quantum state is driven by left-multiplications with unitary matrices.

Three essential properties of quantum computation are direct consequences of how quantum states evolve:

1. The sequential application of two operators A and B on $|\phi\rangle$ yields the state $B(A|\phi\rangle) = (BA)|\phi\rangle$ and is equivalent to the application of the operator BA. Sequential application corresponds to matrix multiplication.

2. To combine operators acting on distinct subsystems, we again use the tensor product. If $|\psi\rangle$ is an $(n+m)$-qubit state and one applies an operator A on the first n qubits (resp. B on the last m qubits) then using the global system on $n+m$ qubits it is equivalent to applying the operator $A \otimes B$ on the state $|\psi\rangle$ with

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1p}B \\ a_{21}B & a_{22}B & \dots & a_{2p}B \\ \vdots & \vdots & & \vdots \\ a_{q1}B & a_{q2}B & \dots & a_{qp}B \end{pmatrix} \in \mathcal{U}(2^{n+m})$$

   for any $A \in \mathcal{U}(2^n), B \in \mathcal{U}(2^m)$.

3. Quantum computation is *reversible*. Namely, if one performs

$$|\phi_2\rangle = A|\phi\rangle$$

   then one can recover the state $|\phi\rangle$ by applying the inverse operator

$$|\phi\rangle = A^\dagger |\phi_2\rangle .$$

   Therefore classical reversible computation is a subset of quantum computation.

Besides sequential composition, tensoring, and inverse, a fourth operation is usually considered: an operator can be *controlled*, meaning that the operator is applied on so-called *target qubits* depending on the logical value of a qubit so-called *controlled qubit*. In other words, if $M \in \mathcal{U}(2^n)$ is an operator acting on n qubits controlled by an additional qubit given without loss of generality as the most significant qubit, there are two canonical operations on $n+1$ qubits: the positively-controlled-M defined as the block matrix $\left(\begin{smallmatrix} I & 0 \\ 0 & M \end{smallmatrix}\right)$ and the negatively-controlled-M, defined as $\left(\begin{smallmatrix} M & 0 \\ 0 & I \end{smallmatrix}\right)$. Both block-matrices are operators in $\mathcal{U}(2^{n+1})$. The former sends $|0\rangle \otimes |\phi\rangle$ to $|0\rangle \otimes |\phi\rangle$ and $|1\rangle \otimes |\phi\rangle$ to $|1\rangle \otimes (M|\phi\rangle)$. The latter does the opposite: it sends $|0\rangle \otimes |\phi\rangle$ to $|0\rangle \otimes (M|\phi\rangle)$ and $|1\rangle \otimes |\phi\rangle$ to $|1\rangle \otimes |\phi\rangle$. This generalizes to multiple controls where an operator is applied depending on the logical values of several control qubits. This also generalizes to the so-called *quantum multiplexors* where a specific operator is applied for each logical value of the control qubits. So, for example, an operator A positively-controlled by p control qubits is a multiplexor where the identity operator is applied for all values of the p control qubits except $|1\rangle^{\otimes p}$ where A is applied. More information about quantum multiplexors are given in Section 4.1.1.

2.3 Quantum Gates and Quantum Circuits

2.3.1 Quantum Gates

Basically, any quantum algorithm takes as input a quantum state, applies a specific unitary operation, and performs some measurements on the output state. However it is impossible to perform an arbitrary unitary operation given the current technology, i.e., the number of different instructions the hardware can perform is limited. In the quantum circuit model, a given technology — related to the hardware used for its physical implementation — provides only a set of *elementary operations* that can be done natively. Generally acting on one or two qubits, these elementary operators are called *quantum gates*, and we can mention the following (see Table 2.1):

- the Pauli operators X, Y, Z (the X gate is equivalent to the classical NOT gate),
- the Hadamard gate H which enables us to transform a pure state ($|0\rangle$ or $|1\rangle$) into an equal superposition of $|0\rangle$ and $|1\rangle$, i.e.,

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad ; \quad H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle),$$

- the continuous set of elementary rotations R_X, R_Y, R_Z defined by

$$R_G(\alpha) = \cos(\alpha/2)I_2 - i\ \sin(\alpha/2)G \text{ with } G \in \{X, Y, Z\}$$

  where X, Y, Z are the Pauli operators and i is the unit imaginary number. We can also write R_x, R_y, R_z with lowercase subscripts: it is this notation we will use throughout this Ph.D. thesis.

- the continuous set of phase gates defined by

$$Ph(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$

  adding a phase to the state $|1\rangle$; among this set two gates are of particular use: the gate T ($\theta = \pi/4$) and the gate S ($\theta = \pi/2$). Note that $Ph(\theta)$ is simply $R_z(\theta)$ modulo a global phase $e^{-i\frac{\theta}{2}}$.

Amongst the frequently used 2-qubit gates, one can name the CNOT-gate, which is the positively-controlled X-gate, and the SWAP gate, flipping the state of two qubits. Other examples of commonly encountered gates are controlled-rotations with arbitrary angles and the Mølmer–Sørensen (MS) gate used in trapped-ions based quantum computers defined by:

$$\text{MS}(\theta) = e^{-i\theta(\sum_{i=1}^{n}\sigma_x^i)^2/4}$$

where σ_x^i is the operator X applied to the i-th qubit.

2.3.2 Quantum Circuits

Given a set of elementary quantum gates, more complex operations can be computed by exploiting the laws of composition and combination. The usual graphical language for representing

$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
X	Y	Z	CNOT	SWAP
$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$		
H	S	T		

Table 2.1: Usual elementary unitary matrices for representing quantum gates.

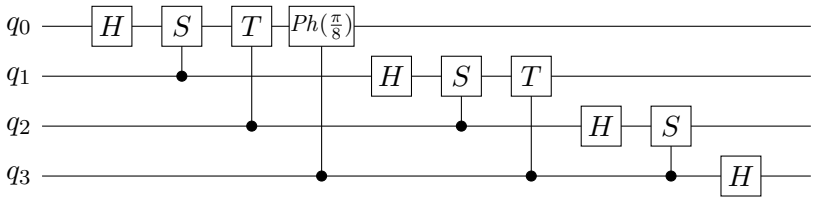


Figure 2.1: Quantum circuit for the Quantum Fourier Transform.

composition and combination of operators is the equivalent of the Boolean circuit for classical computing: the *quantum circuit*. A quantum circuit consists of a series of parallel, horizontal wires on which are attached boxes. Each wire corresponds to a qubit, and vertical combination corresponds to the Kronecker (tensor) product. The circuit is read from left to right and each box corresponds to a quantum gate (i.e., a unitary operator) applied to the corresponding qubits. Controlled-gates have a special representation: the controlling qubit is represented with a bullet (•) if the control is positive and a circle (○) if the control is negative. A vertical line then connects the controlling qubit to the gate to be controlled. The notation easily extends to multiple controls. As an example of a quantum circuit combining several gates, the so-called Quantum Fourier Transform on 4 qubits [148] is represented in Figure 2.1. It enables us to visualize the use of elementary gates: H, S, T, phase-gate, and positive controls. Given a quantum circuit C, we write U_C the associated quantum operator implemented by C. If a quantum circuit represents a single quantum operator, one operator can be synthesized with several quantum circuits. In other words, for two circuits C and C' we may have $U_C = U_{C'}$. In this case, we say that C and C' are functionally equivalent because they result in the same action on the quantum system. A trivial example is given in Fig 2.2 where one can check that two successive Hadamard gates are equivalent to the empty circuit. Therefore, among a set of equivalent circuits, it is possible to search for a circuit that "is better" than the others. This is the central theme of this thesis and more details are given in Section 2.4.

2.3.3 Universality

Given a set of gates $\mathcal{S}$, we write $\Im(\mathcal{S})$ the set of quantum operators that can be implemented by a quantum circuit containing solely gates from $\mathcal{S}$. We say that a set of gates is *universal* if any

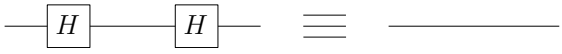


Figure 2.2: Two equivalent circuits. The corresponding matrix equation is $HH = I_2$.

quantum operator, acting on any number of qubits, can be implemented as a sequence of gates from this set (see e.g. [148, Sec. 4.5] for a complete discussion on the matter). For instance, a fundamental (theoretical) result claims that it is possible to realize any operator only with the set of one-qubit gates and one "sufficiently entangling" multi-qubit gate. To be able to implement any quantum algorithm with a given piece of hardware, it is therefore necessary to first find a universal set of technologically implementable gates. The native universal set of gates is different depending on the technology used. We now review some of the gate sets we will consider in this Ph.D. thesis, some being universal, others not.

CNOT + $\mathcal{SU}(2)$

This universal gate set [32] is probably the most common one in quantum compilation. Hence,

$$\Im(\{CNOT, \mathcal{SU}(2)\}) = \bigcup_{n\geq 1} \mathcal{U}(2^n).$$

It is used, e.g., in superconducting quantum computers (IBM [7], Google [16], Rigetti [162] *etc.*) and in linear quantum optics [112].

The { CNOT + $\mathcal{SU}(2)$ } gate set will be used in Chapters 4 and 5.

Clifford or {H, CNOT, S, Pauli}

The Clifford gate set is composed of the Hadamard gate, the CNOT gate, the phase gate S and the Pauli operators. This gate set is known to be efficiently simulable by a quantum computer [69] and therefore is not universal. Any operator that can be implemented with the Clifford gate set is said to be a Clifford operator. More details about this group are given later in this chapter, see Section 2.5.2.

Part C of this Ph.D. thesis will focus on a subclass of the Clifford operators: the linear reversible operators, i.e., operators generated by purely CNOT circuits.

Clifford + T

For fault-tolerant quantum computation, a discrete version of the { CNOT + $\mathcal{SU}(2)$ } gate set is usually considered: the { CNOT, H, T } gate set or Clifford+T gate set. Indeed, up to a global phase, any one-qubit gate can be approximated at any desired precision with only H and T gates [103] (see Section 2.5.4 for more details). The gates from the set { CNOT, H, T } can be implemented fault-tolerantly in many error-correcting protocols, notably the magic state distillation routines [29, 30].

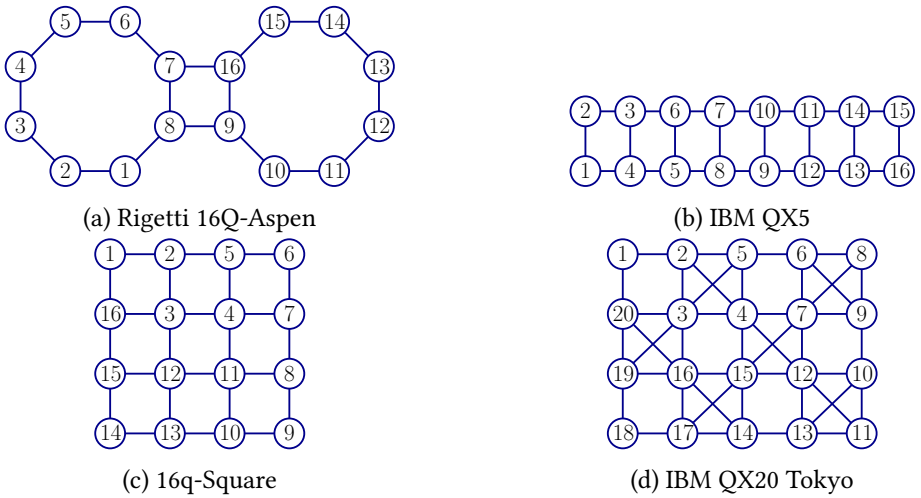
Our works in Part C have some applications in the optimization of Clifford+T quantum circuits, see Section 12.4 for more details.

MS + R_x + R_z

A technology using trapped ions has other gates available like the Mølmer–Sørensen gate (MS gate) [129, 170] defined by

$$MS(\theta) = e^{-i\theta(\sum_{i=1}^{n} \sigma_x^i)^2/4}.$$

where σ_x^i is the operator X applied to the i-th qubit and θ is an arbitrary angle. The MS gate is a global gate, acting on all qubits. The available one-qubit gates are the local R_z rotations and global



(a) Rigetti 16Q-Aspen

(b) IBM QX5

(c) 16q-Square

(d) IBM QX20 Tokyo

Figure 2.3: Qubit connectivity graphs from existing architectures.

R_x rotations, i.e., an R_x gate is simultaneously applied to every qubit with the same angle. Any one-qubit gate U can be expressed as

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta)$$

where $\alpha, \beta, \gamma, \delta$ are the so-called Euler angles of U in the ZYZ basis [148]. As

$$R_y(\gamma) = R_x(-\pi/2) \times R_z(\gamma) \times R_x(\pi/2)$$

one can see that any one-qubit gate can be implemented with the available gates for trapped-ions quantum computers. Finally, any CNOT gate can be synthesized with two MS gates and local one-qubit gates [133]. Hence the set of gates for trapped-ions quantum circuits is also universal.

The native gate set for trapped-ions quantum computers will be used in Chapter 5.

2.3.4 Hardware Constraints

The hardware technology used for the implementation of a quantum computer determines not only the kind of elementary gates that one can apply but also the possible interactions that can be performed between the qubits. The physical position of the qubits in the hardware plays a role and limits the interactions between pairs of qubits. In some architectures, the qubits can only interact with their direct neighbors. In other words, the 2-qubit gates can only be performed between qubits that are neighbors in the hardware. This is the case for superconducting technologies where full connectivity between the qubits cannot be achieved. The connections between the qubits are usually 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. In our setting we include the graph into the set $\mathcal{S}$. Provided that the graph is connected, the universality of $\mathcal{S}$ is preserved. Examples of connectivity graphs from current physical architectures are given in Fig 2.3. Note that the architectures presented in Fig 2.3 all rely on superconducting qubits with a specific notion of locality. With some processors based on neutral atoms, the qubits have an interaction radius and their interaction area is given by spheres in 2D or 3D [79].

2.4 Synthesis and Optimization of Quantum Circuits

Having an implementable universal set of gates is not enough: if we are given a quantum operator as a unitary matrix, or any other abstract representation non-understandable by the hardware, one also has to find a way to turn the desired operator acting on a potentially large number of qubits into a quantum circuit made of local, elementary gates. This problem is known as *circuit synthesis*, or equivalently, the *compilation* of the operator into a circuit. The synthesis can be *exact* or *approximate* with a certain precision ϵ, i.e., given an operator U to synthesize we authorize ourselves any circuit C such that $\|U_C - U\| \leq \epsilon$. We also define the *state preparation* problem as a special case of circuit synthesis where we want to synthesize only the first column of a unitary matrix. This corresponds to the case where we have the basis state $|00..0\rangle = |0\rangle^{\otimes n}$ as input and we want the state $|\psi\rangle$ as output, i.e., we want to implement an operator U such that $U|0\rangle^{\otimes n} = |\psi\rangle$. State preparation is a less constrained synthesis problem and more efficient solutions can be computed.

Many different circuits can synthesize a given operator. This means that we can choose, or at least search for the best circuit that minimizes a precise criterion. We evaluate this way the quality of the synthesis with the properties of the computed circuit. Two criteria are widely considered:

- The *size* of the circuit given by the number of elementary gates. Each gate needs computational resources to be executed and one cannot avoid some noise in the physical realization of the gate. This results in interferences that can alter the output of the quantum algorithm and potentially return a wrong result. The shorter the circuit generated is, the less noisy will be the execution of the algorithm.

- The *depth* of the circuit represents the number of time-steps needed to execute the circuit provided that two non-overlapping gates can be executed simultaneously. The depth of the circuit is closely related to the time needed for the execution of the quantum circuit. A major problem faced by the physicists when designing a quantum computer is the *decoherence time*. This time gives the maximum time available to execute a quantum algorithm before the qubits involved in the computation interact enough with the outside environment to lose all the information they carry. A shallow circuit is of interest to ensure that the quantum algorithm will be able to terminate.

The synthesis can also be done with the help of extra quantum memory, i.e., with the use of *ancillary* qubits. There are generally two types of ancillae:

- the *clean* ancillae whose initial values are known, generally $|0\rangle$, and, after the execution of the quantum circuit, the ancillae have to return to their initial values.

- the dirty ancillae whose initial values are unknown and there is no condition on their output values.

Ancillary qubits are useful in many quantum compilation procedures and often help with the parallelization of the circuit.

Overall we give the following definition of the quantum circuit synthesis problem:

Quantum circuit synthesis problem

Input
- An abstract representation U of an n-qubit quantum operator,
- A set of gates $\mathcal{S}$ such that $U \in \Im(\mathcal{S})$,
- $\epsilon > 0$,
- m ancillary qubits,
- $t > 0$,
- a cost function f and $c > 0$.

Problem Find a quantum circuit C such that:
- C contains only gates from $\mathcal{S}$,
- C acts on $n + m$ qubits with m ancillary qubits,
- $\|U - U_C\| < \epsilon$ for a given norm $\|\cdot\|$,
- $f(C) < c$,
- the classical time to compute C, $T(U_C)$, does not exceed t.

This rather general definition makes it possible to encode many problems of the literature as instances of the quantum circuit synthesis problem. A variety of examples are given Table 2.2. Now a few remarks about the quantum circuit synthesis problem:

- The cost function f can embed any metric of importance to evaluate the quality of the circuit solution. It can be either the size of the circuit or its depth or any custom function like the total noise or the execution time. For simplicity, we assume that f outputs a real value but one can imagine a vector of costs with a multi-objective optimization problem.
- One can expect different algorithms to solve the quantum circuit synthesis problem depending on the available universal set of gates. The multiplicity of candidates for the physical implementation of a scalable quantum computer represents, in fact, a challenging task in quantum circuit synthesis. On the other hand, looking for efficient algorithms for any possible hardware will give insights on what technology should be preferred depending on its "compilation friendly" features.
- Any algorithm has a range of validity, for instance in some cases it cannot return a circuit with a sufficiently good approximation error or the classical time to compute the circuit is too long. Or else, some methods may perform better on specific operators but will perform poorly on other cases. Besides proposing new algorithms for the synthesis of quantum circuits, an essential work consists in determining the range of validity of the synthesis methods.

Circuit synthesis should not be confused with *circuit optimization*. Circuit optimization is closely related to circuit synthesis but, in this case, the input of the problem is already a quantum circuit. The goal is to return an optimized circuit — a shorter one, a shallower one, *etc.* — that is functionally equivalent. Despite the difference between the two problems, it is still possible to establish a link between the two with the help of the *classical simulation* of a quantum circuit. The classical simulation of a quantum circuit C consists in computing the output state $U_C |0\rangle^{\otimes n}$ with a classical computer. Doing so for every classical input state and one can recover the whole unitary U_C on which a synthesis algorithm can be applied. The other way around, one can still use circuit optimization methods to further optimize a circuit produced by a synthesis algorithm. We illustrate the relationships between circuit simulation, optimization, and synthesis in Fig 2.4.

Table 2.2: Examples of quantum circuits synthesis problems. Each problem is associated to a specific abstract representation of the quantum operator, the set of gates with which we want to implement the operator, the desired error and the metric to optimize. By the generic term "circuit size", we mean that there is not a particular metric to optimize except the number of gates in the circuit.

	Representation	Set of gates	Error	Cost function
Generic circuits synthesis	$U \in \mathcal{SU}(2^n)$	CNOT + $\mathcal{SU}(2)$	$\epsilon = 0$	Circuit size
State preparation	$\|\psi\rangle \in \mathbb{C}^{2^n}$	CNOT + $\mathcal{SU}(2)$	$\epsilon = 0$	Circuit size
Diagonal synthesis	$D = \text{diag}(e^{i\theta_1}, \ldots, e^{i\theta_{2^n}})$	CNOT + $\mathcal{SU}(2)$	$\epsilon = 0$	Circuit size
(H,T) synthesis	$U \in \mathcal{SU}(2)$	{H,T}	Arbitrary	T-count
Multiqubit (H,T) synthesis	$U \in \mathcal{SU}(2^n)$	{CNOT,H,T}	Arbitrary	T-count
Hamiltonian simulation	$U = e^{iHt}$	Arbitrary	Arbitrary	Circuit size
CNOT circuit synthesis	$A \in F_2^{n \times n}$	{CNOT}	$\epsilon = 0$	CNOT count
Clifford synthesis	$A \in Sp(2n, F)$	{CNOT, H, S}	$\epsilon = 0$	Circuit size
Phase polynomial synthesis	$U\|x\rangle = e^{i\frac{\pi}{4}\sum_{c,f} c \cdot f(x)}\|g(x)\rangle$	{CNOT, T}	$\epsilon = 0$	T-count/CNOT count
Oracle synthesis	$O\|x\rangle\|y\rangle = \|x\rangle\|y \oplus f(x)\rangle$	{Toffoli, CNOT, X}	$\epsilon = 0$	Circuit size
Reversible synthesis	$P \in S_{2^n}$	{Toffoli, CNOT, X}	$\epsilon = 0$	Circuit size

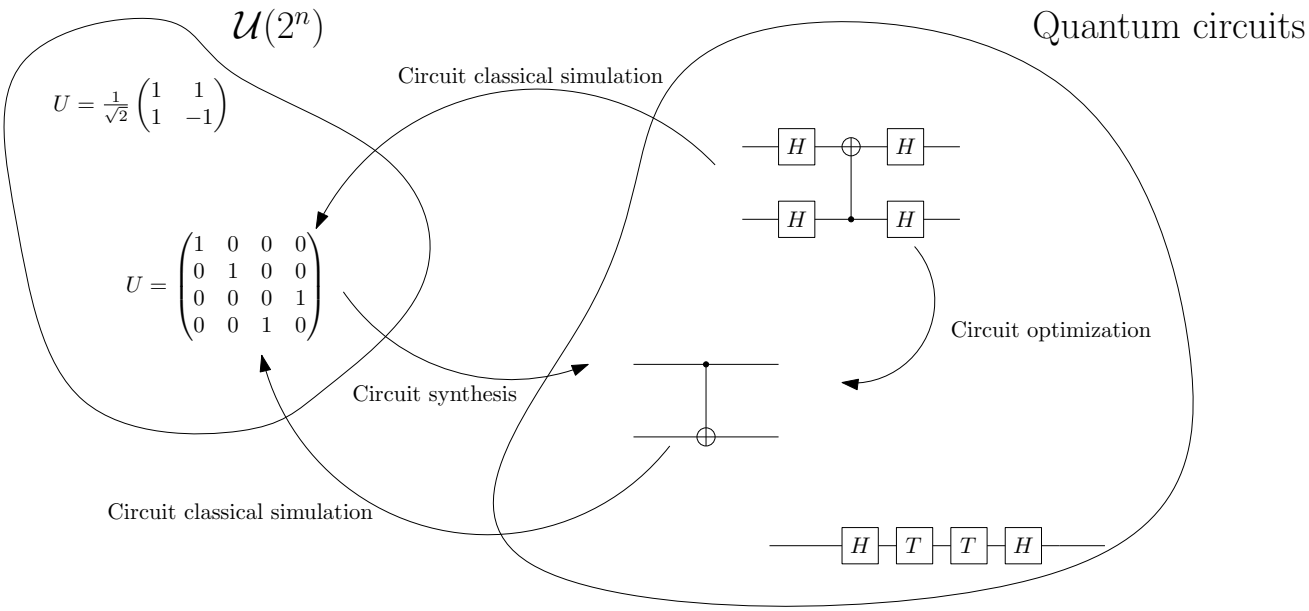


Figure 2.4: Relationships between quantum circuit synthesis, optimization and simulation.

2.5 State of the Art

Synthesis methods can change radically depending on the abstract representation of the quantum operator. We give a general review of the state of the art in the following cases:

- the synthesis of generic operators given by their matrix representation $U \in \mathcal{U}(2^n)$,
- the synthesis of CNOT circuits given by their matrix representation $A \in F_2^{n\times n}$,
- the synthesis of CNOT+T and CNOT+Rz circuits given by their phase polynomial representation.

2.5.1 Synthesis of Generic $\mathcal{U}(2^n)$ Operators

In this synthesis problem, we deal with unitary matrices of size 2^n without any particular structure. This highlights an important feature about generic quantum circuit synthesis: the problem is exponentially hard by nature. Having complete knowledge of the operator requires an exponentially sized memory and an equivalently exponential amount of time to be able to access each component of the matrix. Therefore the use of heuristic methods or optimization frameworks is limited to problems defined on a very limited number of qubits. However, the state of the art is rich in algebraic methods with theoretical results and we give a historical overview.

Two works in 1995 laid the foundations of quantum circuit synthesis: in [17] several identities for decomposing multi-controlled operators into circuits of one and two-qubit gates were given. Combining these results with a QR decomposition formula from [161] the authors showed that any operator on n qubits can be synthesized with a circuit containing $\mathcal{O}(n^3 4^n)$ elementary one and two-qubit gates. They also conjectured a lower bound on the number of elementary two-qubit gates required to synthesize an arbitrary n-qubit operator:

$$\Omega(n) = \frac{1}{9}4^n - \frac{1}{3}n - \frac{1}{9}. \tag{2.1}$$

By lower bound we mean that *almost all* operators need at least this number of two-qubit gates to be synthesized. Some operators can be implemented with fewer gates, take for instance the identity operator that needs 0 gates to be implemented, but they represent a negligible fraction of operators. This lower bound shows that almost every operator needs an exponential number of gates for their execution on a quantum computer. More details about the lower bounds are given in Chapter 5. Later in 1995 Knill improved the result from [17] and reduced the number of elementary two-qubit gates to $\mathcal{O}(n4^n)$ [110]. He also formalized the lower bound Eq. (2.1) and studied the case where an approximation of the operator is allowed. This does not change the exponential complexity required for most of the operators. To illustrate it, we quote one remark Knill made about state preparation:

> " if $b\log(b) = o(k2^n)$, then for most $|u\rangle$ the circuits with less than b gates can do essentially no better at mapping $|0\rangle$ to $|u\rangle$ than a random unitary operator." [110]

Therefore the future works about circuits synthesis focused on improving the asymptotic complexity of exact methods. By combining other works in the literature it is possible to recover the same complexity in $\mathcal{O}(n4^n)$ [3, 43] but the complexity remained a factor of n from the theoretical lower bound for almost 10 years. A breakthrough happened in the years 2004-2006 when a series of papers significantly improved the number of CNOT gates and one-qubit gates required for the

synthesis of general quantum operators [143, 174, 176, 194]. First Shende *et al.* improved the lower bound for CNOT based quantum circuits [176]:

$$\Omega(n) = \left\lceil \frac{1}{4}(4^n - 3n - 1) \right\rceil. \tag{2.2}$$

Then an improvement over the initial QR decomposition was proposed in [194] and reached a complexity of $\mathcal{O}(4^n)$. Using Givens rotations, the method has a complexity evaluated at approximately 8.7×4^n while the upper bound is 11×4^n, still a factor 44 away from the theoretical lower bound. With the use of the Cosine-Sine Decomposition (CSD) [66] — whose use was already sketched in [188] — and uniformly-controlled one-qubit rotations the same authors first reached a complexity of $4^n - 2^{n+1}$ CNOTs [143] before improving their result by introducing an efficient decomposition for uniformly-controlled general one-qubit gates. This led to a complexity of $\frac{1}{2}4^n - \frac{1}{2}2^n - 2$ CNOTs and $4^n - 1$ R_z or R_y rotations.

At the same time, another research team used another quantum circuit decomposition also based on the CSD, the NQ decomposition, to produce circuits of similar complexity, namely $\frac{1}{2}4^n - \frac{3}{2}2^n + 1$ CNOTs and $\frac{9}{8}4^n - \frac{3}{2}2^n + 3$ R_y or R_z rotations [175]. They also used uniformly-controlled one-qubit rotations but under the name of *quantum multiplexors*.

Finally, both teams improved their results to reach a final CNOT complexity of $\frac{23}{48}4^n - \frac{3}{2}2^n + \frac{4}{3}$ [142, 174]. The global quantum circuit decomposition used in [174] is the Quantum Shannon Decomposition (QSD). For more details about the CSD, QSD, and quantum multiplexors see Chapter 4.

To our knowledge, no method improving the CNOT complexity of the QSD and the method from [142] has been proposed since. In [165] a method combining the QSD and the CSD was used to reduce the total number of gates in the circuits. They managed to produce circuits with better tradeoffs between the number of CNOTs and the number of one-qubit gates but they did not improve the asymptotic complexity of the CNOT count.

More recently, in [50] another decomposition was proposed: the block-ZXZ decomposition. This new decomposition relies on a block version of Sinkhorn's decomposition of unitary matrices [64, 82]. It does not improve the gate count but shows new theoretical relationships between different groups of unitaries. Moreover, this decomposition is more suitable for the synthesis of classical reversible circuits as it preserves the structure of the permutation matrices encoding the reversible functions.

Synthesis of Generic Quantum States, Isometries, Diagonal Operators

State Preparation. The history of algorithms on state preparation is similar to that on quantum circuits synthesis. Initially, Knill proved an upper bound of $\mathcal{O}(n2^n)$ on the number of gates [110]. Then efficient methods for the synthesis of quantum states were provided in [142, 174]. Those methods rely on the use of quantum multiplexors. The best circuit that transforms a state $|a\rangle$ to a state $|b\rangle$ requires $2 \times 2^n - 2n - 2$ CNOTs and $2 \times 2^n - n - 2$ one-qubit gates. For standard state preparation, $a = |0\rangle^{\otimes n}$ and the gate count is roughly divided by 2, giving a leading complexity of 2^{n-1} CNOTs and 2^n one-qubit gates [142].

In 2011, the highest known lower bound on the number of CNOTs for state preparation was derived:

$$\Omega(n) = \left\lceil \frac{1}{4}(2^{n+1} - 2n - 2) \right\rceil \tag{2.3}$$

and a new method for state preparation was designed [155]. The method relies on the use of Schmidt's decomposition [153] to reformulate state preparation on n qubits as quantum circuits

synthesis on $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ qubits. Those unitaries can be synthesized with the QSD method and the circuits produced have asymptotically $\frac{23}{24}2^n$ CNOTs if n is even, slightly improving the previous best method from [142].

For some families of quantum states, efficient algorithms are known. A series of papers proposed polynomial sized quantum circuits for preparing a state $|\psi\rangle$ [70, 97, 180] up to an arbitrary error providing some information on the amplitudes and phases of the components of $|\psi\rangle$ is classically computable efficiently.

Isometries. In [86] the concept of *isometries* was introduced as a generalization of quantum circuit synthesis and quantum state preparation. Isometries are notably useful for the design of quantum channels [85]. An m to n isometry can be thought as a unitary operator acting on a set of n qubits with $n-m$ of them being in a fixed state, $|0\rangle$ for instance. When $m = n$ the isometry is a complete unitary operator. When $m = 0$ the isometry is equivalent to state preparation. For arbitrary $m \leq n$ an m to n isometry is given by a $2^n \times 2^m$ complex matrix V such that

$$V^\dagger V = I_{2^m \times 2^m}.$$

All the concepts related to quantum circuit synthesis and state preparation apply. In [86] the authors show that

$$\Omega(n,m) = \frac{1}{4}(2^{n+m+1} - 2^{2m} - 2n - m - 1) \tag{2.4}$$

is a lower bound on the number of CNOTs needed to implement m to n isometries. They also give several constructions for arbitrary isometries. Their methods rely on multiplexors, the QSD, and Knill's decomposition from [110]. For each method, an estimation of the number of CNOTs required is given. These methods form the basis of an open-source software package *UniversalQ-Compiler* written in Mathematica. Additional details, notably about the classical run time, are given in [87].

Diagonal Operators. Diagonal operators are of the form

$$D = \begin{pmatrix} e^{i\theta_1} & & \\ & \ddots & \\ & & e^{i\theta_{2^n}} \end{pmatrix}$$

and are completely specified by the angles of each entry. Standard techniques for decomposing a diagonal operator to a circuit with $2^n - 1$ R_z rotations and $2^n - 2$ CNOTs are known [36, 174]. Recent works optimized the approximate synthesis of diagonal operators in the Clifford+T gate set [201] or the standard CNOT+$\mathcal{SU}(2)$ gate set [202]. The former work proposes different circuits constructions based on cascaded entanglers while the latter expands diagonal operators in the specific basis of Walsh functions [199]. When either the number of distinct phases is low (in [201]) or the diagonal operator is sparse in the Walsh basis (in [202]) then the constructions provide more optimized circuits. Diagonal operators are also well understood as part of the CNOT+R_z framework, see Section 2.5.3 for more details.

Synthesis of Operators on a Small Number of Qubits

For quantum operators of arbitrary size, the improvements in the synthesis methods are essentially done for a worst-case scenario. With operators of fixed small size, one can expect to use more optimized routines. For instance, the problem can be considered to be solved when $n = 2$.

It has been shown that at most 3 CNOTs and 15 elementary rotations are sufficient in the worst case, with explicit constructions given in [176]. Moreover, circuits with 3 CNOTs are proven to be necessary in some cases — the SWAP gate, for instance, cannot be implemented with less than 3 CNOTs [195]. A characterization of the 2-qubit operators that can be implemented with 2 CNOTs has also been provided [197]. Given the fact that operators implementable with 0 or 1 CNOT are easily identifiable, the implementation of any two-qubit operator can be made optimally. All these methods rely on an algebraic decomposition called Cartan's KAK decomposition, more details are given in Section 2.5.1.

For $n = 3$, a decomposition with 40 CNOTs and 98 elementary R_y and R_z rotations has been first presented in 2004 [196]. To our knowledge, the best algebraic construction for $n = 3$ is now given by the QSD with 20 CNOTs. The theoretical lower bound is 14. As far as we know, no specific construction has been proposed for $n > 3$.

Besides custom circuit decompositions, non-algebraic methods turn out to be useful for small problem sizes. For instance, brute-force search (with some optimization) remains feasible and guarantees optimality in the results. A meet-in-the-middle algorithm was proposed in [11] to synthesize circuits of optimal depth in the Clifford+T set. Optimal synthesis of all 4-bit reversible functions is reported in [67].

The use of heuristics also provides interesting results. In [121] genetic algorithms were used for an approximate synthesis of adders with experiments on a 5-qubit chip from IBM. Genetic algorithms were also used for the synthesis of classical reversible gates [125, 126].

Theoretical Insights

Decompositions of unitary matrices are part of the theory of Lie groups and Lie algebras. More precisely, this is the use of Cartan's KAK decompositions that have found great utility for the synthesis of quantum circuits. The first application of KAK decompositions can be found in [98]. The authors gave a decomposition of any $U \in \mathcal{SU}(2^n)$ into a product of unitary matrices in $\mathcal{SU}(2^{n-1}) \otimes \mathcal{SU}(2)$ and exponentials of elements from some Cartan subalgebras. They also explained the case $\mathcal{SU}(4)$ which served as a basis to other works in the literature that provide efficient decompositions of 2-qubit operators [35, 195, 197, 212].

Since then much progress has been made. This results in a deeper understanding of the role KAK decompositions can play in the decomposition of unitary matrices and the synthesis of quantum circuits [189]. For example, we now know that the QSD is, in fact, a special case of KAK decomposition [34, 53]. New decompositions and generalizations of existing decompositions have also been provided [44, 54, 144]. Yet, in the realm of quantum circuit synthesis, the QSD remains the preferred decomposition and KAK based methods seem confined to theoretical results.

2.5.2 Synthesis of CNOT Circuits and Related Classes of Circuits

CNOT circuits implement a subclass of classical reversible operators called *linear reversible operators*. Linear irreversible circuits have a much longer and richer history than its reversible counterpart [6, 28, 191]. In both irreversible and reversible cases, it is known that a linear operator acting on n (qu)bits is encoded into a boolean matrix $A \in F_2^{n \times n}$ where the i-th row of A gives the output of the i-th bit as a linear combination of the inputs. In our case, the sum is given by the XOR operation but other models of linear circuits exist, for example with the OR operation [94]. In the reversible case, A needs to be invertible and the basic linear gate used is the CNOT gate. In the irreversible case, no assumption is made on A and there are fewer constraints, notably on the use of garbage bits, making it difficult to transpose techniques to the quantum case.

In the literature, it was first shown that n^2 CNOT gates are sufficient to synthesize any linear reversible operator with a Gaussian elimination algorithm [177]. Then Patel *et al.* derived a lower bound of $\mathcal{O}(n^2/\log_2(n))$ CNOT gates and proposed a block version of the Gaussian elimination algorithm [152]. By lower bound we mean that at least one n-qubit operator requires $\mathcal{O}(n^2/\log_2(n))$ CNOT gates to be implemented. [152, Algo. 1] reaches circuits of size $\mathcal{O}(n^2/\log_2(n))$. This algorithm is asymptotically optimal: its worst-case complexity can only be improved by a constant factor. To our knowledge [152, Algo. 1] is the best algorithm for the synthesis of CNOT circuits when the metric to optimize is the size of the circuits and when there is full qubit connectivity. It is notably used in recent quantum compilers [9, 10].

When the qubit connectivity is restricted, the first proposed approach has been to transform the circuits given by an unrestricted algorithm with swap insertion algorithms to match the connectivity constraints [120,154,204]. To produce more efficient circuits, two concomitant papers proposed a modification of the Gaussian elimination algorithm with the use of Steiner trees [102, 146]. This new method outperforms swap insertion techniques and produces circuits of size at most $\mathcal{O}(n^2)$.

Algorithms producing shallow circuits were also proposed. First, it was shown that any n-qubit CNOT circuit can be parallelized to $\mathcal{O}(\log_2(n))$ depth with the help of n^2 ancillary qubits [141]. In 2007 an ancilla-free algorithm producing circuits of linear depth was proposed with an extension to restricted connectivities [130]. At the same time, Kutin *et al.* proposed an algorithm with linear depth complexity for the Linear Nearest Neighbor (LNN) architecture [117]. During a decade the best ancilla-free methods produced circuits of linear depth while the theoretical optimal bound is $\mathcal{O}(n/\log_2(n))$ in a fully connected architecture. A major question was therefore whether this bound could be reached. The answer was given very recently in [90] with a theoretical algorithm producing circuits of depth $\mathcal{O}(n/\log 2(n))$, matching the theoretical bound. The authors also improve the case where ancillary qubits are available. They notably show that $\mathcal{O}(n^2/\log_2^2(n))$ ancillae are sufficient to reduce the depth to $\mathcal{O}(\log_2(n))$.

Synthesis of Stabilizer Circuits

Stabilizer circuits are a subclass of quantum circuits containing solely Hadamard gates, Phase gates, and CNOT gates. They implement quantum operators belonging to the so-called Clifford groups $\mathcal{C}_n$ and any Clifford operator can be implemented by a stabilizer circuit. In other words

$$\Im(\{H, S, CNOT\}) = \bigcup_{n\geq 1} \mathcal{C}_n$$

up to a global phase. $H, S, CNOT$ are said to be *Clifford gates*. Stabilizer circuits over n qubits are in a one-to-one equivalence with matrices from the binary symplectic group $SP(2n, F_2)$ and are known to be efficiently simulable by a classical computer [2, 69]. Therefore the set of Clifford gates is not universal for quantum computation and stabilizer circuits offer no computational advantage compared to classical computation. However, they play an important role in the design of error-correcting codes — the so-called *stabilizer codes* [38,68] —, quantum tomography, randomized benchmarking, etc.

Clifford operators are implemented via canonical forms of stabilizer circuits. Canonical forms are given as a series of layers containing only one type of Clifford gate. For instance, in [2] it was shown that any stabilizer circuit can be written as -H-C-P-C-P-C-H-P-C-P-C- where -H- stands for a layer of Hadamard gates, -P- is a layer of Phase gates and -C- a layer of CNOT gates, namely linear reversible circuits. The size of the -H- and -P-layers can only be $\mathcal{O}(n)$ as $H^2 = I$ and $P^4 = I$, therefore the size of stabilizer circuits is dominated by the size of the -C- stages. To our knowledge,

this 11-stage decomposition was the reference decomposition until Maslov and Roetteler provided a shorter canonical form [134]. This new decomposition was then improved very recently in [31].

Linear reversible circuits and stabilizer circuits are closely related, one is a subclass of the other. Any improvement in the synthesis of CNOT circuits also improves the synthesis of stabilizer circuits. It turns out that the converse may be true. In [31] the authors discuss the computational advantage of using other Clifford gates to implement CNOT circuits to reach lower CNOT counts. They exhibit a concrete example where the use of Hadamard gates enables to implement a CNOT circuit of size 8 with only 7 CNOTs. Yet they also show that the size can only be improved by a constant factor.

2.5.3 Synthesis of CNOT+T Circuits

CNOT+T circuits (and its extension CNOT+Rz) represent another non-universal subclass of quantum circuits. The action of a CNOT+T circuit can be summarized with its phase polynomial representation. This structure is of polynomial size in the number of qubits and the number of T gates in the circuit [10,11]. This compact representation makes it easier to design efficient methods to optimize several metrics like the T-count, the T-depth, or the CNOT count.

In [172] a method for minimizing the T-depth was presented but it focused on the class of circuits that can be parallelized to T-depth one with the help of ancillae. This work has been extended, improved in [10] with the CNOT+T formalism. The proposed algorithm, Tpar, optimizes both T-count and T-depth. Much progress has been made since with better theoretical understandings on the T-count optimization of CNOT+T circuits but also with efficient optimization algorithms [12, 80, 138].

Recent quantum compilers for the Clifford+T gate set rely on the optimization of CNOT+T circuits. Either by rearranging CNOT+T circuits around the Hadamard gates [10] or by gadgetizing the said Hadamard gates [80], the optimization of CNOT+T circuits is central in the minimization of the cost of a quantum circuit.

Although fault-tolerant protocols are much more costly for the T gate than for Clifford gates [150], the Clifford cost may represent a non-negligible part, and even the most costly part, of running an entire quantum circuit [132]. In that case, one should also optimize the Clifford cost, namely the CNOT cost, in CNOT+T circuits. Optimizing the CNOT circuits occurring in a CNOT+T circuit is one way to go. Amy *et al.* designed an algorithm, GraySynth, that optimizes directly the CNOT count in CNOT+Rz circuits [9] for an architecture with full qubit connectivity. This work was then extended to restricted connectivities [49, 146].

2.5.4 Other References

We present succinctly some reference works about problems that are part of quantum circuit synthesis but that are not directly in the scope of this thesis.

(H,T) Synthesis

The Clifford+T gate set is universal for quantum computation. This non-trivial theorem relies on two results:

1. the set of one-qubit gates with the CNOT gate is universal [17],
2. any one-qubit gate can be approximated at an arbitrary precision with the (H,T) gate set. This is the Solovay-Kitaev theorem [45, 103].

In this section, we focus on practical implementations of the second result. The Solovay-Kitaev theorem also states that the number of (H,T) gates required is only logarithmic in the inverse of the error. More precisely the length of the gate sequence was shown to be $\mathcal{O}(\log^c(1/\epsilon))$ with ϵ the error and c is a constant which was evaluated at the time between 3 and 4. In other words, any one-qubit gate can be efficiently approximated with only the two gates H and T but there was room for improvement as the optimal scaling is known to be $\mathcal{O}(\log(1/\epsilon))$ [45].

Today the problem can be considered as solved. Several works in the years 2011-2016 got closer and closer to an optimal algorithm. First, a brute-force algorithm was proposed, capable of handling errors up to $\epsilon = 10^{-4}$ [61]. An optimal algorithm was proposed for single-qubit unitaries that can be synthesized exactly with the set (H,T) [108]. Then more and more efficient approximation methods were published, focusing on the approximation of R_z rotations. With ancillary qubits, an asymptotically optimal algorithm was designed [107]. Without ancillae, an improvement over [61] was proposed in [109]. It was capable of reaching errors around $\epsilon = 10^{-17}$ but its classical runtime was still exponential in the number of T gates. An algorithm reaching a T-count of $K + 4\log_2(1/\epsilon)$ was proposed [173], making one step closer to the information-theoretic lower bound of $K + 3\log_2(1/\epsilon)$. Finally, an optimal and fast probabilistic algorithm is given in [164].

While the size of purely unitary circuits produced by the best methods attain the theoretical lower bound [164], with additional techniques it is possible to generate even shorter circuits. For instance, the Repeat-Until-Success (RUS) protocol produces (non-unitary) circuits with approximately 2.5 times fewer gates than this theoretical lower bound [26]. A simpler protocol is also proposed in [25].

Finally, some extensions to other universal gate sets have been presented [105, 163].

Reversible Synthesis, Oracle Synthesis

When an operator consists in a permutation of the basis states, its matrix representation is a permutation matrix of size 2^n and we enter the realm of reversible circuit synthesis [177]. We refer the reader to [166] for a survey about this topic.

Oracle synthesis is a special case of reversible circuit synthesis where we want to implement a specific Boolean function over n inputs. Oracles are used in many different quantum algorithms [39, 71, 77]: the complexity of quantum algorithms is sometimes measured in the number of calls made to the oracle. It is, therefore, crucial to be able to provide efficient implementations of those logical functions.

Given a Boolean function f over n bits, the usual way to implement it reversibly is to use an ancillary qubit and to perform the operation U defined on $n+1$ qubits by

$$U\left|x\right\rangle\left|y\right\rangle = \left|x\right\rangle\left|f(x) \oplus y\right\rangle.$$

f can be given in multiple different forms (formula, boolean network, *etc.*). Several methods have been designed during the last years and we cannot summarize them all. We refer the reader to some recent works in the literature, mostly focusing on the quantum implementation in the Clifford+T gate set [137, 139].

2.6 Problems Tackled in this Ph.D. thesis

We summarize in Table 2.3 the different problems that we have tackled during this Ph.D. thesis. Below is the list of the main algorithms/routines that have been produced for solving each of these problems.

1. ZUNQRF is a LAPACK-like [13] routine for a specific QR factorization designed for unitary matrices. We can express the synthesis of a unitary matrix $U \in \mathcal{U}(2^n)$ as a succession of state preparations that can be performed either using the standard {CNOT + $\mathcal{SU}(2)$ } gate set or the specific { MS + $\mathcal{SU}(2)$ } gate set for trapped-ions. The main objective is to optimize the compilation time in order to address larger problem sizes than those tractable with current state-of-the-art algorithms. We also optimize the total number of gates so that we produce circuits at most twice larger than those produced by the best algorithm in the literature. The synthesis is exact and may not require any ancillary qubit. These results have been published in Elsevier's journal *Computer Physics Communications* [47].

2. We use the well-known BFGS algorithm to synthesize circuits of optimal size for random operators. Given a unitary U, we use the BFGS algorithm to compute the best angles in a parameterized circuit given in a canonical form of optimal size. We reach the theoretical lower bound and give insights about the validity of such bound. Again, our framework works for both the standard { CNOT + $\mathcal{SU}(2)$ } gate set or the specific trapped-ions { MS + $\mathcal{SU}(2)$ } gate set. Due to the "continuous variable optimization" nature of this framework, the synthesis is approximate. We do not use ancillary qubits. Part of these results have been presented at ICCS 2019 and they are published by Springer in *LNCS* [46].

3. GreedyGE is a greedy variant of the Gaussian elimination algorithm. We use it for the synthesis of CNOT circuits that can be represented as a boolean matrix and we optimize the size of the circuits. The synthesis is exact, does not require ancillary qubits, and can be made on unconstrained architectures. We also optimize the compilation time. Overall this method has the best asymptotic results, outperforming the state-of-the-art methods in both circuit size and compilation time.

4. DaCSynth is a divide-and-conquer algorithm for the synthesis of CNOT circuits on unconstrained architectures. The goal is the optimization of the depth of the circuits and a version both with and without ancillary qubits have been designed. We report the best results for operators of intermediate size ($< 10^3$ qubits).

5. We designed an algorithm for CNOT circuits synthesis that relies on the solution of the cryptographic problem called the "syndrome decoding problem". We use it for optimizing the size of the circuits on unconstrained and constrained architectures. We do not use ancillary qubits. Our method outperforms state-of-the-art methods for various architectures. These results have been presented at RC 2020 and they are published by Springer *LNCS* [48].

2.7 Notations

Throughout this manuscript, the term *flops* stands for *floating-point operations* and the flop count evaluates the volume of work in a computation. Unless otherwise specified theses flops are given in complex arithmetic. The linear algebra formulas will be presented using Matlab-like notations.

	Chapter 4	Chapter 5	Chapter 9	Chapter 10	Chapter 11
Abstract Representation	$U \in \mathcal{U}(2^n)$	$U \in \mathcal{U}(2^n)$	$A \in F_2^{n\times n}$	$A \in F_2^{n\times n}$	$A \in F_2^{n\times n}$
Set of gates $\mathcal{S}$	{ CNOT, $\mathcal{SU}(2)$ } or { MS, $\mathcal{SU}(2)$ }	{ CNOT, $\mathcal{SU}(2)$ } or { MS, $\mathcal{SU}(2)$ }	{CNOT}	{CNOT}	{CNOT}
Error ϵ	0	Arbitrary	0	0	0
Ancillary qubits m	0	0	0	Arbitrary	0
Cost function f	Size	Size	Size	Depth	Size
Main objective	$T(U_C)$	f	f	f	f
Second objective	f	$T(U_C)$	$T(U_C)$	$T(U_C)$	$T(U_C)$
Algorithm	ZUNQRF	BFGS	GreedyGE	DaCSynth	Syndrome

Table 2.3: Problems tackled and algorithms designed in the Ph.D. thesis.

Part B

Synthesis on $\mathcal{SU}(2^n)$

Chapter 3

Introduction to Generic Quantum Circuits Synthesis

Contents

3.1 Presentation of the Problem

This part of the thesis is dedicated to what we informally call *generic quantum circuits synthesis*, i.e., the synthesis of generic operators given as unitary matrices. We immediately exclude any subclass of operators that has a custom representation and custom synthesis methods: Clifford circuits, CNOT+T circuits, etc. We also assume that the operators to synthesize are random, sampled from a uniform distribution according to the Haar measure [73]. This measure is the standard measure for sampling random matrices from compact topological groups, e.g., $GL_n(\mathbb{C}), \mathcal{U}(n)$. For QRAM generation — probably the main application of generic synthesis — the assumption of randomness in the classical data to load seems viable, for instance for the generation of an unstructured database [23].

In other words, we do not expect the operators to have a particular short circuit implementation. We will not study methods that try to optimize at most the size of the circuits: these methods generally either try to exploit the specificities of the input matrix or perform some kind of brute force search to find an optimum. Given that random operators almost surely represent the worst cases possible — more details are given in Chapter 5 —, in the former case there are no specificities an algorithm can exploit and in the latter case the methods are limited to extremely small problem sizes for savings that will not be as consequent as one can expect from an exact method.

Instead, we focus on the asymptotic worst-case behavior. As a function of the number of qubits, generic synthesis of quantum circuits takes exponentially large matrices as inputs and outputs exponentially large quantum circuits most of the time. The scalability of any algorithm for this problem is therefore limited. In this thesis we push the limits of generic quantum circuits synthesis a little further, in two antinomic directions:

1. What is the largest unitary one can synthesize in a *reasonable* amount of time?

2. What is the shortest circuit one can synthesize for a given unitary in a *reasonable* amount of time?

Those two metrics — circuit size/quantum resources and generation time/classical resources — are unfortunately incompatible. If one wants the shortest circuit possible then the use of exponentially costly algorithms is often necessary. This results in a doubly exponential algorithm as the inputs are already exponentially large in the number of qubits. Faster algorithms will not produce optimal circuits but at least larger problems can be addressed. This inevitable tradeoff between classical and quantum resources is at the core of our work and is one of its guiding threads. More precisely, methods for generic synthesis produce circuits of asymptotic size $\alpha \times 4^n$ where n is the number of qubits (as discussed in Section 2.5). Our goal is to understand how to optimally trade the value of α such that the circuits generation time remains reasonable. By reasonable we mean less than a few hours, but all our results can be extended to the case where more running time is allowed.

To summarize, we focus on the following task:

Synthesize an operator on n qubits in a reasonable amount of time with a circuit of size $\alpha \times 4^n$.

Our goal is to succeed for as large an n as possible and, for each n, with the smallest α possible. This part is divided into three chapters:

1. In Chapter 4 we propose a fast algebraic algorithm based on the QR factorization with Householder transformations to synthesize the largest unitaries possible while keeping α as low as possible. These results have been published in Elsevier's journal *Computer Physics Communications* [47].

2. In Chapter 5 we use numerical optimizers to synthesize unitaries with the lowest α possible and we explore the tradeoff between classical time and circuit size. Part of these results have been presented at ICCS 2019 and they are published by Springer in *LNCS* [46].

3. Finally, in Chapter 6 an attempt to extend the reuse method [104] to generic circuits synthesis is proposed. These results have been presented at AMMCS 2017 and they are published by *Springer in Recent Advances in Mathematical and Statistical Methods* [5] .

3.2 Notations

Throughout this part, we will make use of the following notation: $\#\text{Op.}_{\text{Meth.}}(n)$ gives the entangling cost, i.e., the number of entangling gates (most of the time this will be the CNOT cost) for synthesizing the family of operators "Op.", with the method "Meth", as a function of the number of qubits n. For instance:

- $\#\text{U}_{\text{QSD}}(n)$ is the number of CNOT gates in the circuits given by the Quantum Shannon Decomposition (QSD) for the synthesis of generic unitary operators on n qubits.
- $\#\text{SP}_{\text{Schmidt}}(n)$ is the number of CNOT gates in the circuits given by the method using Schmidt's decomposition for state preparation from [155].

For completeness, all the families of operators, all the methods and all the corresponding notations we will encounter in this part are given in Table 3.1.

Family of operators	Method	Notation
Generic unitary	QSD	$\#\mathrm{U}_{\mathrm{QSD}}$
	Householder	$\#\mathrm{U}_{\mathrm{Hous.}}$
	Householder + rotation multiplexors	$\#\mathrm{U}_{\mathrm{Hous.,\ rot}}$
	Householder + $\mathcal{SU}(2)$ multiplexors	$\#\mathrm{U}_{\mathrm{Hous.,\ su}}$
	Householder + Schmidt's decomposition	$\#\mathrm{U}_{\mathrm{Hous.,\ Schmidt}}$
	Householder + Schmidt's decomposition + QSD	$\#\mathrm{U}_{\mathrm{Hous.,\ Schmidt,\ QSD}}$
	Up to diagonal operator	$\#\mathrm{U}_{\mathrm{-diag}}$
State preparation	Rotation multiplexors	$\#\mathrm{SP}_{\mathrm{rot}}$
	$\mathcal{SU}(2)$ multiplexors	$\#\mathrm{SP}_{\mathrm{su}}$
	Schmidt's decomposition	$\#\mathrm{SP}_{\mathrm{Schmidt}}$
	Schmidt's decomposition + QSD	$\#\mathrm{SP}_{\mathrm{Schmidt,\ QSD}}$
	Real state with R_y-multiplexor	$\#\mathrm{SP}_{\mathrm{real,\ rot}}$
Isometries	QSD	$\#\mathrm{Iso}_{\mathrm{QSD}}$
	Householder + rotation multiplexors	$\#\mathrm{Iso}_{\mathrm{Hous.,\ rot}}$
	Householder + $\mathcal{SU}(2)$ multiplexors	$\#\mathrm{Iso}_{\mathrm{Hous.,\ su}}$
	Householder + Schmidt's decomposition	$\#\mathrm{Iso}_{\mathrm{Hous.,\ Schmidt}}$
	Householder + Schmidt's decomposition + QSD	$\#\mathrm{Iso}_{\mathrm{Hous.,\ Schmidt,\ QSD}}$
Multiplexors	Rotations	$\#\mathrm{Mult}_{\mathrm{rot}}$
	$\mathcal{SU}(2)$ operators	$\#\mathrm{Mult}_{\mathrm{su}}$
Diagonal operators	R_z multiplexors	$\#\mathrm{Diag}_{\mathrm{rot}}$
Generalized Toffolis	From [78]	$\#\mathrm{Toff}_{[78]}$

Table 3.1: Summary of the notations used in Part B.

Chapter 4

A QR-Based Synthesis Algorithm Using Householder Transformations

Contents

4.1 Motivation and Technical Background

This chapter is dedicated to a question that has gathered little interest since now: the classical time complexity of quantum circuit synthesis methods. This question is doubly important:

- Improving the time complexity will be useful when one has to compile a continuous stream of quantum circuits on the fly or when the quantum operator is parameterized and one has to recompile the parameters of the resulting quantum circuit every time the operator changes.
- Maybe more importantly, improving the compilation time also allows reaching larger problem sizes. This will be of crucial importance for quantum algorithms that need to process

classical data. If the size of the classical data to load is too large the synthesis of the associated unitary operator may be impossible because it will require too much classical time. Fast quantum circuit synthesis methods are important if only to widen the field of applications of quantum algorithms.

In the literature related to quantum circuit optimization, the running times of the methods are usually provided [9, 10, 80]. This is particularly instructive when the algorithm is a heuristic and that it is complicated to give a theoretical time complexity. In quantum circuit synthesis, the approach is different. The methods in the literature are mostly algebraic, with exponential complexity. Therefore the time complexity was not a major concern until the last decade where some works integrated it in their analysis [11, 87, 135]. Yet we face two limitations: first, the running times were given in the context of (optimized) brute-force search methods where the scalability of the methods is limited (only a few qubits) and represents the main issue to tackle [11, 135]. For purely algebraic quantum circuit synthesis methods the time complexity was given theoretically with no deeper insights [87]. For instance, the constant factors of the leading terms were not given. We claim they are very important and we show in this chapter that improving the constant factor enables us to reach problems with one, two, three more qubits, and consequently to process classical data that are two, four, eight times larger.

We start by giving an overview of the different quantum circuit synthesis methods. We introduce the basic blocks and basic structures used as we will need them for the method presented in Sections 4.3 and 4.4. We also investigate the time complexity of state-of-the-art methods. If not specified otherwise, all the calculations we present can be found in the corresponding original article.

4.1.1 Quantum Multiplexors, Diagonal Operators

Also named *uniformly controlled-rotations* [18], quantum multiplexors are a generalization of the controlled gates. A unitary positively controlled by k qubits is applied if the value of all k qubits is $|1\rangle$, otherwise the identity operator is applied. A quantum multiplexor applies a different operator for each value of the control qubits. For instance, a multiplexor controlled by two qubits and acting on k qubits has the matrix block structure

$$A = \begin{pmatrix} A_0 & 0 & 0 & 0 \\ 0 & A_1 & 0 & 0 \\ 0 & 0 & A_2 & 0 \\ 0 & 0 & 0 & A_3 \end{pmatrix}$$

where A_0 is a k-qubit operator that is applied if both control qubits are 0, A_1 is applied if the first control qubit is 0 and the second one is 1, etc. The graphical representation of a multiplexor is illustrated in Figure 4.1 with the correspondence between A and a succession of multi-controlled gates. In the circuit, the crossed-out line stands for several qubits (in this case, k qubits).

The problem of decomposing a multiplexor into elementary gates admits algorithms with varying numerical costs depending on the choice of elementary gates [143, 174]. In the case of a multiplexor applying only one kind of elementary rotations (along one of the axis Y or Z – we call such a structure a rotation multiplexor) the transition from the angles of the multiplexors to the angles in the corresponding quantum circuits can be performed via a single matrix-vector product [143]. Moreover, the decomposition is much simpler than the general case shown in Figure 4.1: the decomposition of an R_k-multiplexor controlled by n qubits into two multiplexors controlled by $n-1$ qubits has the shape shown in Figure 4.2 and the special case with one control qubit is shown in

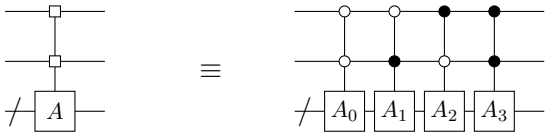


Figure 4.1: Circuit equivalence for a multiplexor.

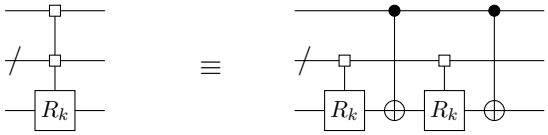


Figure 4.2: Decomposition of a rotation multiplexor.

Figure 4.3 (where we omit angles for legibility). Such decompositions can be applied recursively and by removing some CNOT gates that cancel (see Figure 2 in [174] for more details) we obtain a final quantum circuit composed of

$$\#\text{Mult}_{\text{rot}}(n) = 2^{n-1} \tag{4.1}$$

CNOTs and 2^{n-1} elementary rotations. Note that, for the synthesis of R_y multiplexors, a functionally equivalent circuit can be obtained by replacing each CNOT gate by a Controlled-Z (CZ) gate [174]. For multiplexors in $\mathcal{SU}(2)$ the decomposition given in Figure 4.2 remains valid up to a diagonal matrix that replaces the last CNOT gate and operators in $\mathcal{SU}(2)$ that replace the elementary rotations. Hence, without considering the diagonal gate — for our purpose we will be able to remove it — we need

$$\#\text{Mult}_{\text{su}}(n) = 2^{n-1} - 1 \tag{4.2}$$

CNOTs and 2^{n-1} generic one-qubit gates to implement an $\mathcal{SU}(2)$-multiplexor.

Quantum multiplexors are expressive enough to be reused in other constructions, thus helping during the compilation process. This structure is at the core of modern quantum circuit synthesis methods producing the best results in terms of gate count [142, 174].

Diagonal Operators

An example of the use of multiplexors is for the synthesis of diagonal operators. In [174] it was shown that a diagonal operator on n qubits is equivalent to a diagonal operator on $n-1$ qubits plus an R_z multiplexor controlled by those same $n-1$ qubits. For instance, given a diagonal operator D, for each bitstring s of size $n-1$ one can always find an angle α_s such that

$$R_z(\alpha_s) \begin{pmatrix} D_{0s} & \\ & D_{1s} \end{pmatrix} = e^{i\theta_s} I_2$$

where $\begin{pmatrix} D_{0s} & \\ & D_{1s} \end{pmatrix}$ is a 2×2 matrix and D_{0s} and D_{1s} are the diagonal elements of D at location $0s$ and $1s$. Setting $D_{0s} = e^{i\phi_s}$ and $D_{1s} = e^{i\varphi_s}$ one can check that $\alpha_s = \phi_s - \varphi_s$ and $\theta_s = (\phi_s + \varphi_s)/2$ works. Applying all those R_z rotations for each bitstring s defines an R_z multiplexor controlled by the last $n-1$ qubits. After application of such a multiplexor, we have a new operator $D' = \begin{pmatrix} D_2 & \\ & D_2 \end{pmatrix}$ with D_2 diagonal on $n-1$ qubits and for each bitstring s of size $n-1$ we have

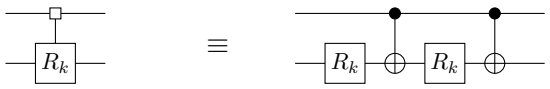


Figure 4.3: Decomposition of a rotation multiplexor with one control qubit.

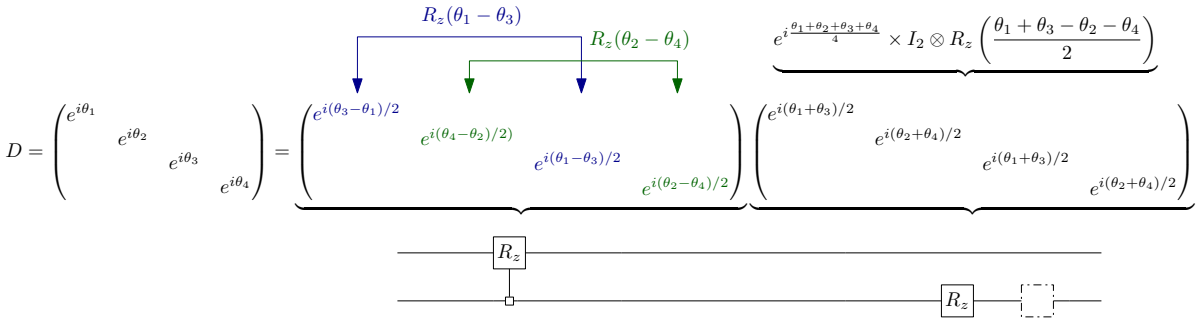


Figure 4.4: Process for the synthesis of a diagonal operator on 2 qubits. The dashed dotted box stands for a global phase, it can be applied on any qubit and at any time in the circuit.

$(D_2)_s = e^{i\theta_s}$. One can pursue the synthesis of D_2 on the last $n-1$ qubits. The synthesis process for $n = 2$ is given in Fig 4.4. Overall we need

$$\#\text{Diag}_{\text{rot}}(n) = \#\text{Mult}_{\text{rot}}(n) + \#\text{Diag}_{\text{rot}}(n-1) = \sum_{k=2}^{n} 2^{k-1} = 2^n - 2$$

CNOT gates for the implementation of a diagonal operator.

4.1.2 Quantum State Preparation

With Multiplexors

Similarly to the diagonal operator case, a common method for preparing a generic quantum state on n qubits consists in applying a series of operations such that we are left with the preparation of a quantum state on $n-1$ qubits, and we repeat the process until we have to prepare only a one-qubit state. Given as input a unit complex vector $\Psi \in \mathbb{C}^{2^n}$, disentangling the first qubit is for instance equivalent to zeroing the second half of the components of Ψ. To do so, one can apply for each bitstring $s \in F_2^{n-1}$ a specific one-qubit operation U_s on the first qubit such that $U_s(\Psi_{0s}|0s\rangle + \Psi_{1s}|1s\rangle) = \Psi'_{0s}|0s\rangle$. Then the global operator $\bigoplus_s U_s$ can be implemented either by applying successively one R_z-multiplexor and one R_y-multiplexor, both on the first qubit and controlled by the $n-1$ other ones, or by applying one $\mathcal{SU}(2)$-multiplexor, still on the first qubit and controlled by the other qubits [142, 174]. In the case where we only use R_y and R_z multiplexors, we can simply repeat the operation on the $n-1$ remaining qubits. Overall we need to implement two rotation multiplexors on n qubits, two on $n-1$ qubits, etc. The implementation of a rotation multiplexor can be reversed if you take the opposite angle for each rotation in the circuit, so it is possible to cancel two CNOTs for each pair of R_y/R_z multiplexors. Overall one needs a total of

$$\#\text{SP}_{\text{rot}}(n) = \sum_{k=2}^{n} (2 \times \#\text{Mult}_{\text{rot}}(k) - 2) = 2^{n+1} - 2n - 2$$

CNOTs and $2 + 2 \times \sum_{k=2}^{n} 2^{k-1} \approx 2^{n+1}$ elementary rotations to implement an arbitrary state on n qubits.

When using multiplexors in $\mathcal{SU}(2)$, the additional diagonal gate in the synthesis of the multiplexor can be merged with the remaining quantum state as adding phases to each component of the state will not change the number of nonzero elements. So preparing a quantum state with multiplexors in $\mathcal{SU}(2)$ requires to implement one $\mathcal{SU}(2)$-multiplexor on n qubits, one on $n-1$ qubits, etc., without considering the extra diagonal gates, for a total of

$$\#\text{SP}_{\text{su}}(n) = \sum_{k=2}^{n} \#\text{Mult}_{\text{su}}(k) = 2^n - n - 1$$

CNOTs and approximately 2^n generic one-qubit gates. Finally, to have the total count for the number of elementary rotations, we decompose each one-qubit gate U as a product of three elementary rotations (ignoring the global phase) [148]

$$U = R_x(\alpha) \times R_z(\beta) \times R_x(\gamma) \tag{4.3}$$

where α, β, γ are three real parameters. R_x rotations commute with the CNOT gate if the R_x gate acts on the target qubit of the CNOT gate. So for each quantum subcircuit implementing an $\mathcal{SU}(2)$-multiplexor and starting from the leftmost rotation, we can commute the R_x gate, merge it with the next generic one-qubit gate, and repeat the process (the decomposition shown in Eq. (4.3), commutation, and merging) until we reach the last one-qubit gate of the multiplexor implementation. Thus, up to a linear number of gates, all the generic one-qubit gates can be decomposed into only two elementary rotations, for a total of approximately 2^{n+1} rotation gates.

Note that a quantum state on n qubits is completely defined by $2^{n+1} - 2$ real parameters: 2 parameters for each complex entry and the two removed parameters correspond to the arbitrary global phase and the norm of the vector equal to 1. This means that the method for quantum state preparation using either rotation multiplexors or $\mathcal{SU}(2)$ multiplexors is optimal in the number of rotations up to a constant additive factor. The only possible improvement is in the number of CNOT gates.

Note finally that for a real quantum state we can solely use R_y multiplexors and the circuit construction and CNOT complexity are similar to the case where we synthesize a diagonal operator. Yet we can save some CNOT gates if we implement the multiplexors with controlled-Z (CZ) gates replacing the CNOT gates, as done in [174] for the Quantum Shannon Decomposition. We can start the implementation of a R_y multiplexor with a CZ gate but this gate does not modify the number of nonzero elements in the state to synthesize. It only multiplies by -1 some of the entries. These changes of phases can also be handled by changing the angles of some R_y rotations. Therefore we can remove the first CZ gate of each multiplexor and

$$\#\text{SP}_{\text{real, rot}}(n) = \sum_{k=2}^{n} (\#\text{Mult}_{\text{rot}}(k) - 1) = 2^n - n - 1.$$

With Schmidt Decomposition

Schmidt's decomposition expresses a quantum state $|\psi\rangle$ as

$$|\psi\rangle = \sum_{i=0}^{2^r-1} \alpha_i |u_i\rangle \otimes |v_i\rangle \tag{4.4}$$

where $2r \leq n$, $\alpha \in \mathbb{R}^{2^r}$, $\|\alpha\| = 1$, $(u_i)_{0 \leq i < 2^r}$ is an orthonormal set of quantum states on r qubits and $(v_i)_{0 \leq i < 2^r}$ is an orthonormal set of quantum states on $n-r$ qubits. In [155] a method for preparing any state $|\psi\rangle$ is given exploiting Schmidt's decomposition. It consists in three parts:

1. starting from $|0\rangle^r \otimes |0\rangle^{n-r}$, prepare the state $|\alpha\rangle$ on the first r qubits. Note that α is a real vector of unit norm and of size 2^r so preparing α on a register of r qubits is a perfectly valid operation. We get the state
$$\sum_{i=0}^{2^r-1} \alpha_i |i\rangle \otimes |0\rangle^{n-r}.$$

2. apply CNOT gates with control j and target $n-r+j$ for $j = 1..r$ such that we get the state
$$\sum_{i=0}^{2^r-1} \alpha_i |i\rangle \otimes |0\rangle^{n-2r} \otimes |i\rangle.$$

3. apply the r-qubit operator $U = \begin{pmatrix} u_0 & u_1 & \dots & u_{2^r-1} \end{pmatrix}$ on the first r qubits and the $n-r$ qubit operator $V = \begin{pmatrix} v_0 & v_1 & \dots & v_{2^r-1} & \star & \dots & \star \end{pmatrix}$ on the last $n-r$ qubits. We finally get the state
$$\sum_{i=0}^{2^r-1} \alpha_i |u_i\rangle \otimes |v_i\rangle = |\psi\rangle.$$

This construction transforms the synthesis of a state on n qubits to the synthesis of an r-qubit operator and an r to $n-r$ isometry. Given that the synthesis of operators is more studied than the synthesis of isometries, the case where $r = \lfloor n/2 \rfloor$ and $n-r = \lceil n/2 \rceil$ is of particular interest. Namely, if n is even then the synthesis of a state on n qubits is equivalent to the synthesis of two operators on $n/2$ qubits. If n is odd, then we have to synthesize an operator of $(n-1)/2$ qubits and half of an operator on $(n+1)/2$ qubits (an $\lfloor n/2 \rfloor$ to $\lceil n/2 \rceil$ isometry). In terms of resources, the two operators represent the main quantum cost: preparing a state on r qubits only requires $\mathcal{O}(2^r)$ gates, the number of CNOTs in Stage 2 of the procedure is linear in the number of qubits. Synthesizing operators on $n/2$ qubits needs some $\mathcal{O}(4^{n/2}) = \mathcal{O}(2^n)$ gates. Those two operators carry almost all the information on the state to synthesize. An example for $n = 5$ is given Fig 4.5. The total number of CNOTs is given by

$$\#\mathrm{SP}_{\mathrm{Schmidt}}(n) = \#\mathrm{SP}_*(\lfloor n/2 \rfloor) + \lfloor n/2 \rfloor + \#\mathrm{U}_*(\lceil n/2 \rceil) + \#\mathrm{U}_*(\lfloor n/2 \rfloor) \tag{4.5}$$

if one considers the isometry as a full operator. A more precise formula is

$$\#\mathrm{SP}_{\mathrm{Schmidt}}(n) = \#\mathrm{SP}_*(\lfloor n/2 \rfloor) + \lfloor n/2 \rfloor + \#\mathrm{Iso}_*(\lfloor n/2 \rfloor, \lceil n/2 \rceil) + \#\mathrm{U}_*(\lfloor n/2 \rfloor). \tag{4.6}$$

These formulas are a little more general than those given in [155]. One is free to choose any method for state preparation and the unitary/isometry synthesis parts. This is symbolized by the index $*$ in the formulas. In [155] the QSD was used for the synthesis of the unitaries. The QSD produces circuits of size $\frac{23}{48}4^n$, therefore if n is even one can show that the state preparation routine based on Schmidt's decomposition produces circuits of size $\frac{23}{24}2^n$, improving on the method based on $\mathcal{SU}(2)$ multiplexors. A similar complexity was derived later when n is odd in [86]. The authors used a custom QSD for the synthesis of the $\lfloor n/2 \rfloor$ to $\lceil n/2 \rceil$ isometry which requires fewer CNOT gates than simply synthesizing a full operator on $\lceil n/2 \rceil$ qubits. More details are given in the next section.

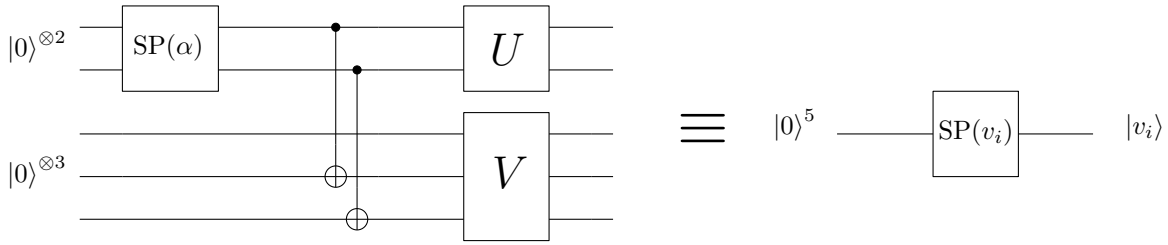


Figure 4.5: Example of state preparation based on Schmidt's decomposition for $n = 5$ qubits.

4.1.3 Quantum Shannon Decomposition

Among the various existing synthesis methods [50, 148, 194], the one giving the shortest circuits in terms of number of gates is the Quantum Shannon Decomposition (QSD) [142, 174]. It relies on the following two decomposition formulas:

- the first one is the Cosine-Sine decomposition (CSD) of a unitary matrix U on n qubits [66]:

$$U = \begin{pmatrix} A_1 & \\ & A_2 \end{pmatrix} \begin{pmatrix} C & -S \\ S & C \end{pmatrix} \begin{pmatrix} B_1 & \\ & B_2 \end{pmatrix}. \tag{4.7}$$

  A_1, A_2, B_1, B_2 are unitary matrices on $n-1$ qubits and C, S are real positive diagonal matrices such that $C^2 + S^2 = I_{2^{n-1}}$. The central term in the CS decomposition is in fact an R_y-multiplexor controlled by the $n-1$ least significant qubits. The circuit equivalence is given in Figure 4.6a, where angles are omitted for legibility.

- The second formula decomposes a multiplexor

$$\begin{pmatrix} A_1 & \\ & A_2 \end{pmatrix} = \begin{pmatrix} V & \\ & V \end{pmatrix} \begin{pmatrix} D^\dagger & \\ & D \end{pmatrix} \begin{pmatrix} W & \\ & W \end{pmatrix} \tag{4.8}$$

  with D a diagonal matrix on $n-1$ qubits, V and W are unitary operating $n-1$ qubits. The central term involving the matrix D is in fact a R_z multiplexor controlled by the $n-1$ least significant qubits. The circuit equivalence is represented in Figure 4.6b, again with omitted angles. The computation of V, W and D can be done, e.g., by diagonalizing the matrix $A_2 A_1^\dagger = V D^2 V^\dagger$. Then $W = D V^\dagger A_1$.

Finally, synthesizing U on n qubits is equivalent to synthesizing 3 rotation multiplexors on n qubits and 4 matrices on $n-1$ qubits on which we can apply the QSD again as shown in Figure 4.6c. We repeat the process until we get only multiplexors and gates acting on a small number of qubits (typically 2) for which an exact decomposition is known [35, 116, 195–197].

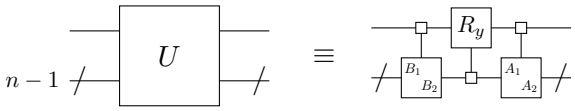
Therefore to have the total number of CNOTs required, we solve the recursive formula

$$c_n = \underbrace{4c_{n-1}}_{\text{the four unitaries on } n-1 \text{ qubits}} + \underbrace{3 \times 2^{n-1}}_{\text{the three multiplexors}}.$$

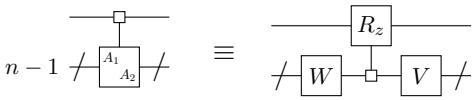
If we stop the recursion at p qubits, one can prove by induction the following expression for c_n:

$$c_n = 4^{n-p}(c_p + 3 \times 2^{p-1}) - 3 \times 2^{n-1}.$$

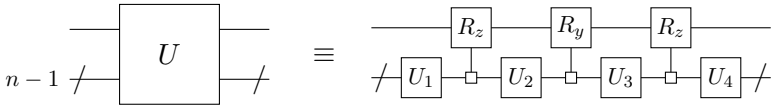
The R_y multiplexor can be implemented with CZ rotations instead of CNOT gates. Depending on if the implementation of the R_y multiplexor begins or ends with a CZ gate, one CZ gate, being



(a) Cosine-Sine Decomposition



(b) Decomposing a multiplexor



(c) Full QSD decomposition

Figure 4.6: Circuit equivalences for the QSD.

diagonal, can be merged with either the operator $\begin{pmatrix} A_1 & \\ & A_2 \end{pmatrix}$ or the operator $\begin{pmatrix} B_1 & \\ & B_2 \end{pmatrix}$. This saves one CZ gate for each R_y multiplexor. In total $(4^{n-p}-1)/3$ CZ gates are saved. If only CNOT gates are available, it is still possible to switch back to CNOT gates as CNOT and CZ gates are equivalent up to Hadamard gates [174]. Overall, the total number of CNOT gates is given by

$$\#\mathrm{U}_{\mathrm{QSD}}(n) = \frac{\#\mathrm{U}(p) + 3 \times 2^{p-1} - 1/3}{4^p} \times 4^n - 3 \times 2^{n-1} + \frac{1}{3}. \tag{4.9}$$

Lastly, when $p = 2$, the two-qubit operators can be implemented as the concatenation of a diagonal operator and a two-qubit operator that only needs 2 CNOT gates. Then, starting from the left-most two-qubit operator, the diagonal gate can commute with the controls of the multiplexors and merge with the next operator. Pursuing until we reach the last operator, every two-qubit operator but the last can be implemented with only 2 CNOT gates. Therefore one can save $4^{n-2}-1$ additional CNOT gates or equivalently one can replace $p = 2, \#\mathrm{U}(p) = 2$ and add 1 to get the current best CNOT count for circuit synthesis:

$$\#\mathrm{U}_{\mathrm{QSD}}(n) = \frac{23}{48}4^n - 3 \times 2^{n-1} + \frac{4}{3}. \tag{4.10}$$

Assuming that we can synthesize an operator on p qubits up to a diagonal gate with $\#\mathrm{U}_{\text{-diag}}(p)$ gates, we have a more general formula than the one given in [174]:

$$\#\mathrm{U}_{\mathrm{QSD}}(n) = \alpha_p \times 4^n - 3 \times 2^{n-1} + \frac{1}{3} + \delta_p \tag{4.11}$$

where

$$\alpha_p = \frac{\#\mathrm{U}_{\text{-diag}}(p) + 3 \times 2^{p-1} - 1/3}{4^p},$$
$$\delta_p = \#\mathrm{U}(p) - \#\mathrm{U}_{\text{-diag}}(p).$$

Application to Isometry Synthesis and State Preparation

We can combine the QSD with the state preparation scheme using Schmidt's decomposition. We get new equations that cannot be found in the literature. Using Eq (4.9) in Eq (4.5) we have:

$$\#\mathrm{SP}_{\text{Schmidt, QSD}}(n) = \alpha_p \times \left(4^{\lceil n/2 \rceil} + 4^{\lfloor n/2 \rfloor}\right) - \frac{3}{2}\left(2^{\lceil n/2 \rceil} + 2^{\lfloor n/2 \rfloor}\right) + 2^{\lfloor n/2 \rfloor} - \frac{1}{3} + 2\delta_p. \tag{4.12}$$

However, this represents an improvement over the method using $\mathcal{SU}(2)$ multiplexors only when n is even. When n is odd we have to synthesize an $r-1$ to r isometry with $r = \lceil n/2 \rceil$. This synthesis can be done more efficiently by using a tailored QSD as done in [86]. Namely, in the synthesis of an $r-1$ to r isometry we only want to synthesize the first 2^r columns of an operator of size 2^{r+1}. We can equivalently assume that the first qubit is in the fixed state $|0\rangle$. When applying the CSD decomposition one can see that the first multiplexor $\begin{pmatrix} B_1 & \\ & B_2 \end{pmatrix}$ from Eq (4.7) needs not to be implemented completely as the control qubit is in the fixed state $|0\rangle$. In other words, we only need to implement the operator B_1 on the last $r-1$ qubits, plus the usual R_y multiplexor and the second multiplexor $\begin{pmatrix} A_1 & \\ & A_2 \end{pmatrix}$. Overall the synthesis on an $r-1$ to r isometry is equivalent to the synthesis of two rotation multiplexors on r qubits and three operators on $r-1$ qubits. We have

$$\#\text{Iso}(r-1, r) = 3 \times \#\text{U}(r-1) + 2\#\text{Mult}_{\text{rot}}(r).$$

And using the QSD for the synthesis of the three operators we get

$$\#\text{Iso}_{\text{QSD}}(r-1, r) = \frac{3}{4}\alpha_p 4^r - \frac{5}{4}2^r + 1 + 3\delta_p. \tag{4.13}$$

Combining Eq. (4.13) and Eq. (4.6), we finally get

$$\#\text{SP}_{\text{Schmidt, QSD}}(n) = \alpha_p \times \left(4^{\lfloor n/2 \rfloor} + \frac{3}{4}4^{\lceil n/2 \rceil}\right) - \frac{1}{2}2^{\lfloor n/2 \rfloor} - \frac{5}{4}2^{\lceil n/2 \rceil} + \frac{1}{3} + 4\delta_p. \tag{4.14}$$

when n is odd. To summarize:

- if n is even, then

$$\#\text{SP}_{\text{Schmidt, QSD}}(n) = 2\alpha_p \times 2^n - 2 \times 2^{n/2} - \frac{1}{3} + 2\delta_p. \tag{4.15}$$

  With $p = 2, \alpha_p = 23/48, \delta_p = 1$ we recover the result from [155].

- if n is odd, then

$$\#\text{SP}_{\text{Schmidt, QSD}}(n) = 2\alpha_p \times 2^n - 3 \times 2^{(n-1)/2} + \frac{1}{3} + 4\delta_p. \tag{4.16}$$

  With $p = 2, \alpha_p = 23/48, \delta_p = 1$ we recover the result from [86].

Classical Resource Estimation

If the Quantum Shannon Decomposition (QSD) is the method giving the best asymptotic number of CNOTs in the circuit so far: $\frac{23}{48} \times 4^n$, one result of this thesis is to explicitly show that this method has nonetheless a drawback: the method is very costly in terms of classical resources. What follows is a summary of how the QSD can be implemented with an estimation of the total flops count. Notably, the algorithm used for computing the Cosine-Sine Decomposition (CSD) is taken from [185]. This algorithm is now part of LAPACK (routine ZUNCSD) and has already been used in other implementations [87, 200].

The algorithm for computing Formula (4.7) consists in reducing the $K \times K$ matrix U into a 2×2 bidiagonal block form. Then the 4 bidiagonal blocks are simultaneously diagonalized using bidiagonal SVD algorithms. The first part is the most expensive one in terms of floating-point operations:

by applying Householder reflectors to the left and right of U, we progressively bidiagonalize U — this requires $K^3/3$ flops for each block — and we store the accumulation of each Householder reflector to compute A_1, A_2, B_1, B_2 — this requires $K^3/6$ flops for each block. Overall, computing the CSD on a $K \times K$ matrix requires $2 \times K^3$ flops. Concerning Formula (4.8), we follow the methodology given in [174]. One has to perform two matrix/matrix products and an eigenvalue decomposition. With square matrices of size K, each matrix/matrix product requires $2 \times K^3$ flops and the eigenvalue decomposition needs around $26 \times K^3$ [24, Table 3.13]. Overall for the first step of the Quantum Shannon Decomposition applied to a matrix of size N we have to compute:

- one CSD of a matrix of size N,
- decompose two multiplexors, i.e., four matrix/matrix products of size $N/2$ and two eigenvalue decompositions of size $N/2$ too,
- decompose three rotation multiplexors, this part can be computed with a single matrix/vector product and is negligible.

This represents a total of $2 \times N^3 + 4 \times 2 \times (N/2)^3 + 2 \times 26 \times (N/2)^3 = \frac{19}{2} \times N^3$ flops. To pursue the algorithm we have to perform the same operations on 4 matrices of size $N/2$, then on 16 matrices of size $N/4$, etc. Overall, with $N = 2^n$ we can approximate the total number of flops to 19×8^n which is very expensive. Standard QR factorizations only require about N^3 flops, so we may expect other synthesis methods to be much more efficient in terms of classical time complexity.

4.1.4 Other Synthesis Methods

We just saw in the previous section that the QSD has a large overhead in its classical time complexity. Cheaper methods, even though they produce less optimal circuits, are of interest if they are able to synthesize unitaries on a larger number of qubits. In this section, we explore other methods in the literature and we evaluate their complexities in terms of time complexity and gate counts.

We can distinguish two classes of methods for quantum circuit synthesis:

1. the methods that synthesize the operator column by column, i.e., given an operator $U \in \mathcal{U}(2^n)$ we compute a sequence of gates such that we are left with the synthesis of an operator $V \in \mathcal{U}(2^n - 1)$ and we repeat the process. QR methods typically lie in this class.
2. the methods that transform the synthesis on an n-qubit operator into the synthesis of several $(n-1)$-qubit operators. The QSD is such an example.

QR Decompositions

A QR decomposition based on Givens rotations is proposed in several works [3, 43, 194]. Givens rotations are 2-level unitary matrices that act on only two basis states. One rotation can, therefore, be used to zero one specific entry of the operator. With $\mathcal{O}(4^n)$ such rotations one can reduce an operator U to the identity by zeroing iteratively each column of U. By doing so the zeroed entries will not be modified again in the QR factorization.

Most of the work done to improve these methods is about the implementation of the Givens rotations into quantum circuits and in which order should we implement them to reduce the gate count the most. We will not detail it here but the authors in [194] gave an estimation of the average number of gates in the circuits: 8.7×4^n with an upper bound of 11×4^n. This method produces

circuits that are between 16 and 20 times larger than the QSD. Such overhead is too important to retain this method. Besides, although this is an a posteriori argument, the use of Givens rotations is not recommended compared to the use of Householder reflections — like our method does, see Section 4.3 — when the operator is *dense*, which is the case if we assume that our operator is generic.

For sparse operators, on the other hand, a method based on Givens rotations is relevant. But we do not focus on sparse operators in this chapter. For all these reasons we do not add this method in our benchmarks.

Another QR decomposition can be found in [86]. They also synthesize the operator column by column with the use of multiplexors and multi-controlled gates that prevent the zeroed columns to be modified while zeroing the next ones. In [87] both time and gate complexities are given for the synthesis of an m to n isometry:

- the method produces circuits with $2^{m+n} - \frac{1}{24}2^n + \mathcal{O}(n^2)2^m$ CNOTs,
- the classical time complexity is $\mathcal{O}(n2^{2m+n})$.

With this method, we have the inverse problem compared to the QR factorization based on Givens rotations: the CNOT count is close to that of the QSD but the classical run time has an extra factor of n. For general unitary synthesis with $n = m$, this method cannot be efficient compared to the QSD.

Knill Decomposition

An in-between method is given by Knill's decomposition [110]. It is not directly based on a QR factorization but it does not decompose U into operators on fewer qubits neither. Given $U = PDP^\dagger$ an eigenvalue decomposition of U, we have

$$U = \prod_{i=1}^{2^n} P_i \Lambda_{n-1}(Ph(D_i)) P_i^\dagger$$

where:

- P_i is an operator that prepares the state $P[i,:]$, i.e., the i-th eigenvector of U, and $P_i |0\rangle = P[i,:]$.
- $D_i = e^{i\theta_i}$ for some real θ_i is the i-th diagonal entry of D, i.e., the i-th eigenvalue of U,
- $\Lambda_{n-1}(U)$ stands for an operator U negatively-controlled by the last $n-1$ qubits (see Section 2.2.2 for a definition of a negatively-controlled operator),
- $Ph(\theta) = \begin{pmatrix} \theta & 0 \\ 0 & 1 \end{pmatrix}$.

This decomposition is used in [86] and [87]. With some optimizations, the number of CNOT gates depends on the parity of n. Again the complexity is given for an m to n isometry:

- if n is even the method produces circuits with

  $$\frac{23}{24}(2^{m+n} + 2^n) + \frac{23}{12}2^{m+n/2} - 2^{m+n/4+2} + \mathcal{O}(n^2)2^m$$

  CNOTs,

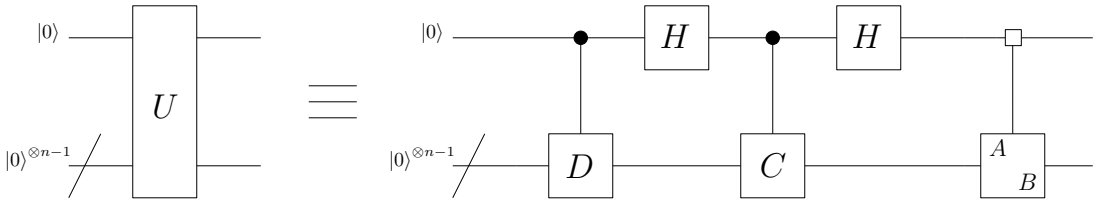


Figure 4.7: Block-ZXZ decomposition of an operator U.

- if n is odd the method produces circuits with

$$\frac{115}{96}(2^{m+n}+2^n)+\frac{23}{12}2^{m+(n-1)/2}-2^{m+(n-1)/4+2}+\mathcal{O}(n^2)2^m$$

  CNOTs.

So the method produces circuits twice larger than the QSD, which is of interest if the run time is much lower. Computing the complete eigenvalue decomposition (eigenvalues and eigenvectors) requires about $26N^3$ flops for an $N \times N$ matrix. In our case, we need 26×8^n flops. Synthesizing a state requires $\mathcal{O}(2^n)$ flops, so the synthesis of 2^n states requires $\mathcal{O}(4^n)$ flops which is negligible compared to the eigenvalue decomposition. Overall the time complexity is dominated by the eigenvalue decomposition and is more costly than the QSD requiring 19×8^n flops.

4.1.5 Other Decomposition Schemes

The QSD is the best method among those who decompose the synthesis into syntheses of operators acting on a smaller number of qubits. It is, in fact, the culmination of the works based on the CSD. Apart from that, the only decomposition using a similar technique is the block-ZXZ decomposition [50]. The n-qubit operator U is decomposed as

$$U=\frac{1}{2}\begin{pmatrix}A & 0\\ 0 & B\end{pmatrix}\begin{pmatrix}I+C & I-C\\ I-C & I+C\end{pmatrix}\begin{pmatrix}I & 0\\ 0 & D\end{pmatrix}$$

where A, B, C, D are operators acting on $n-1$ qubits and I is the identity operator. Given that $\begin{pmatrix}I+C & I-C\\ I-C & I+C\end{pmatrix} = (H\otimes I)\begin{pmatrix}I & 0\\ 0 & C\end{pmatrix}(H\otimes I)$ we get the decomposition scheme given in Fig 4.7. Then the decomposition can be applied recursively on A, B, C, D.

Actually this decomposition is not far from the QSD. By applying the multiplexor decomposition given by Eq. (4.8) on the operators $\begin{pmatrix}A & 0\\ 0 & B\end{pmatrix}$, $\begin{pmatrix}I & 0\\ 0 & C\end{pmatrix}$ and $\begin{pmatrix}I & 0\\ 0 & D\end{pmatrix}$ and merging some operators, it is easy to see that we recover the QSD decomposition.

4.2 Contributions

A summary of the state-of-the-art methods (flop count and CNOT count) is given in Table 4.1.

The QSD appears to be the best method for synthesizing small circuits but also the quickest method. This is quite surprising as we highlighted that the QSD is, in fact, costly in terms of classical resources. The only method in the literature that may be less expensive than the QSD is the QR factorization based on Givens rotations but we saw that the size of the circuits produced is too large to be used. Clearly, it lacks a quick synthesis method that produces circuits of reasonable

Method	Asymptotic CNOT count	flop count
QSD [142, 174]	$\mathbf{\frac{23}{48} \times 4^n}$	$\mathbf{19 \times 8^n}$
QR (Givens) [194]	$\approx 8.7 \times 4^n$	$\mathcal{O}(8^n)$
Column by Column [86]	4^n	$\mathcal{O}(n8^n)$
Knill's Decomposition [86, 110]	$\begin{cases} \frac{23}{34} \times 4^n \text{ if } n \text{ is even} \\ \frac{115}{96} \times 4^n \text{ if } n \text{ is odd} \end{cases}$	26×8^n

Table 4.1: Summary of the state-of-the-art methods for generic quantum circuit synthesis. The best results for the CNOT count and the flop count are in bold.

size. In response to this, we propose a method based on Householder transformations. Strongly connected to classical results about QR decomposition, this method aims at achieving better performance in the synthesis of quantum circuits by finding a compromise between circuit size and calculation time.

The main contributions of the chapter are as follows.

- We adapt the well known and numerically stable QR factorization based on Householder transformations [66] to the factorization of unitary matrices. The adaptation heavily relies on the specific structure of unitary matrices. We exhibit a significant theoretical and practical speedup of our specific QR algorithm compared to the unmodified QR routine and the usual technique for quantum circuit synthesis based on the quantum Shannon decomposition (QSD) [174].

- We propose a complete circuit synthesis method using this specialized QR decomposition with a complexity analysis for circuit size and arithmetical operations. If some existing theoretical and experimental works for quantum circuits synthesis with Householder transformations have been undertaken [37, 88, 190], to our knowledge none has proposed an implementation method and a final circuit construction with clearly defined properties. Overall, our technique is faster than the QSD-based method while providing circuits only twice as large[1].

- We back up our approach with benchmarks on multicore and GPU architectures for random unitary matrices operating on up to 15 qubits.

These results have been published in Elsevier's journal *Computer Physics Communications* [47].

The plan of the chapter is as follows. In Section 4.3 we detail the new adapted Householder algorithm. We explain in Section 4.4 how to convert this factorization into a quantum circuit. Section 4.5 presents the performance obtained on multicore and GPU architectures by our algorithm. We also compare our results with a reference algorithm based on the Quantum Shannon Decomposition method.

[1]This extra cost in the final quantum circuit is not negligible, especially when considering the current limitations of the quantum hardware. It may be possible that the gain in the classical process will not compensate the execution time of the twice as large quantum circuit on real hardware. However, we can handle problem sizes that were unreachable before with the QSD, regardless of the quality of the hardware. We believe our approach highlights the tradeoffs between two measures of complexity (circuit size/compilation time) and that this has to be taken in consideration when synthesizing generic quantum circuits.

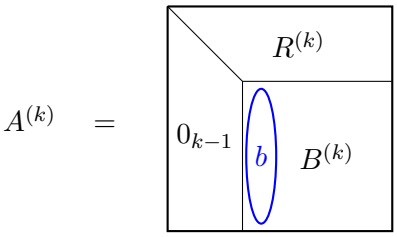


Figure 4.8: Matrix pattern at step k-th of Householder transformation.

4.3 A QR Algorithm for Unitary Matrices with Householder Transformations

In this section, we first recall the main principles of the QR factorization of a general complex square matrix via Householder transformations. Then we consider the special case of unitary matrices.

The QR decomposition of a matrix $A \in \mathbb{C}^{n\times n}$ expresses A as the product of a unitary matrix $Q \in \mathcal{U}(n)$ and an upper triangular matrix R. A standard algorithm to compute such a factorization consists in applying a series of Householder transformations [66, p. 209] zeroing out successively the subdiagonal entries of each column.

At step k ($1 \leq k \leq n-1$) of the QR algorithm, we zero out all but the first entry of the vector b in the matrix depicted in Figure 4.8 using the Householder transformation, $H'_k = I_n - \tau_k u_k u_k^\dagger$, where $u_k \in \mathbb{C}^{n-k+1}$ and $\tau_k = 2/u_k^\dagger u_k$. Note that in the complex case, the Householder matrix H'_k can be sometimes referred to as "elementary unitary matrix" (e.g. in [118]).

Then the k-th iteration ends with the computation of the matrix

$$A^{(k)} = H_k A^{(k-1)},$$

with $H_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & H'_k \end{pmatrix}$, $A^{(0)} = A$ and $H_1 = H'_1$. This operation updates $B^{(k)} = A^{(k)}(k:n, k:n)$ via the relation

$$\left(I_{n-k+1} - \tau_k u_k u_k^\dagger\right) B^{(k)} = B^{(k)} - \tau_k u_k \left(\left(B^{(k)}\right)^\dagger u_k\right)^\dagger, \tag{4.17}$$

which zeroes out the subdiagonal entries in column k (but does not affect the zeros already introduced in previous columns). The problem is now to find the vector $u_k \in \mathbb{C}^{n-k+1}$ such that

$$\left(I_{n-k+1} - \tau_k u_k u_k^\dagger\right) b = (\beta_k, 0, \ldots, 0)^T = \beta_k e_1.$$

with $\beta_k \in \mathbb{C}$. From [66, p. 233], we have $u_k = b \pm e^{i\theta}\|b\|e_1$ with $\theta = \arg(b_1)$, but various choices for u_k have been proposed in numerical libraries (see [118] for a review of these choices).

At the end of the algorithm we have computed a set of $n-1$ Householder transformations $H_1, H_2, ..., H_{n-1}$ such that

$$\left(\prod_{i=1}^{n-1} H_{n-i}\right) A = R$$

where R is upper triangular. Since the Householder matrices are Hermitian, we obtain

$$A = \left(\prod_{i=1}^{n-1} H_i\right) R = QR.$$

In the QR algorithm, the Householder matrices H'_k never need to be explicitly formed and the expensive part of the computation is the update of the matrix $B^{(k)}$, given in Equation (4.17), which requires at each iteration a matrix-vector multiplication followed by a rank-1 update of $B^{(k)}$. The total cost of the factorization is about $\frac{4}{3}n^3$ complex flops ($\frac{16}{3}n^3$ real flops).

A block version of the algorithm uses the fact that a product of p Householder matrices $H_1 \times \ldots \times H_p$ can be written as $I - VTV^\dagger$ where V is an $n \times p$ rectangular matrix with the Householder vector $u_k \in \mathbb{C}^n$ at the k-th column and T is an upper triangular matrix [184]. The algorithm consists in partitioning A into blocks of size $n \times n_A$ for some n_A, factorizing the first block and updating the remaining blocks via the operation (we use a matlab-like notation)

$$A(:, n_A + 1 : n) = A(:, n_A + 1 : n) - VTV^\dagger A(:, n_A + 1 : n),$$

and repeating the process with the next block until the whole matrix is triangularized. The update is richer in BLAS 3 operations [51], potentially leading to better performance, yet without decreasing the flop count [171].

Let us now exploit the specificity of quantum operators where the corresponding matrix A is unitary and see how the QR decomposition simplifies. In this case, the triangular factor is also unitary and thus diagonal and the QR algorithm of A consists in a progressive diagonalization of A. For sake of simplification, we detail in the remainder only the first iteration. Let $b = (b_1, \ldots, b_n)^T$ be the first column of the unitary matrix A and $r = (r_1, \ldots, r_n)$ its first row. We choose the value of the Householder vector $u = b \pm e^{i\theta}\|b\|e_1$ as defined in [66, p. 233] but we will choose the sign "+". This choice has the advantage of maximizing $\|u\|$ (for sake of stability [66, p. 233]) and of simplifying the final decomposition of the quantum operator into elementary circuits, as we will see in Section 4.4. Since A (and $A^{(k)}$ at the kth iteration) is unitary, we have $\|b\| = 1$ and we get

$$u = b + e^{i\theta}e_1$$

and

$$\tau = \frac{2}{\|u\|^2} = \frac{2}{\|b\|^2 + \|e_1\|^2 + 2|b_1|} = \frac{1}{1 + |b_1|}.$$

Then applying the Householder transformation H to b gives

$$Hb = -e^{i\theta}\|b\|e_1 = -e^{i\theta}e_1. \tag{4.18}$$

The gain in flops complexity occurs in the update phase. Using the orthonormality of the vectors of A, the update expressed in Equation (4.17) simplifies to

$$HA = A - \tau(b + e^{i\theta}e_1)\left(A^\dagger\left(b + e^{i\theta}e_1\right)\right)^\dagger = A - \tau(b + e^{i\theta}e_1)(e_1^T + e^{-i\theta}r).$$

Then we have

$$HA = A - \tau(be_1^T + e^{i\theta}e_1e_1^T + e_1r + e^{-i\theta}br).$$

The first column of A does not need to be updated in this computation because Equation (4.18) implies that we can ignore the term $be_1^T + e^{i\theta}e_1e_1^T$. Similarly the first row of A does not need to be updated because the unitarity of the rows of A ensures that $r = -e^{i\theta}e_1^T$ after application of H. Moreover $\tau e^{-i\theta} = 1/(e^{i\theta} + |b_1|e^{i\theta}) = 1/(b_1 + e^{i\theta}) = 1/u_1$, where u_1 denotes the first component of u. So we are left with the rank-1 update

$$(HA)_{2:n,2:n} = A_{2:n,2:n} - \frac{b(2:n) \cdot r(2:n)}{u_1}.$$

The matrix-vector product expressed in Equation (4.17) for the classical QR factorization is avoided. The matrix obtained after the first iteration is then

$$A^{(1)} = \begin{pmatrix} -e^{i\theta} & 0 \\ 0 & (HA)_{2:n,2:n} \end{pmatrix}$$

and we can continue the algorithm on the unitary matrix $A^{(1)}(2:n, 2:n)$ and so on, until A becomes diagonal. The update at the k-th iteration requires only $(n-k)^2$ multiplications and $(n-k)^2$ additions. Finally this new algorithm requires $\sum_{k=1}^{n-1} 2 \times (n-k)^2 \sim \frac{2}{3}n^3$ complex flops, which is twice as less than the standard case.

It is possible to choose the vector u such that $u_1 = 1$, then the value of τ will be adjusted so that the resulting Householder transformation H remains the same. More precisely, keeping the notations above we set

$$u \leftarrow \frac{1}{e^{i\theta}(1+|b_1|)} u \tag{4.19}$$

and we obtain $\tau = (1+|b_1|)$ and then the update phase becomes

$$(HA)_{2:n,2:n} = A_{2:n,2:n} - u(2:n) \cdot r(2:n). \tag{4.20}$$

The algorithm can easily be done in place. One can store the Householder vectors in the strictly lower triangular part of A, the diagonal elements of R are stored in the diagonal and the τ_i's are stored in a specific array.

The main cost of the algorithm resides in the rank-one update phase in Equation (4.20). In order to use more optimized BLAS 2 and BLAS 3 operations we can derive from Equation (4.20) new update relations. Suppose we have already performed the factorization and the update for the first nb rows and columns for some nb. Therefore the first nb columns of A contain the Householder vectors, and the block $A(1:nb, nb+1:n)$ has been updated following (4.20). Let $i, j \in [\![nb+1, n]\!]$, one can verify that the update of the element $A(i,j)$ is given by

$$A(i,j) \leftarrow A(i,j) - \sum_{k=1}^{nb} A(i,k) \times A(k,j) \tag{4.21}$$

by simply applying successively the update (4.20).

In terms of matrix and vector operations we have

$$A(i, nb+1:n) \leftarrow A(i, nb+1:n) - A(i, 1:nb) \times A(1:nb, nb+1:n) \tag{4.22}$$

for the update of one row,

$$A(nb+1:n, j) \leftarrow A(nb+1:n, j) - A(nb+1:n, 1:nb) \times A(1:nb, j) \tag{4.23}$$

for the update of one column and

$$\begin{aligned} &A(nb+1:n, nb+1:n) \leftarrow \\ &\qquad A(nb+1:n, nb+1:n) - A(nb+1:n, 1:nb) \times A(1:nb, nb+1:n) \end{aligned} \tag{4.24}$$

for the update of the full matrix. This last update is a BLAS 3 operation and can potentially yield higher performance on hybrid CPU-GPU architectures [187].

Using these new update relations we can improve the algorithm by three means:

- First we can improve the unblocked algorithm. Instead of updating the whole matrix at each iteration with a rank one update we only update one row and one column: at the k-th iteration we have computed the k-th Householder vector and we update the row $A(k+1, k+1:n)$ and the column $A(k+1:n, k+1)$ via the relations (4.22) and (4.23). Such updates consist in more and bigger matrix-vector operations and experimentally it appears to scale better.

- Secondly this naturally leads to a blocked version of the algorithm. Let nb be the size of our block. Once we have done the computations on the first nb rows and nb columns of A with an unblocked version, we can update the rest of the matrix with a matrix/matrix product via equation (4.24) and continue the algorithm on the matrix $A(nb+1:n, nb+1:n)$ until we reach the last block where the unblocked algorithm is applied.

- A third improvement can be made in order to avoid using the unblocked algorithm to compute the full panel of nb rows and columns of A. Indeed the update of the blocks $A(nb+1:n, 1:nb)$ and $A(1:nb, nb+1:n)$ can be performed with BLAS 3 operations. One can prove that there exist triangular matrices T_1^i, T_2^i of size $i \times i, i = 1..nb$ such that

$$A(i+1:n, 1:i) \leftarrow A(i+1:n, 1:i) \times T_1^i \tag{4.25}$$

$$A(1:i, i+1:n) \leftarrow T_2^i \times A(1:i, i+1:n). \tag{4.26}$$

The matrices T_1^i, T_2^i are computed using the following recursive formula:

$$T_1^1 = 1, T_1^{i+1} = \begin{pmatrix} T_1^i & -p_{i+1}T_1^i/\mathcal{N}_i \\ & 1/\mathcal{N}_i \end{pmatrix}, \tag{4.27}$$

$$T_2^1 = 1, T_2^{i+1} = \begin{pmatrix} T_2^i & \\ -q_{i+1}T_2^i & 1 \end{pmatrix} \tag{4.28}$$

with $p_i = A(1:i, i), q_i = A(i, 1:i)$. $\mathcal{N}_i$ is the normalization factor expressed in Equation (4.19).

Proof of Formulas (4.25) to (4.28). By induction on i. We do it for T_2 only. The case $i = 1$ is trivial because we do not have to update the first row. Now suppose the result is true for some i, $1 \leq i < nb$. The first i rows are already updated by the application of T_2^i, we only need to update the next row $i+1$. Let $A(i+1, j), j \in [\![nb+1, n]\!]$ be an element of this row. If the column $A(1:i, j)$ was already updated the update of $A(i+1, j)$ would be given by the equation (4.21), i.e.,

$$A(i+1, j) \leftarrow A(i+1, j) - \sum_{k=1}^{i} A(i+1, k) \times A(k, j).$$

Written differently $A(i+1, j) \leftarrow A(i+1, j) - A(i+1, 1:i+1) \cdot A(1:i, j)$. By hypothesis $A(1:i, j)$ is updated by the relation $A(1:i, j) \leftarrow T_2^i A(1:i, j)$. This gives

$$A(i+1, j) \leftarrow [-A(i+1, 1:i+1) \times T_2^i \,;\, 1] \cdot A(1:i+1, j).$$

Doing it for all j and concatenating it with the update of the first i rows by the action of T_2^i gives the result. The same proof can be done with T_1 but one has to be careful about the normalization factors of the Householder vectors. □

Therefore T_1^{nb} and T_2^{nb} only depend on the block $A(1 : nb, 1 : nb)$ and can be used to update $A(nb + 1 : n, 1 : nb)$ and $A(1 : nb, nb + 1)$ in two BLAS 3 updates. This means that during an iteration we only need to perform an unblocked Householder factorization on a square matrix of size nb and then perform 3 BLAS 3 updates. The pseudo code of the algorithm is given in Algorithm 4.1 (we call the corresponding routine ZUNQRF and its unblocked version ZUNQR2). ZLARFT2 refers to the adaptation of the standard ZLARFT routine that computes the triangular matrices. Note that the factorization routine aims at being integrated in the public domain libraries LAPACK and MAGMA, in collaboration with the University of Tennessee.

Algorithm 4.1: Householder factorization of a unitary matrix A - ZUNQRF.

Require: $N \geq 0$, A$\in \mathcal{U}_N$
Ensure: $A = QR$
// NX determines when to switch from blocked to unblocked code
// NB is the block size
for $I = 1, NX, NB$ **do**
  $IB \leftarrow MIN(N - I + 1, NB)$
  call ZUNQR2(IB, IB, A(I, I), TAU(I))
  $T_1, T_2 \leftarrow$ ZLARFT2(N, IB, A(I, I), TAU(I))
  update $A(I : N, I : I + IB)$ via a call to ZTRMM
  if $I + IB \leq N$ **then**
    update $A(I : I + IB, I : N)$ via a call to ZTRMM
    update $A(I + IB : N, I + IB : N)$ via a call to ZGEMM
  end if
end for
if $I \leq N$ **then**
  call ZUNQR2(N-I+1, N-I+1, A(I, I), TAU(I))
end if

Thanks to the above QR decomposition resulting in a product of Householder matrices and a diagonal matrix, we store the information of a unitary matrix into the subdiagonal part of the complex matrix (the Householder vectors), and two real vectors, including the θ's (angles of the diagonal entries) and the τ's. In Section 4.4 we use this factorization of unitary operators to obtain quantum circuits.

4.4 From the Householder Decomposition to a Quantum Circuit

In this section we develop several methods to convert the Householder representation of a unitary matrix into a quantum circuit. We present a general method in the Section 4.4.1 and we optimize it in Section 4.4.2.

4.4.1 General Method

Let $U \in U(2^n)$ be the unitary matrix we want to synthesize. The QR factorization of U gives normalized vectors $u_1, u_2, \ldots, u_{2^n-1}$ and a diagonal matrix D such that

$$U = \prod_{i=1}^{2^n-1} H_i \times D.$$

where H_i are Householder matrices defined by $H_i = I_{2^n} - 2u_i u_i^\dagger$ as in Section 4.3 (since the u_i are normalized). The synthesis of a diagonal operator is a well-known problem [36, 203]. Therefore, the main issue is the synthesis of the Householder matrices. We recall that

$$H_i u_i = -u_i$$

and

$$\forall v, v \perp u_i \Rightarrow H_i v = v.$$

Consequently, for any unitary matrix P_i whose first column is u_i we can write

$$H_i = P_i D_G P_i^\dagger \tag{4.29}$$

with

$$D_G = \begin{pmatrix} -1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix}.$$

Indeed P_i can be regarded as an orthonormal basis of vector columns containing u_i. In other words, Equation (4.29) is a diagonalization of H_i.

From this analysis, we get the following decomposition of the unitary matrix U

$$U = \prod_{i=1}^{2^n-1} P_i D_G P_i^\dagger \times D \tag{4.30}$$

and we can derive a quantum circuit as depicted in Figure 4.9.

- As mentioned already, the synthesis of D is a problem with known solutions.
- Each block $P_i D_G P_i^\dagger$ is equivalent to un-preparing the state u_i, applying D_G and repreparing the state u_i.
  - The matrix D_G is the "zero phase shift" operator and is used for instance in the Grover diffusion operator in Grover's algorithm [71]. Up to X operators this operator is equivalent to the multi-controlled Z gate. So, up to X and H operators, D_G is equivalent to a generalized Toffoli gate. Many different implementations of the generalized Toffoli gate exist in the literature [17, 78, 91, 131, 167]. In our calculations we write $\#\text{Toff}_{[78]}(n)$ the CNOT complexity of the generalized Toffoli gate on n qubits. For numerical applications we authorize ourselves the use of one ancillary qubit and we use the circuit from [78]. According to [78], we can synthesize a generalized Toffoli gate on n qubits ($n-1$ controls) with
    $$\#\text{Toff}_{[78]}(n) = 24n - 72$$
    CNOT gates if $n > 4$, otherwise we need 1 CNOT for $n = 2$ and 6 CNOTs for $n = 3$.
  - In our circuits we use the notation $SP(v)$ to refer to a black box that prepares the state v. Although many different operators can prepare the state v, we point out that in one circuit the operators preparing and un-preparing the same state are exactly the same, otherwise the decomposition would not be valid. Many previous research studies have sought to optimize the preparation of states and we use their results for our synthesis [142, 155, 174].

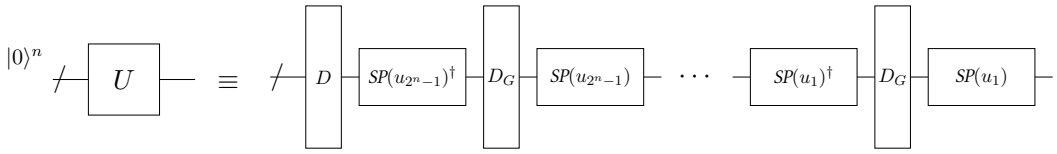


Figure 4.9: A first circuit for the Householder method.

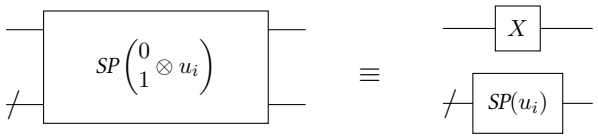


Figure 4.10: Desentangling one qubit in state preparation.

4.4.2 Resources Estimation

We now turn to the question of the size of the circuit sketched in Figure 4.9 (measured in number of CNOTs), and to the computational cost to generate the circuit (measured in flops). Asymptotic results are summarized in Tables 4.2 and 4.3.

Using naively Eq (4.30), the number of CNOT gates is

$$\#\mathrm{U}_{\mathrm{Hous.}}(n) = \#\mathrm{Diag}_{\mathrm{rot}}(n) + \sum_{k=1}^{2^n-1} 2\#\mathrm{SP}(n) + \#\mathrm{Toff}_{[78]}(n).$$

The complexity of the size of the circuit is therefore essentially due to the preparation and un-preparation of quantum states. Because of the structure of the problem, we can do better than systematically applying state preparation on n qubits for each of the u_i vectors. We describe two successive optimizations. The first one relies on the possibility to perform state preparations on less than n qubits; the second one proposes to fuse adjacent sequences of un-preparations and preparation of states.

Optimization Based on State Preparation

When preparing u_i, if only the last 2^k elements of $u_i \in \mathbb{C}^{2^n}$ are non zero, the state is encodable on k qubits only. This means that a k-qubit operator can prepare the state $u_i' \in \mathbb{C}^{2^k}$ such that $u_i = \left(\begin{smallmatrix} 0 \\ u_i' \end{smallmatrix}\right)$. Let Q be such an operator, then the operator

$$P = X^{\otimes(n-k)} \otimes Q = \begin{pmatrix} (0) & & 1 \\ & \iddots & \\ 1 & & (0) \end{pmatrix} \otimes \begin{pmatrix} u_i' & (*) \\ & \end{pmatrix} = \begin{pmatrix} 0 & * \\ u_i' & * \end{pmatrix}$$

prepares u_i. A quantum circuit to illustrate this is given in Figure 4.10.

Thus, up to the operators X, D and D_G, we observe that the Householder method breaks down as follows: the synthesis of the first 2^{n-1} columns is done via operators acting on n qubits, then the next 2^{n-2} columns are synthesized with operators acting on $n-1$ qubits, *etc.*

To simplify the calculations, we use the concept of isometries introduced in [86]. We recall that, with $n > m$, an m to n qubits isometry can be represented as a $2^n \times 2^m$ matrix V such that

$$V^\dagger V = I_{2^m \times 2^m}.$$

The data of the 2^m first columns of an operator on n qubits can be regarded as such an isometry V. For instance an isometry from 0 to n qubits is a quantum state on n qubits. Synthesizing an isometry from $n-1$ to n qubits is equivalent to synthesizing the first 2^{n-1} columns of an n-qubit operator. With this formalism the synthesis of an n-qubit operator via the Householder method naturally leads to synthesizing n isometries, respectively isometries from $k-1$ to k qubits for k from 1 to n. Therefore

$$\#\mathrm{U}_{\mathrm{Hous.}}(n) = \#\mathrm{Diag}_{\mathrm{rot}}(n) + \sum_{k=1}^{n} \#\mathrm{Iso}_{\mathrm{Hous.}}(k-1,k). \tag{4.31}$$

With this decomposition we voluntarily omit the side-effects that may occur between two subcircuits acting on a different number of qubits — typically the transition between the subcircuit preparing states on j qubits only and the subcircuit preparing states on $j+1$ qubits. These side-effects are asymptotically negligible and not taking them into account highly simplifies the calculations.

We are concerned with estimating $\#\mathrm{U}_{\mathrm{Hous.}}(n)$: to this end we focus on the estimation of $\#\mathrm{Iso}_{\mathrm{Hous.}}(k-1,k)$. With our current circuit, we have

$$\#\mathrm{Iso}_{\mathrm{Hous.}}(k-1,k) = 2^{k-1} \times (2 \times \#\mathrm{SP}(k) + \#\mathrm{Toff}_{[78]}(n)) \tag{4.32}$$

We have seen that the value $\#\mathrm{SP}(k)$ varies depending on the structure of subcircuits we consider:

- With rotation multiplexors [142, 174],

  $$\#\mathrm{SP}_{\mathrm{rot}}(k) = 2^{k+1} - 2k - 2, \quad \#\mathrm{Iso}_{\mathrm{Hous.,\ rot.}}(k-1,k) = 2 \times 4^k - (k+1)2^{k+1} + 2^{k-1}\#\mathrm{Toff}_{[78]}(n)$$

  hence

  $$\#\mathrm{U}_{\mathrm{Hous.,\ rot.}}(n) = \frac{8}{3}4^n - 4n2^n + (2^n - 1)\#\mathrm{Toff}_{[78]}(n) + \#\mathrm{Diag}_{\mathrm{rot}}(n) - \frac{8}{3}.$$

- With multiplexors in $\mathcal{SU}(2)$ [142],

  $$\#\mathrm{SP}_{\mathrm{su}}(k) = 2^k - k - 1, \quad \#\mathrm{Iso}_{\mathrm{Hous.,\ su}}(k-1,k) = 4^k - (k+1)2^k + 2^{k-1}\#\mathrm{Toff}_{[78]}(n)$$

  and

  $$\#\mathrm{U}_{\mathrm{Hous.,\ su}}(n) = \frac{4}{3}4^n - 2n2^n + (2^n - 1)\#\mathrm{Toff}_{[78]}(n) + \#\mathrm{Diag}_{\mathrm{rot}}(n) - \frac{4}{3}.$$

The same calculation can be done for the number of rotations in the circuit. Actually, Equations (4.31) and (4.32) highlight the decomposition of the quantum circuit into smaller subcircuits, thus remain true by replacing the number of CNOTs with the number of elementary rotations. A quantum state preparation on n qubits requires 2^{n+1} rotations, whether using rotations or $\mathcal{SU}(2)$ multiplexors [142]. Overall the number of rotations r_n required for the synthesis of a n-qubit operator with the Householder method is

$$r_n \sim \frac{8}{3}4^n.$$

Method	CNOT count	Rotation count
QR	8.7×4^n	Unavailable
Quantum Shannon	$23/48 \times 4^n$	$9/8 \times 4^n$
Householder (with rotation multiplexors)	2×4^n	2×4^n
Householder (with multiplexors in $\mathcal{SU}(2)$)	4^n	2×4^n
Lower bound	$1/4 \times 4^n$	4^n

Table 4.2: Asymptotic gate counts for decomposition methods.

Optimizing adjacent state preparations and un-preparations

For each $k-1$ to k isometry we synthesize with the Householder method, the corresponding quantum circuit is composed of:

- one state un-preparation at the beginning and one state preparation at the end of the circuit, both on k qubits,
- 2^{k-1} generalized Toffoli gates on n qubits,
- $2^{k-1}-1$ concatenations of a state preparation $SP(u_{i+1})$ and a state un-preparation $SP(u_i)^\dagger$, both on k qubits.

We now focus on the concatenations of the adjacent subcircuits $SP(u_{i+1})$ and $SP(u_i)^\dagger$ in the circuit of Figure 4.9. These sequences of operations can indeed be optimized.

To this end we need to look in more details to the state preparation circuits. A circuit P that prepares the state ψ on n qubits can be decomposed as

$$P = DY$$

where

$$D = \begin{pmatrix} e^{i\theta_1} & & \\ & \ddots & \\ & & e^{i\theta_{2^n}} \end{pmatrix}$$

such that $\theta_j = arg(\psi_j)$ and Y prepares the real state

$$\Psi = \begin{pmatrix} |\psi_1| \\ \vdots \\ |\psi_{2^n}| \end{pmatrix}.$$

Y can be synthesized with only R_y rotations by following the standard methodology for state preparation [174] without caring about the phases equal to 0. Using this decomposition the products preparation/un-preparation that we encounter in the global decomposition are of the form

$$P_j^\dagger P_i = Y_j^T D_j^* D_i Y_i$$

and the diagonal matrices can merge, thus diminishing the size by a cost of a diagonal matrix. In total $2^{k-1}-1$ diagonals on k qubits vanish. Equivalently each subcircuit (state preparation + state un-preparation) is now composed of two real state preparations and a diagonal gate. Note that the

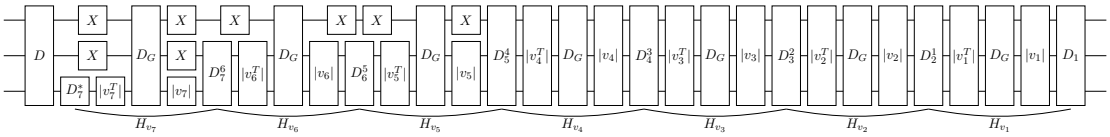


Figure 4.11: Quantum circuit designed by the Householder method for 3 qubits.

state preparation and state un-preparation at both ends of the circuit also have to be synthesized by a real state preparation followed by a diagonal gate, otherwise the synthesis of the corresponding Householder operators would not be valid. In other words we can replace Eq (4.32) by

$$\begin{aligned}\#\text{Iso}_{\text{Hous., rot}}(k-1,k) &= 2^k \times \#\text{SP}_{\text{real, rot}}(k) + (2^{k-1}+1) \times \#\text{Diag}_{\text{rot}}(k) + 2^{k-1}\#\text{Toff}_{[78]}(n) \\ &= \frac{3}{2}4^k - (k+1) \times 2^k - 2 + 2^{k-1}\#\text{Toff}_{[78]}(n)\end{aligned}$$

and

$$\#\text{U}_{\text{Hous., rot.}}(n) = 2 \times 4^n - (2n-1) \times 2^n - 2n - 4 + (2^n-1)\#\text{Toff}_{[78]}(n). \tag{4.33}$$

Asymptotically this also represents a gain of $2/3 \times 4^n$ rotations, bringing the asymptotic total to 2×4^n rotations. For completeness, we also give the total CNOT count if we merge the diagonal operators between the isometries. For each isometry, except the $\text{Iso}(n, n-1)$ one, one diagonal operator is saved, for a total of $\sum_{k=1}^{n-1} \#\text{Diag}_{\text{rot}}(k) = 2^n - 2n$ CNOT gates saved, hence

$$\#\text{U}_{\text{Hous., rot.}}(n) = 2 \times 4^n - 2n \times 2^n - 4 + (2^n-1)\#\text{Toff}_{[78]}(n). \tag{4.34}$$

If we use multiplexors in $\mathcal{SU}(2)$, only multiplexors on k qubits are merging and not diagonal anymore, saving twice less CNOTs but the same number of rotations. Eq. (4.32) is now

$$\begin{aligned}\#\text{Iso}_{\text{Hous., su}}(k-1,k) &= (2^{k-1}-1) \times (2 \times \#\text{SP}_{\text{su}}(k) - \#\text{Mult}_{\text{su}}(k)) + 2\#\text{SP}_{\text{su}}(k) + 2^{k-1}\#\text{Toff}_{[78]}(n) \\ &= \frac{3}{4}4^k - 2^k k - 1 + 2^{k-1}\#\text{Toff}_{[78]}(n)\end{aligned}$$

and

$$\#\text{U}_{\text{Hous., su}}(n) = 4^n - (2n-3)2^n + (2^n-1)\#\text{Toff}_{[78]}(n) - n - 5. \tag{4.35}$$

We also save $2/3 \times 4^n$ rotations and the number rotations becomes asymptotically equal to 2×4^n. We also notice that the operators X that appear when we switch to synthesis on a lower number of qubits disappear too by multiplying themselves. An example on 3 qubits is showed in Fig 4.11. We use the following notation: $|v_k|$ (resp. $|v_k^T|$) represents the operator that prepares (resp. un-prepares) the real state $|v_k|$ consisting of the amplitudes of the components of the state v_k. D_k is the diagonal gate containing the phases of the components of the state v_k and $D_k^j = D_j^* \times D_k$. The results for the final gate counts are given Table 4.2.

Optimization using Schmidt's Decomposition

When we have the concatenation of a circuit preparing a state $|u_i\rangle$ and a circuit un-preparing a state $|u_{i+1}\rangle$, one can realize that using the circuit template based on Schmidt's decomposition we can merge the operators or isometries. An example is given in Fig 4.12. Compared to the case where we only merged diagonal gates, we merge "almost entirely" the two state preparations so this should represent a greater gain in the number of gates. Yet we face a particular problem: how do we merge two operators whose only first half column/half rows are of interest? This happens

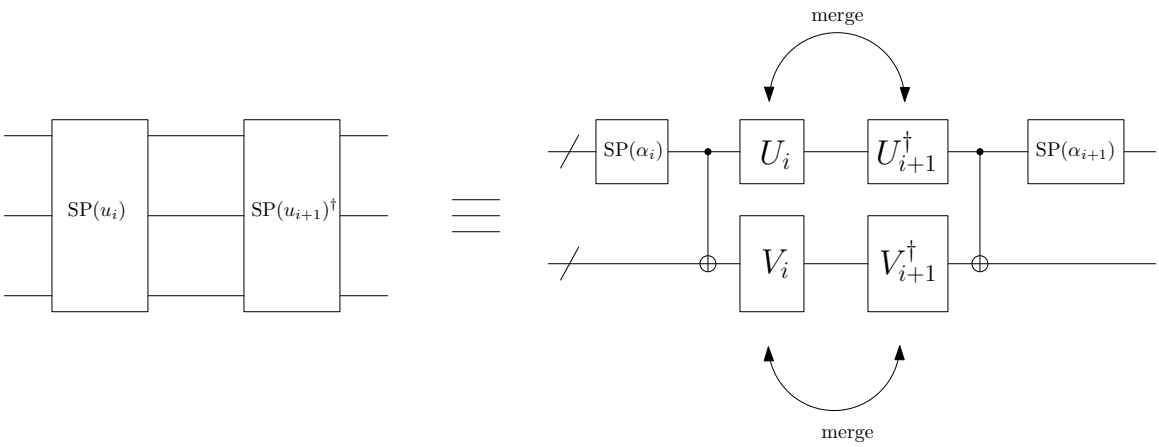


Figure 4.12: Merge of state preparation and state un-preparation with Schmidt's decomposition.

when n is odd and we have a product of an isometry and the inverse of another isometry. We have to consider a full representation of both isometries and computing the complete operator given by their product. We lose some optimality but we did not find a better way to do this merge.

Again it is easier to estimate the complexity of the k to $k+1$ isometries appearing in the circuit. We avoid the side effects when two state preparations on different numbers of qubits are concatenated and where we are supposed to merge operators not acting on the same qubits. In a $k-1$ to k isometry each merge of state preparation and state un-preparation is composed of:

- a state preparation on $\lfloor k/2 \rfloor$ qubits,
- a wall of $\lfloor k/2 \rfloor$ CNOTs,
- one unitary on $\lfloor k/2 \rfloor$ qubits and one unitary on $\lceil k/2 \rceil$,
- another wall of $\lfloor k/2 \rfloor$ CNOTs,
- a state un-preparation on $\lfloor k/2 \rfloor$ qubits.

The CNOT count for the isometry is now

$$\#\text{Iso}_{\text{Hous., Schmidt}}(k-1,k) = (2^{k-1}-1)\times(2\times\#\text{SP}(\lfloor k/2\rfloor)+2\lfloor k/2\rfloor+\#\text{U}(\lfloor k/2\rfloor)+\#\text{U}(\lceil k/2\rceil)) + 2^{k-1}\times\#\text{Toff}_{[78]}(n)+2\times\#\text{SP}_{\text{Schmidt}}(k) \quad (4.36)$$

The main interest of this method is that we reformulated the synthesis of an n-qubit operator into a succession of syntheses of $n/2$-qubit operators. The complexity results now depend on the method used for the state preparation and unitary synthesis for smaller problems. Even if we choose one specific method, it will be fastidious to derive the exact CNOT count. Instead, we derive an asymptotic complexity for the Householder method assuming that we have a general method for unitary synthesis achieving an asymptotic complexity of $\beta 4^n$ CNOTs for some β. With this simplifications we have

$$\#\text{Iso}_{\text{Hous., Schmidt}}(k-1,k) \sim 2^{k-1}\beta\times\left(4^{\lfloor k/2\rfloor)}+4^{\lceil k/2\rceil)}\right).$$

If $k=2r$ is even

$$\#\text{Iso}_{\text{Hous., Schmidt}}(k-1,k) \sim 2^k\beta\times 4^r.$$

If $k = 2r + 1$ is odd

$$\#\text{Iso}_{\text{Hous., Schmidt}}(k-1,k) \sim 2^{k-1}\beta \times (5 \times 4^r).$$

We get similar formulas that the ones obtained in [86] with Knill's decomposition.
We finally have

$$\#\text{U}_{\text{Hous., Schmidt}}(n) \sim \frac{7}{5}\beta 16^{\lfloor n/2 \rfloor} + \frac{5}{4}\beta(\lceil n/2 \rceil - \lfloor n/2 \rfloor)4^{2\lceil n/2 \rceil - 1}.$$

In other words, if n is even then

$$\#\text{U}_{\text{Hous.,Scmhidt}}(n) \sim \frac{7}{5}\beta 4^n$$

and if n is odd

$$\#\text{U}_{\text{Hous., Schmidt}}(n) \sim \frac{8}{5}\beta 4^n.$$

The overhead of $\frac{7}{5}$ or $\frac{8}{5}$ is due to the merge of the isometries. Given that the other best asymptotic result for the Householder method is 4^n CNOTs, if we manage to have a second method producing circuits with CNOT count at most $\frac{5}{8} \times 4^n$ then we have a direct improvement of the Householder method by combining the two synthesis algorithms. This works in the case of the QSD method for instance: replacing $\beta = \frac{23}{48}$ and we recover the recent results from [127]. Our result can be seen as a generalization of their method where we are free to choose any secondary method. This algorithm combining Householder and a second method will be of particular interest if the second method is too expensive to be used on large problem sizes. At the cost of a constant factor, we can first use the Householder method to decompose our large unitary into several smaller unitary synthesis and then apply a more efficient synthesis algorithm.

For completeness, we give an approximate CNOT count if we use the QSD at the second level of recursion for unitary synthesis and the standard state preparation method with $\mathcal{SU}(2)$ multiplexors:

$$\#\text{Iso}_{\text{Hous., Schmidt, QSD}}(k-1,k) = f(k) + 2^{k-1}\#\text{Toff}_{[78]}(n) + 2(2^k - k - 1)$$

with

$$f(k) = \begin{cases} \alpha_p \times 16^r - \frac{1}{2} \times 8^r \quad + \left(\delta + 2\alpha_p - \frac{2}{3}\right) \times 4^r - 3 \times 2^r \quad + \frac{2}{3} + 2\delta \text{ if } k = 2r \\ 5\alpha_p \times 16^r - \frac{5}{2} \times 8^r \quad + \left(2\delta + 3\alpha_p - \frac{4}{3}\right) \times 4^r - \frac{7}{2} \times 2^r \quad + 2 + 6\delta \text{ if } k = 2r+1 \end{cases}$$

This formula is valid only for $k \geq 2p$ with p the number of qubits where we stop the recursion in the QSD algorithm. To simplify the calculations we assume that this formula remains true for $k < 2p$. This will modify the CNOT count by a constant negligible factor given by the difference between the exact value of $f(1) + ... + f(2p-1)$ and the one given by our formula above. We also ignore other constant factors (for instance appearing in the CNOT count for diagonal operators) and we finally get that the number of CNOT gates $\#\text{U}_{\text{Hous., Schmidt, QSD}}(n)$ is

$$\frac{7}{5}\alpha_p 4^n - \frac{13}{14}\sqrt{8}^n + \frac{6\delta + 11\alpha_p - 1}{3} \times 2^n - \frac{19}{2}\sqrt{2}^n + \left(4\delta + \frac{4}{3}\right) n + (2^n - 1)\#\text{Toff}_{[78]}(n) + \text{ cste} \tag{4.37}$$

Method	flops
Quantum Shannon	19×8^n
Householder	$2/3 \times 8^n$
Classical QR factorization	$4/3 \times 8^n$

Table 4.3: Asymptotic flop counts for decomposition methods.

if n is even and

$$\frac{8}{5}\alpha_p 4^n - \frac{3\sqrt{8}}{7}\sqrt{8}^n + \frac{6\delta + 10\alpha - 1}{3} \times 2^n - \frac{13}{\sqrt{2}}\sqrt{2}^n + \left(\frac{2}{3} + 2\delta\right)(2n+1) + (2^n - 1)\#\text{Toff}_{[78]}(n) + \text{cste} \tag{4.38}$$

if n is odd.

Flop Counts

Apart from the circuit size, the other measure we are interested in is the computational cost, measured in flops.

The computational cost of the synthesis part is negligible compared to the cost of the Householder decomposition. Overall state preparations of states of size $2^n, 2^n - 1, \ldots, 3, 2$ need to be performed. For a state on k qubits, it requires $O(k2^k)$ operations, and we need to do it for 2^{k-1} states. Thus the synthesis part needs around

$$\sum_{k=1}^{n} k2^k \times 2^{k-1} = O(n4^n)$$

floating point operations. This is asymptotically negligible compared to the Householder factorization where $O(8^n)$ operations are needed. Table 4.3 summarizes the flop count for the various methods.

4.4.3 Extension to other gate sets

To synthesize a unitary matrix into a quantum circuit via the use of Householder transformations, we go through an intermediate abstract step, corresponding to Eq. (4.30). This step reformulates the synthesis of a unitary operator as a succession of state preparations, generalized Toffoli gate syntheses and the synthesis of a diagonal operator. This intermediate representation makes it easier to use any desired universal gate set. In this chapter we focused on producing CNOT based quantum circuits but if one can provide routines for state preparations, Toffoli and diagonal synthesis in another gate set $\mathcal{S}'$ then any quantum operator can be implemented with this new gate set $\mathcal{S}'$. Besides, the computational core of the factorization is identical. In some sense we can say that the Householder transformations based synthesis algorithm is partially hardware agnostic.

4.5 Experimental Results

The experiments have been carried out on one node of the QLM (Quantum Learning Machine) located at ATOS/BULL. This node is a 24-core Intel Xeon(R) E7-8890 v4 processor at 2.4 GHz. Hyper-threading has been disabled.

Most of the programs are written in C with the C-interface for LAPACK [13] (LAPACKE). We adapted the LAPACK routine ZGEQRF (in Fortran) to compute the QR factorization of unitary matrices using the blocked algorithm described in Section 4.3. LAPACK is linked with the MKL [84] multithreaded BLAS. The original ZGEQRF routine computes the best block size according to the size of the matrix and the hardware, we keep this computation in our modified routine. Our experiments use random unitary matrices generated via the LAPACK routine ZLAROR which generate matrices from a uniform distribution according to the Haar measure [183]. This way we get the most generic matrices possible: dense, without any particular structure or pattern in the matrix elements. We are thus ensured to have a worst case scenario in terms of performance for our algorithms.

We present here numerical experiments to evaluate successively the sequential performance, the strong scalability and the weak scalability using multiple cores.

4.5.1 Sequential Runs

In Figure 4.13 we compare the performance (in time) of the following routines or programs:

- The LAPACK routine ZGEQRF that computes the QR factorization of a complex matrix in double precision (note that here the matrix is square), to serve as a reference.
- Our modified ZGEQRF routine adapted for unitary matrices.
- The complete circuit synthesis process which includes the QR factorization and the synthesis of the circuit obtained from this decomposition as explained in Section 4.4.
- The Quantum Shannon Decomposition (QSD), where the implementation essentially relies on the methodology described in [174] and uses the LAPACK routine ZUNCSD to compute the Cosine-Sine Decomposition (CSD). The routine implements the algorithm in [185]. This algorithm is the state of the art and has already been used in other implementations [87,200].

We considered matrices of sizes $2^k \times 2^k, k = 1 \dots 15$ (operators acting on 1 to 15 qubits). The upper limit of 15 was chosen so that all decompositions can be achieved within an hour. This is why the curve plotting the QSD decomposition stops for 12 qubits.

As expected all the methods follow asymptotically $O(8^n)$ (curve also plotted) in accordance with the theoretical complexity. The gap between the general and modified QR factorizations in log scale corresponds approximatively to a factor of 2, in accordance with the flop count.

When comparing the QR and QSD methods, we observe that for the same amount of time we can synthesize matrices with 2, almost 3 qubits more. The ratio between the times taken by the QSD and our method is even increasing with the number of qubits, reaching a value of almost 300 for 12 qubits which is much bigger than the expected ratio of 30. This is due to the routine ZUNCSD that does not follow the theoretical complexity and does not scale well with the number of qubits.

4.5.2 Multithreaded Runs

Because we could reach 15 qubits (unitary matrices of size 32768×32768) with a sequential run in less than one hour, we chose this size for our multithreaded runs. The strong scalability is then evaluated using up to 24 threads. Since the ZUNCSD routine used for the QSD is not parallel, it has been excluded from our experiments. Figure 4.14 presents performance results (in time and Gflop/s) for the chosen number of threads. The time of the modified ZGEQRF scales like the full circuit synthesis since the QR factorization represents most of the computational cost in

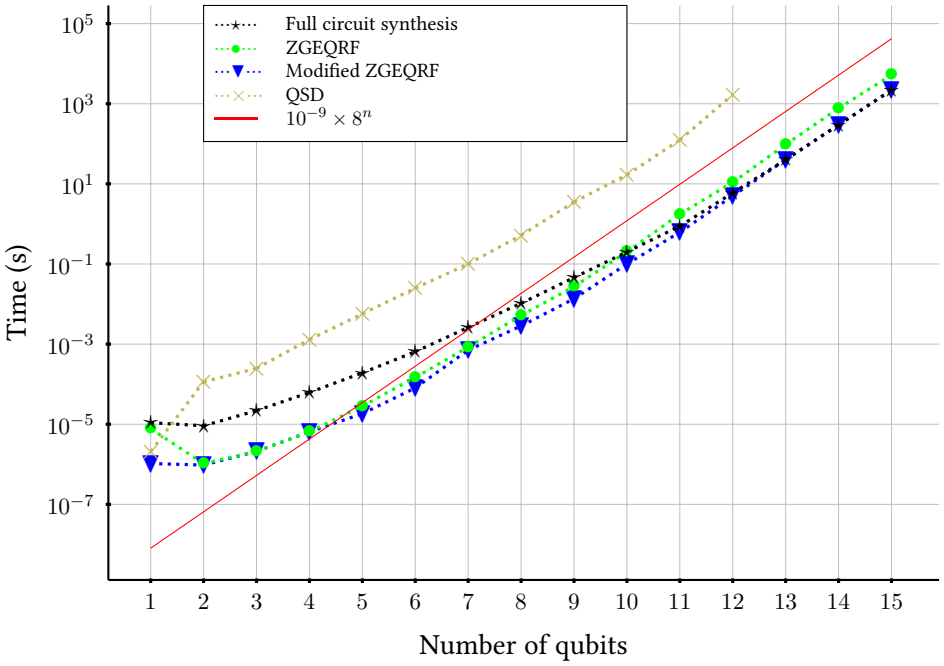


Figure 4.13: Sequential time for operator decomposition and circuit synthesis. The x-axis corresponds to the problem sizes considered given by the number of qubits on which the synthesized operator act. The computational times of the original LAPACK QR factorization ZGEQRF, our modified version of ZGEQRF, our full circuit synthesis scheme and the Quantum Shannon Decomposition are given.

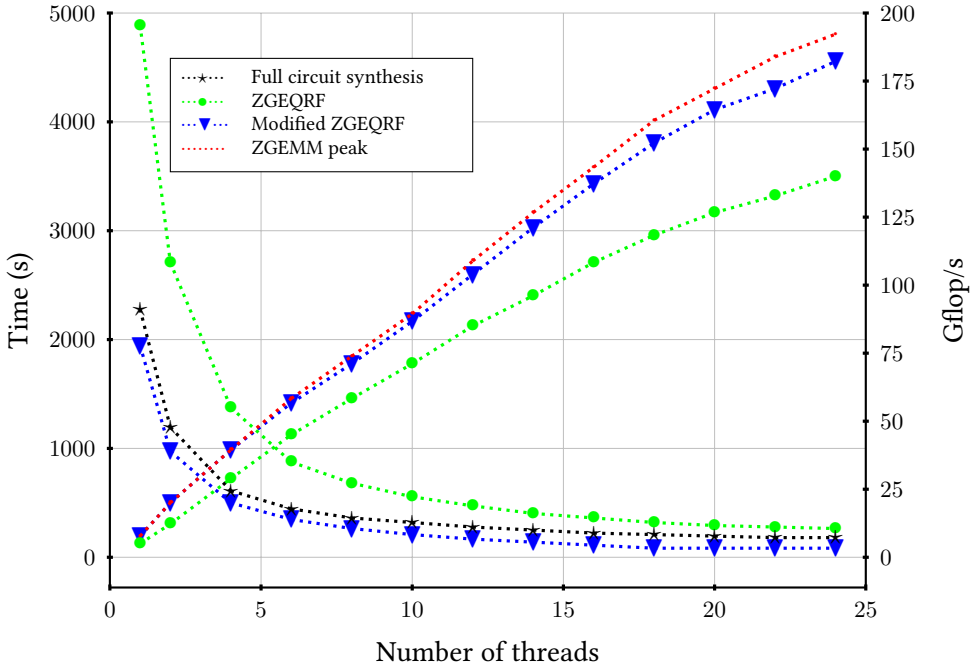


Figure 4.14: Strong scaling for quantum operator decomposition and circuit synthesis on 15 qubits. For each method (the original QR routine ZGEQRF, our modified ZGEQRF routine and our full circuit synthesis scheme) the evolution of the computational time as we increase the number of threads is plotted. For the purely factorization routines (ZGEQRF and modified ZGEQRF), the number of Gflops computed per second is also plotted with the performance of the matrix/matrix product in red.

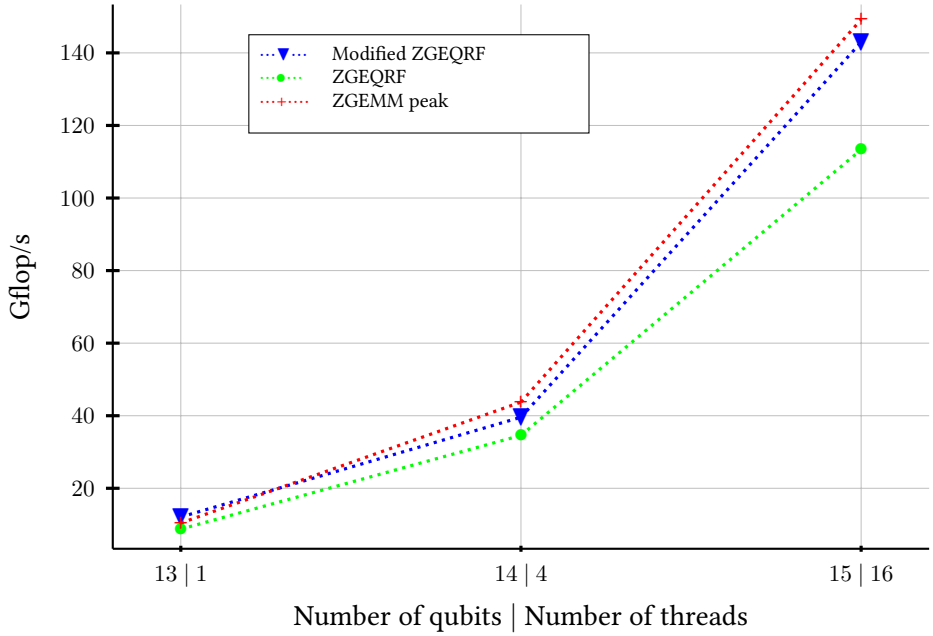


Figure 4.15: Weak scaling for quantum operator decomposition on 15 qubits. We plot the evolution of the number of Gflops computed per second as the number of qubits and threads increase, the idea being to have a fixed problem size per thread. We compare our modified ZGEQRF routine against the original QR routine ZGEQRF and the matrix/matrix product (ZGEMM).

the synthesis. Also, due to a smaller flop count, the modified QR is always much faster than the general QR. Moreover, looking at the Gflop/s performance rate, we observe that our modified QR factorization offers a good scalability due to an algorithm which is rich in BLAS 3 operations and provides a performance close to that of a matrix-matrix product (ZGEMM routine, also plotted in Figure 4.14). Note that the Gflop/s rate for the full circuit synthesis is not plotted since the bulk of the arithmetical operations correspond to those of the factorization and the time of the synthesis itself is negligible.

Our experiments on weak scaling aim at measuring how the performance evolves with the number of threads but with a fixed problem size for each thread. Our algorithm for circuit synthesis can only accept matrices of size $2^n \times 2^n$, i.e., 4^n entries. As we can only multiply the size of our problems by a factor of 4, we need to multiply also the number of threads by a factor of 4. Thus, starting from a sequential run on 13 qubits, we achieved experiments on 14 and 15 qubits using 4 and 16 threads respectively. The results given in Figure 4.15 show that the rate (in Gflop/s) of the modified ZGEQRF increases with the number of threads/qubits with a very good scalability (close to that of ZGEMM) due to the mostly BLAS 3 operations implemented in the algorithm.

4.5.3 Experiments on Graphics Processing Units (GPU)

We performed additional experiments to study the behavior of our QR algorithm for unitary matrices using two Kepler K40 with 2880 CUDA cores and a multicore host composed of two Intel Xeon E5-2620 processors (6 cores each). The time for the synthesis is not plotted here since it is negligible compared to the time of the QR factorization.

[1]The original figure can be found in [47].

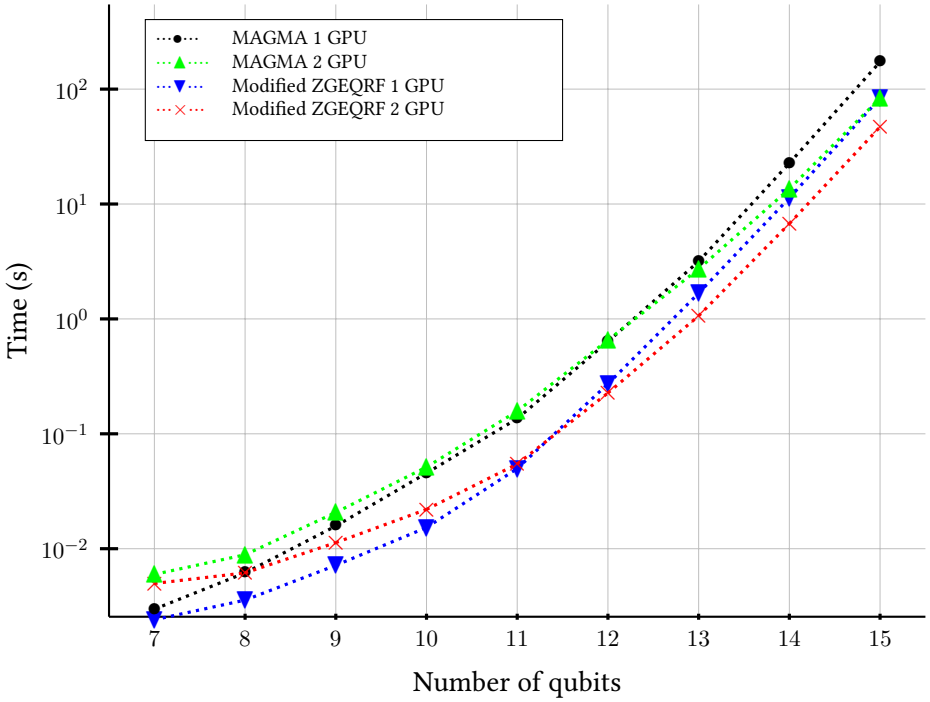


Figure 4.16: Time for factorization of unitary matrices on GPUs. We compare the computational times of the original MAGMA QR routine and our modified ZGEQRF routine adapted for GPU. Results with one and two GPUs are given.

Similarly to what was made previously with the LAPACK routine, we modified the QR routine from the MAGMA [187] linear algebra library for GPUs according to Algorithm 3.1. Note that the transfer of the panel (block column factorized at each iteration) from the CPU to the GPU performed in MAGMA is replaced by a transfer of the 2 triangular matrices mentioned in Section 4.3 which are broadcasted to the GPUs involved in the computation. In Figure 4.16, we obtain the factor of 2 (due to half the number of flops) between the standard and the modified QR factorization. We also observe that using 2 GPUs is not of interest for problems smaller than 12 qubits but we get a factor close to 2 (e.g., 1.84 for 15 qubits) when switching from 1 to 2 GPUs for problems larger than 13 qubits, showing a good scalability of the algorithm.

[1]The original figure can be found in [47].

Chapter 5

Numerical Optimization of Quantum Circuits Synthesis with Continuous Variables

Contents

5.1 Motivation and Technical Background

In the previous chapter, we pushed the limits of quantum circuits synthesis in terms of classical runtime and reachable problem sizes while keeping a reasonable quantum cost. In this chapter, we investigate the opposite problem: how to minimize as much as possible the quantum cost with reasonable classical runtime? This problem is more standard than the classical resource optimization problem. Most of the methods in the literature try to optimize a polynomially sized representation of the operator: a Boolean function in oracle synthesis [137, 139], a quantum circuit in quantum

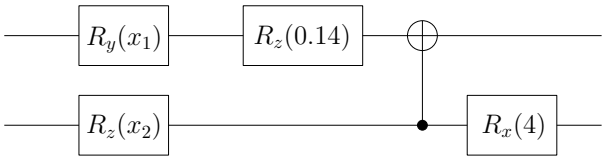


Figure 5.1: Topology on 2 qubits with 2 degrees of freedom: x_1 and x_2.

circuit optimization [145], etc. In the case where we have the complete operator $U \in \mathcal{U}(2^n)$ as input, the literature is more limited on the subject.

Solving this problem even on a small number of qubits offers both short-term and long-term benefits. In the short term, for example, it gives elementary bricks to experimenters to design programmable chips on a few qubits [75, 159]. In the NISQ (Noisy Intermediate Scale Quantum) era [157] compressing circuits as much as possible is crucial to reduce the noise and the circuit execution time as much as possible. In a more long term view, within a compilation-oriented approach, a procedure capable of synthesizing optimal circuits on 2,3,4 qubits or more can be integrated as a subtask in a peep-hole optimization framework [106]. It can also be combined with the QSD synthesis, or any recursive synthesis method, such that we can stop the recursion earlier on larger problem sizes. We discuss this application in Section 5.7 and show how we can improve the gate complexity of the QSD.

Shende *et al.* define in [176] the notion of *circuit topology*. It relies on the fact that some quantum gates are parameterized: the elementary rotations are parameterized by an angle, the MS gates are parameterized (see Section 2.3.3), etc. A *circuit topology*, or *topology* for short, is an abstraction of a quantum circuit made of constant gates and unspecified parameterized gates. The gates are placed in the circuit and the only variable data of a topology are the parameters of the unspecified gates. A priori these parameters can take any value, they are called the *degrees of freedom (DOF)* of the topology. By setting the values of the DOF we create a quantum circuit which is an instantiation of the topology.

Note that a constant gate in a topology can be a parameterized gate in another topology. Simply the value of its parameter is fixed in the first topology and unspecified in the other. An example is given in Fig 5.1. Two elementary rotations are fixed in the topology and two others are parameterized. The topology is therefore defined on 2 qubits and has 2 degrees of freedom. Each quantum circuit implements an operator $U \in \mathcal{U}(2^n)$, therefore for any topology $\mathcal{T}$ with k DOF we define

$$f_{\mathcal{T}} : \mathbb{R}^k \to \mathcal{U}(2^n). \tag{5.1}$$

$f_{\mathcal{T}}$ maps any specification of the k DOF to the corresponding operator implemented by the instantiated quantum circuit. The whole point of this chapter is the study of the functions $f_{\mathcal{T}}$ for suitable $\mathcal{T}$. Notably, a topology $\mathcal{T}$ is said to be *universal* if $\Im(f_{\mathcal{T}}) = \mathcal{U}(2^n)$ (see Section 2.3.3). In other words, for any operator U and universal topology $\mathcal{T}$ there exists a specification of the parameters in $\mathcal{T}$ such that the corresponding quantum circuit implements U.

A natural question one can ask is: what is the minimum number of gates required to have a universal topology? A lower bound of the number of CNOT gates required is given in [176]:

Theorem 5.1.1 ([176]). *A topology $\mathcal{T}$ composed of one-qubit rotations and CNOT gates cannot be universal with fewer than $\left\lceil \frac{4^n-3n-1}{4} \right\rceil$ CNOT gates.*

The proof relies on counting the DOF of the topology and circuit identities. For completeness and because we will use similar techniques, we develop it in details here.

Proof. First we use Sard's theorem [72] to claim that the image of $f_{\mathcal{T}}$ is of measure 0 if $\#DOF < dim(\mathcal{U}(2^n)) = 4^n$. We can remove one DOF corresponding to the global phase. Hence to be sure that we can potentially cover the whole unitary space we need

$$\#DOF \geq 4^n - 1 \tag{5.2}$$

Then, using circuit identities the authors propose a canonical form for CNOT+$\mathcal{SU}(2)$ circuits and they count the maximum possible number of DOF in it. Given a topology with placed CNOT gates, we can add 2 generic one-qubit gates at the left of each CNOT, i.e., on the control and target qubits, and end the topology with a wall of one-qubit gates on each qubit. This gives us a new topology $\mathcal{T}$. It is clear that this topology is "optimal" in the sense that we cannot add one-qubit gates without them merging with other gates in the circuit. Ignoring the global phase, every one-qubit gate U can be decomposed as

$$U = R_x(a) \times R_z(b) \times R_x(c)$$

or

$$U = R_z(a') \times R_x(b') \times R_z(c')$$

with suitable angles a, b, c, a', b', c'. R_x and R_z rotations commute with the CNOT gate depending on which qubit they are applied: R_z gates commute with the CNOT if they act on the control qubit of the CNOT, R_x gates commute with the CNOT if they act on the target qubit of the CNOT. The circuit identities are given in Fig. 5.2. Therefore using one of the two decompositions for all the one-qubit gates in the topology and commuting the appropriate rotations, we can show that $\mathcal{T}$ is equivalent to another topology with a fewer number of elementary rotations in it. Namely, starting from the most left CNOT gate:

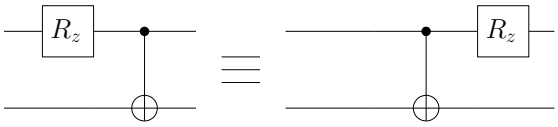
1. consider the two one-qubit gates on its left in the topology. U_c is the one on the control qubit and U_t is the one on the target qubit.
2. Decompose $U_c = R_z R_x R_z, U_t = R_x R_z R_x$, where we omit the angles for simplicity.
3. Commute the R_z gate on the control qubit with the CNOT and merge it with the next one-qubit gate in the circuit. Do the same with the R_x on the target qubit.
4. Repeat operations 2-3 with the next CNOT.
5. Decompose the last one-qubit gates on each qubit the way you want.

After this procedure, we are left with a topology $\mathcal{T}'$ functionally equivalent to the topology $\mathcal{T}$. In fact $\mathcal{T}'$ can be considered to be the canonical form of $\mathcal{T}$. An example is given in Fig. 5.3. Therefore a topology $\mathcal{T}$ with k CNOT gates can have at most $4k + 3n$ DOF. To verify Eq. (5.2) we need at least

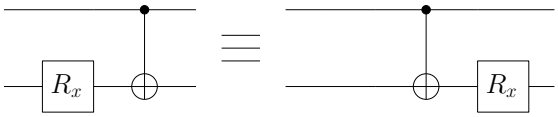
$$\#CNOT \geq \left\lceil \frac{1}{4}(4^n - 3n - 1) \right\rceil$$

for $\mathcal{T}$ to be universal. □

To our knowledge, no proof has been given showing that this lower bound is tight. Clearly this is not an obvious result. Maybe with other circuit identities one can show that some rotations are unnecessary. This will lower the maximum number of DOF a topology with k CNOTs can have and therefore it will increase the theoretical lower bound.



(a) Commutation identity with the CNOT and R_z rotation.



(b) Commutation identity with the CNOT and R_x rotation.

Figure 5.2: Circuit identities with the CNOT gate and elementary rotations.

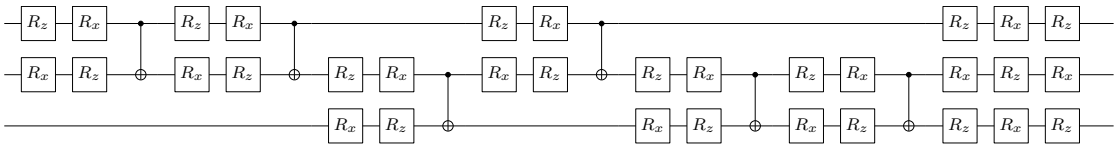


Figure 5.3: Canonical form of a topology on 3 qubits with 6 CNOT gates.

The main goal of this chapter is twofold. With the help of numerical optimizers, we seek to reduce as much as possible the quantum resources and we want to give some insights about the tightness of the lower bounds. Using heuristics or classical optimization methods to synthesize circuits is not new. The BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm [208] has already been used to synthesize trapped-ions circuits [129] and machine learning techniques have been used in the case of photonic computers [15]. Genetic algorithms have also been used in a general context [126] or for the specific IBM quantum computer [121]. However, these works are purely experimental: the overall optimality of their solution is not discussed.

5.2 Contributions

The main contributions of the chapter are as follows.

- We adapt the work in [176] to the technology of trapped-ions quantum computer which uses specific quantum gates. This technology holds great promise to cross the passage to scale: quantum circuits with trapped-ions technology have great fidelities and the qubits have a long decoherence time. In other words, the noise in the results is low and long time computations can be performed before the system interferes with the environment and loses its information. Moreover, the particular architecture of trapped-ions quantum circuits makes the problem simpler for numerical optimization because as we will see it only involves the optimization of continuous real parameters. We derive a lower bound on the number of entangling gates required to perform any computation.

- We provide a simple optimization framework and use it to synthesize generic quantum operators. We give an efficient way to compute the gradient of the cost function in our framework. This results in a better scalability of our optimizer. We can synthesize optimally — or close to optimality — arbitrary operators up to 6 qubits and arbitrary quantum states up to 11 qubits.

- We show that the lower bounds for both trapped-ions and usual superconducting circuits

seem to be tight. Especially, for the CNOT+$\mathcal{SU}(2)$ gate set, we provide different universal topologies that may reveal some structures in the group of unitary operators. We also provide depth-efficient and architecture-aware universal topologies.

- We bring new insights into the inherent tradeoff between quantum resources and classical resources.

The plan of the chapter is the following: in Section 5.3 we develop in more detail the modeling of a circuit as a parameterized topology and the question of the minimum-sized topology necessary to implement any operator. In Section 5.4 we introduce a generic optimization framework formalizing the circuit synthesis as a classical optimization problem. In Section 5.5 we apply this framework to optimally synthesize generic unitary matrices with the trapped-ions natural set of gates. We also discuss the scalability of this method and how we can naturally trade the optimality of the final quantum circuit for shorter computational time. We extend the framework to other gate sets, notably the CNOT+$\mathcal{SU}(2)$ gate set, in Section 5.6.

Sections 5.3, 5.4 and 5.5 have been presented at ICCS 2019 and they are published by Springer in LNCS [46].

5.3 Lower Bounds for the Synthesis of Trapped-Ions Quantum Circuits

In this section, we derive a lower bound in the case of trapped-ions circuits using a similar reasoning to the one done for the CNOT+$\mathcal{SU}(2)$ gate set.

Theorem 5.3.1. *A topology composed of one-qubit rotations and parameterized MS gates cannot be universal with fewer than* $\left\lceil \frac{4^n-3n-1}{2n+1} \right\rceil$ *MS gates.*

Proof. We remind that the standard gate set for trapped-ions quantum circuits is (see Section 2.3.3):

- local R_z gates,
- global R_x gates, i.e., local R_x are applied to every qubit with the same angle,
- the entangling Mølmer–Sørensen gate (MS gate) defined by

$$MS(\theta) = e^{-i\theta(\sum_{i=1}^{n} \sigma_x^i)^2/4}.$$

where σ_x^i is the operator X applied to the i-th qubit.

We give a normal form to trapped-ion circuits in order to count the number of DOF. MS gates operate on all qubits, they are diagonal in the basis $H^{\otimes n} = \bigotimes_{i=1}^{n} H$ obtained by applying an Hadamard gate to each qubit, the so-called "$|+\rangle$ / $|-\rangle$" basis or Hadamard basis. We have

$$MS(\theta) = H^{\otimes n} \times D(\theta) \times H^{\otimes n} \tag{5.3}$$

with

$$D(\theta) = \text{diag}([e^{i(n-\text{wt}(j))^2 \times \theta}]_{j=0..2^n-1}) \tag{5.4}$$

where wt is the Hamming weight in the binary alphabet.

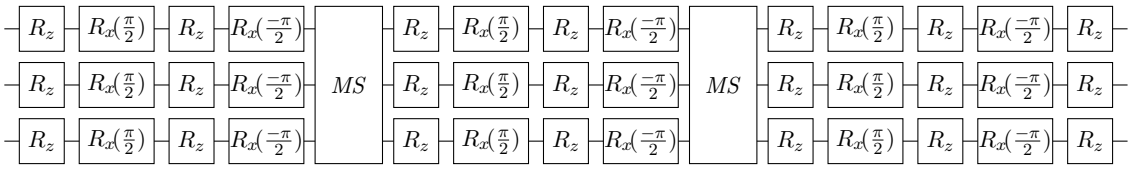


Figure 5.4: Generic quantum circuit on 3 qubits for the trapped-ions technology.

First we can merge the Hadamard gates appearing in Equation (5.3) with the local gates so that we can consider that our circuits are only composed of local gates and diagonal gates given by Equation (5.4). Then we can write each local gate U as

$$U = R_z(\alpha) \times R_x(-\pi/2) \times R_z(\beta) \times R_x(\pi/2) \times R_z(\gamma) \tag{5.5}$$

where α, β, γ parameterize the unitary matrix U. Because the MS gates are now diagonal we can commute the first R_z so that it merges with the next local unitary matrices. By doing this until we reach the end of the circuit we finally get a quantum circuit for trapped-ions hardware with the following basic subcircuit:

- a layer of local R_z gates,
- a layer of global $R_x(\pi/2)$ gates,
- a layer of local R_z gates,
- a layer of global $R_x(-\pi/2)$ gates,
- an MS gate (given in its diagonal form).

This subcircuit is repeated k times for a circuit with k MS gates. Ultimately, the circuit ends with a layer of rotations following the decomposition (5.5) on each qubit. An example of a quantum circuit on 3 qubits with 2 MS gates is given in Figure 5.4. The angles of the R_z rotations are omitted for clarity. The only parameters of such generic circuits are:

- the angles of the R_z rotations,
- the number and the angles of the MS gates.

Each elementary rotation (around the x, y, z axis) is parameterized by one angle so it can only bring one additional degree of freedom, as the MS gates if they are parameterized. Including the global phase, a circuit containing k parameterized MS gates can have at most $(2n+1) \times k + 3n + 1$ DOF. In the example given figure 5.4, the topology has 24 DOF. To reach universality we must verify equation (5.2), which leads to the lower bound

$$\#MS \geq \left\lceil \frac{4^n - 3n - 1}{2n + 1} \right\rceil. \tag{5.6}$$

□

This proof easily transposes to the state preparation problem with a few changes:

- a quantum state is completely characterized by $2^{n+1} - 2$ real parameters,

- starting from the state $|0\rangle^{\otimes n}$ we can only add one degree of freedom to each qubit on the first rotations because the first R_z result in an unnecessary global phase.

Consequently the total number of DOF a topology on n qubits with k MS gates can have is at most $(2n+1)k+2n$. We get the lower bound

$$\#MS \geq \left\lceil \frac{2^{n+1}-2n-2}{2n+1} \right\rceil. \tag{5.7}$$

To our knowledge, this is the first calculation of a lower bound in the context of trapped-ions circuits. In the next section, we propose an algorithm based on numerical optimization that achieves accurate circuit synthesis using a number of gates corresponding to the above lower bounds. This gives a good indication of the tightness of these bounds.

5.4 The Optimization Framework for Continuous Variables Circuit Synthesis

Given a function f representing a topology on n qubits, finding the best possible synthesis of a unitary matrix U with the topology f can be reformulated as solving

$$\arg\ \min_x \|f(x)-U\| = \arg\ \min_x g(x),$$

where $g(x) = \|f(x)-U\|$ and where $\|\cdot\|$ is an appropriate norm. All norms are equivalent in finite dimension, so by appropriate norm we mean a norm that will simplify the calculations. We decompose the cost function g as

$$g(x) = \frac{1}{k}\sum_{i=1}^{k} \left\| f(x)\,|e_{\phi(i)}\rangle - u_{\phi(i)} \right\|,$$

where ϕ is a permutation of $[\![1, 2^n]\!]$ and u_j denotes the j-th column of U. In other words, we generalize our problem to the synthesis of k given columns of our operator U. For simplicity now we can only study the case where $k=1$: this is the state preparation problem.

Our goal is to rely on various algorithms for non-linear optimization. We choose the norm to be the Euclidean norm squared so that with this choice of norm we have a simple expression of the cost error:

$$g(x) = 2 \times \left(1 - \Re\left(\langle 0|\, f(x)^\dagger\, |\psi\rangle\right)\right). \tag{5.8}$$

The cost to compute the error function is equivalent to the simulation cost of the circuit. Starting from the state $|\psi\rangle$ we simulate the circuit $f(x)^\dagger$, then the real part of the first component gives the error. Many methods for simulating a quantum circuit have been designed and we can rely on them to perform calculations as efficient as possible.

In our case g is C^∞. We can use more efficient optimization algorithms if we can compute the gradient and if possible the Hessian. To compute the gradient, we need to give a more explicit formula for $g(x)$. For some j we have

$$\frac{\partial}{\partial x_j} g(x) = -2\frac{\partial}{\partial x_j}\Re\left(\langle 0|\, f(x)^\dagger\, |\psi\rangle\right),$$

and by linearity we get

$$\frac{\partial}{\partial x_j} g(x) = -2\Re\left(\frac{\partial}{\partial x_j} \langle 0| f(x)^\dagger |\psi\rangle\right).$$

Let us suppose we have K gates in our circuit. Then we can write

$$f(x)^\dagger = \prod_{i=1}^{K} A_i$$

where $(A_i)_{i=1..K}$ is the set of gates on n qubits that compose the circuit f. In all circuits encountered the parameterized gates are of the form $e^{i\theta\Omega}$ where Ω can be either X, Y, Z (tensored with the identity if necessary) or more complex Hermitian operators (see the MS gate for instance), θ is the parameter of the gate. We assume that two gates are not parameterized by the same variable. Let k_1 be the index of the gate depending on x_j, then we have

$$\begin{aligned}
\frac{\partial}{\partial x_j} \langle 0| f(x)^\dagger |\psi\rangle &= \frac{\partial}{\partial x_j} \langle 0| \prod_{i=1}^{k_1-1} A_i \times e^{i\times x_j \times \Omega} \times \prod_{i=k_1+1}^{K} A_i |\psi\rangle \\
&= \langle 0| \prod_{i=1}^{k_1-1} A_i \times \frac{\partial}{\partial x_j} e^{i\times x_j \times \Omega} \times \prod_{i=k_1+1}^{K} A_i |\psi\rangle \\
&= i \times \langle 0| \prod_{i=1}^{k_1-1} A_i \times \Omega \times A_{k_1} \times \prod_{i=k_1+1}^{K} A_i |\psi\rangle
\end{aligned}$$

Therefore we have a simple expression for the partial derivatives

$$\frac{\partial}{\partial x_j} g(x) = 2\Im\left(\langle 0| \prod_{i=1}^{k_1-1} A_i \times \Omega \times A_{k_1} \times \prod_{i=k_1+1}^{K} A_i |\psi\rangle\right).$$

To compute the whole gradient vector we propose the Algorithm 5.1.

On total we need 3 circuit simulations to compute the error and the gradient: once to compute $f(x)^\dagger |\psi\rangle$ and twice to compute the gradient.

Note that, for the synthesis of k columns, we need to store $2k$ vectors when computing the gradient. If $k = 2^n$ it is possible to perform a similar algorithm by computing the trace of a $2^n \times 2^n$ matrix and iteratively applying operators on the left and right of the matrix. We will not give more details about this method but it requires less memory than the one we propose because we only need to store one matrix instead of two. However, memory is not a big concern here compared to the computational time — we focus on small problem sizes — so we keep this framework even for $k = 2^n$.

5.5 Numerical Experiments for Trapped-Ions Hardware

The experiments have been carried out on one node of the QLM (Quantum Learning Machine) located at ATOS/BULL. This node is a 24-core Intel Xeon(R) E7-8890 v4 processor at 2.4 GHz. Our algorithm is implemented in C with a Python interface and has been executed on 12 cores using OpenMP multithreading. The uniform random unitary matrices are generated according to the Haar's measure [183]. For the numerical optimization we use the BFGS algorithm provided in the SciPy [158] package.

Algorithm 5.1: Computing the gradient of the cost function.

Require: $n > 0, |\psi\rangle \in \mathbb{C}^{2^n}, f : \mathbb{R}^k \to \mathcal{U}(2^n)$,
Ensure: Computes ∂f
  $\langle\psi_1| \leftarrow \langle 0|$
  $|\psi_2\rangle \leftarrow f(x)^\dagger |\psi\rangle$
  $m \leftarrow 1$
  // N is the total number of gates in the circuit
  for $i = 1, N$ **do**
    // A_i is the i-th gate in the circuit $f(x)^\dagger$
    if A_i is parameterized **then**
      // H_i is the Hamiltonian associated to the gate A_i
      $df[m] \leftarrow 2\Im\left(\langle\psi_1| H_i |\psi_2\rangle\right)$
      $m \leftarrow m + 1$
    end if
    $\langle\psi_1| \leftarrow \langle\psi_1| A_i$
    $|\psi_2\rangle \leftarrow A_i^\dagger |\psi_2\rangle$
  end for

5.5.1 Synthesizing Generic Circuits

A clear asset of trapped-ion technology is that there is no notion of topology. Contrary to superconducting circuits where we have to deal with the placement of the CNOT gates, here we just have to minimize the number of MS gates to optimize the entangling resources. Local rotations are less expensive to realize experimentally [129]. So, as a first approach, we can always apply one-qubit gates on every qubits between two MS gates such that a quantum circuit for trapped ions always follow a precise layer decomposition.

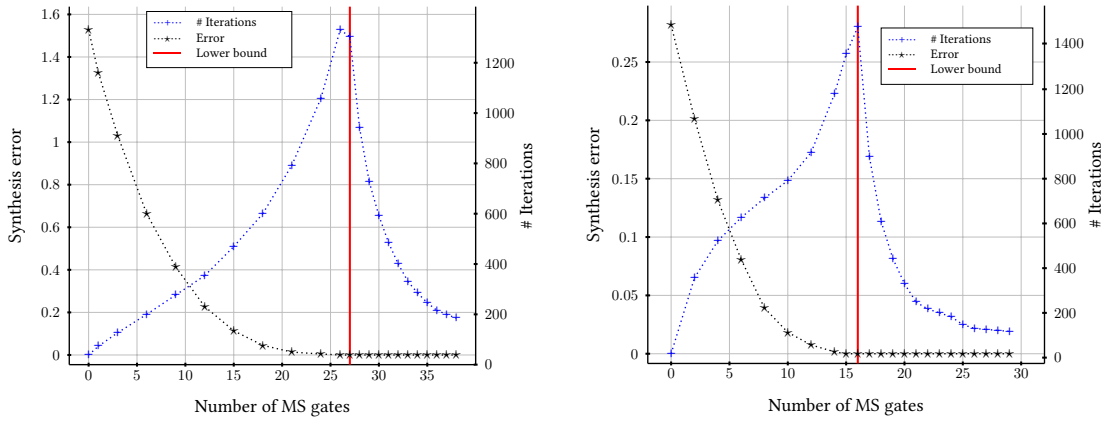
This asset can be qualified because there is not only one trapped-ions technology, and there are models where entangling gates on only subsets of qubits are available (or are at least less costly than a global MS gate). It depends notably on the isolation of these qubits in different traps.

For 50 random unitary matrices on $k \in \{2, 3, 4\}$ qubits, and quantum states on k ranging in $\{2, 3, 4, 5, 6, 7\}$ qubits, we execute Algorithm 5.1 with circuits containing various numbers of parameterized MS gates. The stopping criterion for the optimization is the one chosen in SciPy, i.e., the norm of the gradient must be below 10^{-5}. We repeat the optimization process several times per matrix with different starting points to maximize our chance to reach a global minimum. Then we plot, for various numbers of MS gates,

- the error expressed in Formula (5.8), maximized over the sample of matrices,
- the number of iterations needed to converge to a local minimum, averaged over the sample of matrices.

We summarize our results in Figures 5.5a and 5.5b, corresponding respectively to the circuit synthesis problem on 4 qubits and the state preparation problem on 7 qubits. The results obtained for lower numbers of qubits are similar. The amount of time required to perform such experiments for larger problems is too important (more than a day).

For both graphs, we observe an exponential decrease of the error with the number of MS gates. This strong decrease shows that there is a range of MS gates count for which we have an acceptable accuracy without being minimal. Although we can save only a small number of MS gates by this



(a) 4-qubits Quantum circuit synthesis problem.

(b) 7-qubits State preparation problem.

Figure 5.5: Evolution of the synthesis error and the number of iterations with the number of MS gates. The x-axis gives the number of MS gates in the trapped-ion quantum circuit. The left y-axis gives the maximum error we encountered during the synthesis of a sample of random operators or random states. The right y-axis gives the average number of iterations needed for our numerical optimizer to return a result. We considered both the case of unitary synthesis on 4 qubits and the case of state preparation on 7 qubits. For each problem, there exists a theoretical lower bound on the number of MS gates necessary to be able to synthesize any operator with an arbitrary error. Such lower bound is represented in red on each graph.

trade, we conjecture that for a given precision the number of saved MS gates will increase with the number of qubits. In other words, if we want to synthesize an operator up to a given error, the range of "acceptable" number of MS gates will increase with the number of qubits. We also want to insist on the fact that the error here is, in fact, a squared error. The "real" error would be rather given by the square root of the error we compute. In other words, for a computed error of 10^{-8} we have in fact an error of approximately 10^{-4} between the target and the synthesized operators.

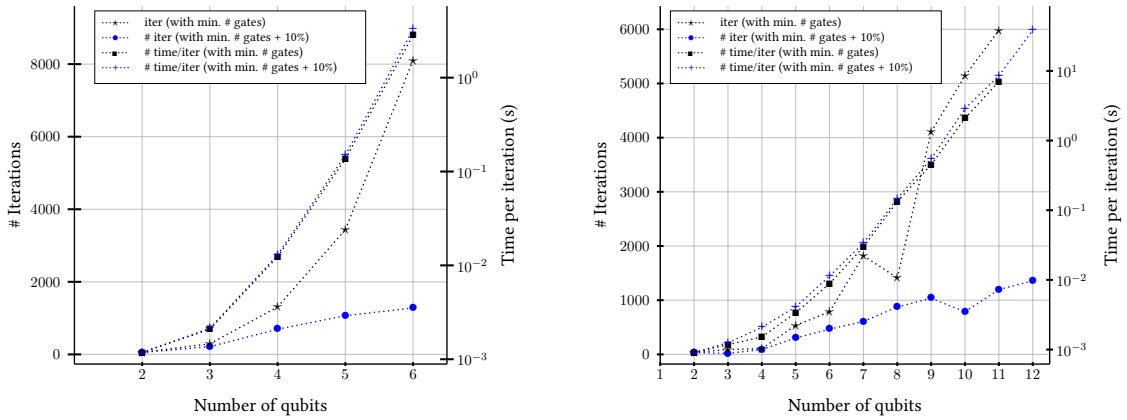
What is the experimental lower bound? Interestingly, the number of iterations is a better visual indicator of this lower bound: once the exact number of DOF is reached, we add redundant degrees of freedom into the optimization problem. We make the hypothesis that this leads to a bigger search space but with many more global minima. Hence the search is facilitated and the algorithm can converge in fewer iterations. So the peak of the number of iterations corresponds to the experimental lower bound, which is confirmed by the computed errors.

The experimental lower bounds follow the formulas Eq. (5.6) and Eq. (5.7) given in Section 5.3 except for the 2-qubit quantum circuit synthesis case where we need 1 more MS gate to reach universality. This gives strong indications about the correctness of the theoretical lower bounds and the relevance of the approach of counting the degrees of freedom to estimate the resources necessary for implementing a quantum operator.

5.5.2 Tradeoff Quantum/Classical Cost

In the previous experiments, we could synthesize unitary operators on 6 qubits in 7 hours and quantum states on 11 qubits in 13 hours, both with circuits of optimal size. In the graphs plotted in Figures 5.5a and 5.5b, we also observe an exponential decrease in the number of iterations after

[0]Please refer to [46] for the official figures.



(a) Quantum circuit synthesis problem.

(b) State preparation problem.

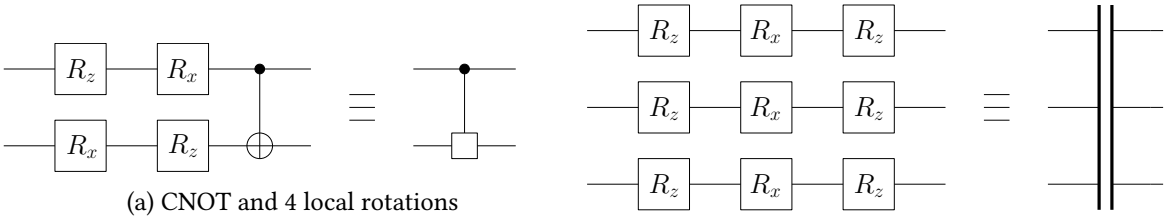
Figure 5.6: Number of iterations and time per iteration for different problem sizes. We compare two cases: the first case corresponds to numerical optimizations done with circuits of optimal size, i.e., given by the theoretical lower bound. For the second case we consider circuits a little less optimal, 10% larger. We consider both the unitary synthesis problem and the state preparation problem, with, overall, similar results.

we have reached the optimal number of MS gates. We can exploit this behavior if we want to accelerate the time of the synthesis at the price of a bigger circuit. By adding only a few more MS gates the number of iterations can be significantly reduced, allowing us to address potentially bigger problems in the synthesis. In Figure 5.6 we show, for different problem sizes, how the iteration count and the time per iteration evolve when we add 10% more MS gates to the optimal number of gates. We observe that the number of iterations is reduced at most by a factor 5 in the case of 6 qubits for generic operators and 11 qubits for quantum states. More importantly, with such an augmentation in the number of MS gates, the number of iterations only slightly increases, almost linearly, in contrary to circuits of optimal size where the number of iterations increases exponentially. This means that the time per iteration, also increasing exponentially with the number of qubits, is the main limiting factor in the good scalability of the method. Thus any improvement in the classical simulation of a quantum circuit (i.e., reducing the time per iteration) will lead to a significant acceleration in the synthesis time.

Finally, in our experiments, we achieved circuit synthesis on 6 qubits in about an hour and state preparation on 11 qubits in about 3 hours. Despite this sacrifice in the quantum cost (since we use more gates), this is still to our knowledge the best method to synthesize generic quantum circuits and quantum states for the trapped-ions technology.

5.6 Universal Topologies for Other Gate Sets

We extend the use of our framework to other gate sets. This task is not simple because we consider gate sets with two-qubit entangling gates (CNOT, iSWAP, $\sqrt{\text{SWAP}}$) and we need to think of how to arrange those gates to have a universal topology. Once we have found good structures with repeating patterns, the behavior of the numerical optimizer is about the same as for the MS+rotations gate set: we observe a strict drop of the error when we reach a certain number of gates in the topology. We also observe the same decrease in the number of iterations if we add more gates, making it possible to trade the quantum cost for the classical cost. In this section, we



(a) CNOT and 4 local rotations

(b) Wall of one-qubit gates

Figure 5.7: Graphical notation for 2 blocks of gates.

will not detail the numerical results each time but rather we show the topologies that are likely to be universal.

We did not repeat the experiments we did with the MS gates. The way we estimate if a topology is universal or not is based on a simplified procedure:

- we set the tolerance of the gradient to 10^{-8},
- we tried several topologies with a regular pattern and we increased the number of gates progressively,
- the error must drop to 10^{-8} at least for a sample of random unitaries. We recall that the error is a squared error so the real error for 10^{-8} is 10^{-4}.

We did not rely on the number of iterations to help evaluate if universality is reached or not. Generally, it is not easy to tell with the experiments if a topology is universal or not. We want to warn that there are some inaccuracies, simplifications, that will be done to get the results we will present. Especially, the notion of universality becomes less and less clear as the number of qubits increases. For example, for unitary synthesis: with $n = 3, 4$ we notice a drop in the error when we reach the theoretical lower bound, for instance from 10^{-5} to 10^{-10} or even less. However, for $n = 5$ the error slowly decreases and we can hardly notice a change at the theoretical lower bound. The error is already at 10^{-7} before reaching the lower bound and the convergence is harder and harder and requires more and more iterations to reach a negligible error. For these reasons, we are aware that our results must be questioned. Yet we believe our results still give good insights about the tightness of the lower bounds.

5.6.1 The CNOT + $\mathcal{SU}(2)$ Gate Set

Following the proof in [174] detailed in Section 5.1, we consider blocks of one CNOT and 4 local rotations and ultimately a wall of one-qubit gates. We give in Fig 5.7 the graphical notations for those two blocks of gates. The lower bound of

$$\left\lceil \frac{1}{4}(4^n - 3n - 1) \right\rceil$$

CNOT gates given in [174] seems to be tight. We were able to reach universality with several topologies. They are summarized in Table 5.1. For each topology, the minimum number of CNOT required to have universality is given by the theoretical lower bound. The first two topologies (stairs and depth friendly) are LNN compliant. The depth friendly topology is also optimal in depth. In our opinion, this should be the preferred topology if one has to synthesize small unitaries on a real architecture. The two others ("one control", "one target") may be useful if one qubit is centralized in the hardware and is connected to all the other isolated qubits.

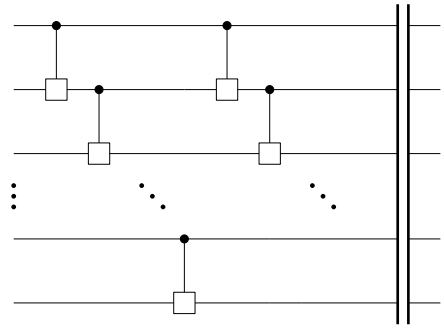
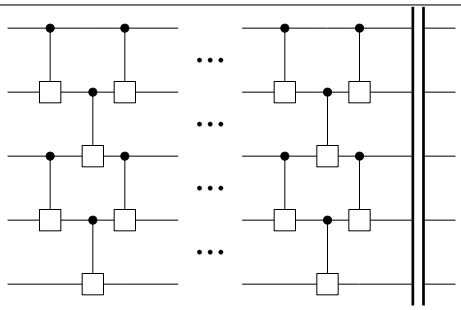
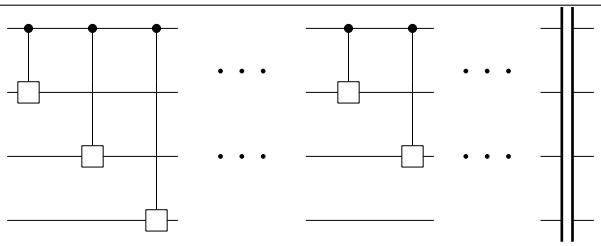
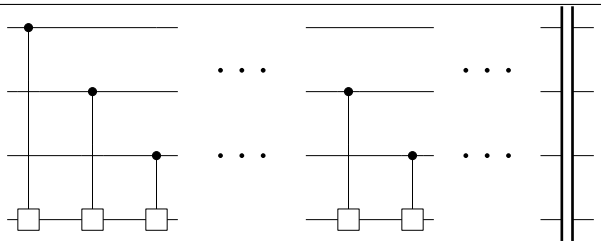
Name	Graphical representation	Features
Stairs		LNN compliant.
Depth friendly		LNN compliant. Optimal in depth.
One control		One centralized qubit.
One target		One centralized qubit.

Table 5.1: Universal topologies for the CNOT+$\mathcal{SU}(2)$ gate set.

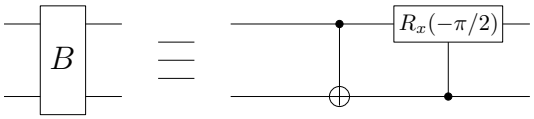


Figure 5.8: Circuit equivalence for the B gate.

5.6.2 The iSWAP + $\mathcal{SU}(2)$ Gate Set

The iSWAP gate, acting on 2 qubits with matrix representation $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, has similar commutation rules with the elementary rotations than the CNOT. Namely, it commutes with the R_z rotations on both qubits. We need to adjust the block of an iSWAP gate but, otherwise, we can transpose all the results for the CNOT gate. The topologies given in Table 5.1 are also universal topologies for the iSWAP + $\mathcal{SU}(2)$ gate set.

5.6.3 With B Gates, Controlled-Phases, Toffoli, etc.

Occasionally a topology with k entangling gates can almost be universal but we still need to add one more entangling gate to reach universality. By doing so we add redundant DOF. The best example is the two-qubit case where a circuit with 2 CNOTs can have at most 15 degrees of freedom while 16 are expected to reach universality. 3 CNOTs are therefore necessary to implement most of the operators but we expect that it is possible to synthesize any two-qubit operator with only two uses of a more suitable entangling two-qubit gate. Such a gate was found in [213]. This gate B is equivalent to a CNOT followed by a controlled-$R_x(-\pi/2)$ where the target of the CNOT is the control of the controlled-R_x, see Fig 5.8. Given the shape of B, one can feel that the commutation rules are not as clear as for the CNOT and that probably more DOF can be carried "on the left" of the gate. If it happens to be true it will explain why fewer B gates are necessary. We did some experiments on this gate and tried to evaluate the number of B gates necessary to implement a 3-qubit operator, a 4-qubit operator, etc. As we do not know how elementary rotations commute with the B gate, we tested several topologies (stairs, depth friendly) with an elementary block made of 6 elementary rotations and one gate B, plus a final wall of three elementary rotations on each qubit. The results are summarized in Table 5.2 with other gate sets. For each problem size from 2 to 5, the number of B gates required to reach an apparent universality follows the formula $\left\lceil \frac{1}{6}(4^n - 3n - 1) \right\rceil$. In other words, each rotation in each block contributes as one DOF. As a 2-qubit gate, the gate B is optimal in the sense that a topology with one two-qubit gate can have at most 6 DOF (7 with the global phase). This also means that the gate B does not commute with any elementary rotation otherwise the number of DOF per B gate would be lower.

This count can also be achieved with another gate: the Toffoli gate. The Toffoli gate follows the same commutation rules as the CNOT but, in this case, we have two controls instead of one. In some architectures, for example with neutral atoms, the Toffoli gate, or the CCZ gate, can be realized with great fidelity [119, 168]. The use of the Toffoli gate instead of the CNOT gate can, therefore, be an asset in this case.

Lastly, we explore the use of another kind of entangling gates: the controlled-rotations. We focus on the controlled-phases. They commute with R_z gates on both qubits so a block can contain up to 4 elementary rotations. However, the controlled-phase is in itself a parameterized gate so a block of elementary rotations and one controlled-phase can contain at most 5 DOF. The lower bound

Gate set	Estimated count for universality				Formula
	$n=2$	$n=3$	$n=4$	$n=5$	
CNOT + $\mathcal{SU}(2)$ iSWAP + $\mathcal{SU}(2)$	3	14	61	252	$\left\lceil \frac{1}{4}(4^n - 3n - 1) \right\rceil$
$\sqrt{\text{SWAP}} + \mathcal{SU}(2)$	4	18	81	336	$\left\lceil \frac{1}{3}(4^n - 3n - 1) \right\rceil + 1$
Controlled-phase + $\mathcal{SU}(2)$	2	11	49	202	$\left\lceil \frac{1}{5}(4^n - 3n - 1) \right\rceil$
B + $\mathcal{SU}(2)$ Toffoli + $\mathcal{SU}(2)$	2 \	9	41	168	$\left\lceil \frac{1}{6}(4^n - 3n - 1) \right\rceil$
MS + $\mathcal{SU}(2)$	3	8	27	92	$\left\lceil \frac{1}{2n+1}(4^n - 3n - 1) \right\rceil + 1$

Table 5.2: Estimated entangling gate counts to reach universality for different gate sets. The figures in red correspond to the cases where it was observed that an additional gate was necessary compared to the theoretical lower bound.

of the number of controlled-phases is, therefore, equal to $\left\lceil \frac{1}{5}(4^n - 3n - 1) \right\rceil$. Again we repeat the same experiments and the results are given Table 5.2. The counts again follow the theoretical lower bound.

5.6.4 The $\sqrt{\text{SWAP}} + \mathcal{SU}(2)$ Gate Set

Finally we explore the use of the gate

$$\sqrt{\text{SWAP}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2}(1+i) & \frac{1}{2}(1-i) & 0 \\ 0 & \frac{1}{2}(1-i) & \frac{1}{2}(1+i) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

This gate is not maximally entangling as we need two uses of the $\sqrt{\text{SWAP}}$ to produce a Bell state from product states. Said differently, two $\sqrt{\text{SWAP}}$ are necessary to implement a CNOT gate [56]. The results we get for this gate are not as clear as for the other gate sets and this may be explained by its lower entangling capabilities. For $n = 3, 4, 5$ a lower bound of $\frac{1}{3}(4^n - 3n - 1)$ seems to be verified. On the other hand for $n = 2$ we need $4\sqrt{\text{SWAP}}$ to reach universality while the theoretical lower bound we deduce from the cases $n = 3, 4, 5$ suggests that only $3\sqrt{\text{SWAP}}$ are necessary. We are unable to explain this gap.

Above all, the theoretical lower bound suggests that each $\sqrt{\text{SWAP}}$ carries at most 3 DOF. This is a strange result because the symmetry of the gate lets think that an even number of elementary rotations can be placed on the left of the $\sqrt{\text{SWAP}}$ gate so we might expect an even number of DOF per $\sqrt{\text{SWAP}}$ gate. Again we do not have any satisfactory explanation and this should be the subject of a future work.

5.6.5 Universal Topologies for State Preparation, Isometries

We have seen in Section 5.5 that the theoretical lower bound for state preparation is probably tight with the MS gate set. In this section, we give similar results for CNOT based topologies and we also generalize to any operator specified by k columns, for any $1 \leq k \leq 2^n$. In a nutshell, we report that the candidate universal topologies (stairs, depth friendly, etc.) are likely to be universal and the theoretical lower bounds again match.

Each column of an n-qubit unitary operator is described by 2^{n+1} real parameters and a set of k columns is described by $k2^{n+1}$ real parameters. The orthonormality conditions between the columns of the operator give us $\binom{k}{2}$ complex equations given by the orthogonality between any pair of columns and k real equations given by the unitarity of each column. Therefore, a set of k columns of a unitary operator is entirely specified by

$$k2^{n+1} - k^2$$

real parameters.

Next, if we are solely interested in the synthesis of k columns, this means that we have $\lceil \log_2(k) \rceil$ qubits unspecified and $n - \lfloor \log_2(k) \rfloor$ that are in a fixed arbitrary state, $|0\rangle$ for instance. We already explained that we can remove one rotation for each qubit in a specific state. With the fact that each CNOT gate can carry at most 4 DOF, the maximum number of DOF in a circuit topology with p CNOT gates and $\lceil \log_2(k) \rceil$ unspecified qubits is

$$4p + 1 + 3\lceil \log_2(k) \rceil + 2(n - \lceil \log_2(k) \rceil)$$

where we added one DOF given by the global phase. The theoretical lower bound is the solution of

$$4p + 1 + 3\lceil \log_2(k) \rceil + 2(n - \lceil \log_2(k) \rceil) > k2^{n+1} - k^2,$$

i.e.,

$$\#CNOT \geq \left\lceil \frac{1}{4} \left(k2^{n+1} - k^2 - 1 - \lceil \log_2(k) \rceil - 2n \right) \right\rceil.$$

Replacing k by 2^n and one recovers the lower bound for unitary synthesis. Replacing k by 0 and one recovers the lower bound for state preparation. Finally replacing k by 2^m and one recovers the lower bound for m to n isometry synthesis.

We tested the four topologies ("stairs", "depth friendly", "one control", "one target") in the following cases:

- state preparation,
- synthesis of a 1 to n isometry,
- synthesis of a $n-1$ to n isometry,
- unitary synthesis,
- unitary synthesis up to diagonal gate.

The results are summarized in Table 5.3. A few remarks about the results:

- the bounds for the state preparation and 1 to n isometry cases are not clear. The differences we observed with the theoretical lower bounds are represented by the figures in red or in blue: the figures in red are a unit larger than the theoretical lower bound, the figures in

Operators	Estimated count for universality												Formula
	$n=2$		$n=3$		$n=4$		$n=5$		$n=6$		$n=7$		
Quantum State	1	1	3	3	7	7	14	14	29	30	62	62	$\left\lceil \frac{1}{4}(2^{n+1}-2n-2) \right\rceil + 1 + 1$
	1	1	3	3	6	6	14	14	29	29	61	61	
1 to n isometry	2	2	6	6	13	14	28	29	60	62	125	125	$\left\lceil \frac{1}{4}(2^{n+2}-2n-6) \right\rceil + 1 + 1$
	2	2	6	5	13	13	28	28	60	60	124	124	
$n-1$ to n isometry	2	2	10	10	45	45	189	189	x		x		$\left\lceil \frac{3}{16} \times 4^n - \frac{3}{4}n \right\rceil$
	2	2	10	10	45	45	189	189					
Unitary	3	3	14	14	61	61	252	252	x		x		$\left\lceil \frac{1}{4}(4^n-3n-1) \right\rceil$
	3	3	14	14	61	61	252	252					
Unitary (up to diag.)	2	2	13	13	58	58	x	x	x		x		$< \left\lceil \frac{1}{4}(4^n-3n-1) \right\rceil$
	2	2	13	13	58	58	x	x					

Table 5.3: Estimated CNOT counts to reach universality for different kinds of operators and four topologies. The northwest quadrant gives results for the stairs topology, the northeast quadrant is for the depth friendly topology, the southwest quadrant is for the one control topology and the southeast quadrant is for the one target topology. The figures in red, resp. blue, correspond to the cases where it was observed that one, resp. two, additional gate was necessary compared to the theoretical lower bound.

blue are two units larger. For $n = 3$, we need an extra CNOT for the four topologies to reach universality. For $n = 4$ the stairs and depth friendly topologies again need an extra CNOT gate compared to the theoretical lower bound to reach universality but not for the "one control" and "one target" topologies.

- For $n = 5$, in the unitary synthesis problems, the BFGS algorithm has trouble finding a clear global minimum when the number of CNOTs is around the theoretical lower bound — we get errors around 10^{-7} — making it hard to decide when universality is reached or not. Especially, when we try to estimate the number of CNOT gates required to synthesize an operator up to a diagonal gate, no known theoretical lower bounds exist and the case $n = 5$ is not clear at all.
- We note that for the "one control" and "one target" topology the convergence of the BFGS algorithm is easier.

5.7 Combining Universal Topologies with Other Synthesis Methods

We discuss the possibility of combining synthesis using universal topologies with algebraic methods. We want to highlight the complementarity of algebraic and numerical methods. The former allows tackling larger problem sizes by breaking down the synthesis into a multitude of

p	Exact Unitary synthesis			Unitary synthesis Up to Diag.		
	#U$_{\text{-diag}}(p)$	δ_p	α_p	c_p	δ_p	α_p
1	0	0	$2/3 \approx 0.666$	0	0	$2/3 \approx 0.666$
2	3	0	$13/24 \approx 0.542$	2	1	$23/48 \approx 0.479$
3	14	0	$77/192 \approx 0.401$	13	1	$37/96 \approx 0.385$
4	61	0	$127/384 \approx 0.331$	58	3	$245/768 \approx 0.319$
5	252	0	$899/3072 \approx 0.293$	x	x	x
6	1020	0	$3347/12288 \approx 0.272$	x	x	x
7	4091	0	$803/3072 \approx 0.261$	x	x	x

Table 5.4: Possible values for the leading coefficient of the QSD method.

syntheses on a small number of qubits. Once a sufficiently small number of qubits is reached, we can call a numerical optimizer as a subroutine and improve the optimality of the initial algebraic decomposition. We illustrate this with two algebraic methods: the QSD and the Householder based synthesis method presented in Chapter 4.

5.7.1 With the QSD

We recall that the CNOT complexity of the QSD method is given by

$$\#\text{U}_{\text{QSD}} = \alpha_p 4^n - 3 \times 2^{n-1} + \frac{1}{3} + \delta_p$$

with

$$\alpha_p = \frac{\#\text{U}_{\text{-diag}}(p) + 3 \times 2^{p-1} - 1/3}{4^p}$$

and

$$\delta_p = \#\text{U}(p) - \#\text{U}_{\text{-diag}}(p)$$

and p is the number of qubits where we decide to stop the recursion. The improvements in the synthesis of small unitaries with the BFGS optimizer have a direct impact on the value of α_p. In Table 5.4 we give the values of α_p for different p and whether we synthesize the small operators completely or up to a diagonal gate. Without even considering a synthesis up to a diagonal operator, assuming that we can always synthesize a p-qubit operator with $\lceil 1/4(4^n - 3n - 1) \rceil$ CNOT gates using a numerical optimizer, the value of α_p decreases exponentially in p up to the asymptotic limit of $1/4$. It is not unimaginable that with good optimizations an exact synthesis on 6 qubits is possible. For $n = 6$, $\alpha_6 \approx 0.272$ and we are very close to the theoretical lower bound of $\alpha = 0.25$.

For $n = 3, 4$ we can save a few CNOTs assuming that the synthesis can be done up to a diagonal gate. The new CNOT counts are given in the last row of Table 5.3. To compute those results, we simply added a diagonal topology to our candidate topology and we reduced the number of CNOT gates in the candidate topology until we were unable to reach apparent universality. Implementing small operators up to a diagonal gate has a non negligible impact on the value of α_p, see the results on the second column of Table 5.4.

However, this combination of the QSD and numerical synthesis raises the problem of the global approximation error. If U, V are two unitaries such that $\|U - V\| < \epsilon_1$ and W, Ω are two other unitaries such that $\|W - \Omega\| < \epsilon_2$, then we have $\|UW - V\Omega\| < \epsilon_1 + \epsilon_2$. In other words, the errors add up. Synthesizing an n-qubit operator via the QSD and stopping the recursion at

p qubits require the synthesis of 4^{n-p} operators on p qubits. Even a good approximation error for each p-qubit operator might not be enough if all errors are added. Therefore we need to have strong confidence that our numerical scheme will be able to reach sufficiently low errors to keep the global error low as well. From the experiments we did, we are confident for the case $n = 4$ but for $n = 5$ we are not as confident. Moreover, the classical runtime increases as the target error is low and on a laptop, without particular optimization, it becomes very costly to run the BFGS algorithm on 5 qubits in order to have an error of 10^{-8} or even lower. From the experiments done in Chapter 4 we know that the QSD method cannot be applied on more than 12-qubit operators, this gives an upper bound of 4^{12-p} on the number of p-qubit operators we have to synthesize. With $p = 4$ we have 65536 operators on 4 qubits to synthesize and it is not too expensive to synthesize each of them with an error below 10^{-9}. Therefore we can manage to have an error of at most 6×10^{-5} for the global operator. We must then see on a case by case basis if this error is acceptable or not.

5.7.2 With the Householder Framework

We simply recall the results from Chapter 4. We have shown that we can use a recursive formulation of the Householder method and if we have a synthesis method achieving a CNOT count of $\beta 4^n$ for $n \leq p$ then we get:

$$c_n = \left(\frac{7}{5}\right)^{q_1} \left(\frac{8}{5}\right)^{q_2} \beta 4^n$$

where q_1, q_2 are given as follows: $n = x_0 x_1 ... x_m$ is the binary decomposition of n, q is the biggest integer such that $x_0 x_1 ... x_q \leq p$ and

$$q_1 = |\{x_i = 0 \mid i \in [\![q+1, m]\!]\}|,$$

$$q_2 = |\{x_i = 1 \mid i \in [\![q+1, m]\!]\}|.$$

We illustrate it with $n = 18, p = 3$. We expect $c_{18} = \frac{7}{5} c_9 = \frac{7}{5}\frac{8}{5} c_4 = \frac{7}{5}\frac{8}{5}\frac{7}{5} c_2$. Given that $18 = 10010$, we find $q = 2$ and $q_1 = 2, q_2 = 1$ hence the result.

In theory, this result is not very useful. For large n and small p one can expect the leading coefficient to be multiplied by at least $\left(\frac{7}{5}\right)^{\log_2(n/p)} \approx n^{\log_2(7/5)} \approx \sqrt{n}$. Even with optimal synthesis for less than p qubits, the improvement in the leading coefficient can only be by a factor of 2 compared to the best method in the literature. So the overhead introduced by the use of the recursion in the Householder framework is not compensated by the optimality of the synthesis on small problems. The CNOT count quickly becomes several orders of magnitude larger than the theoretical lower bound or even the complexities of other methods (the QSD, the Householder with standard state preparation, etc.).

In practice, however, the value of n will never exceed 20. This means that in one step of the recursion we already have to synthesize operators on 10 qubits at most. For this range of sizes, the QSD is fast enough to be called and the gain in complexity compensates the factor of $\frac{7}{5}$ or $\frac{8}{5}$ such that we still have a method that produces smaller circuits than the standard Householder method.

The other possibility was to apply twice the recursive Householder framework and then use our numerical optimizer. But this would give an asymptotic complexity of at least $\left(\frac{7}{5}\right)^2 \times \frac{1}{4} \times 4^n = \frac{49}{100} \times 4^n$ which is not better than using the QSD after one level of recursion.

5.7.3 The Synthesis Methods are Complementary

Overall it seems that three synthesis methods are complementary and produce the shortest circuits in a specific range:

- for small problems, typically $n \leq 4$, a numerical optimizer with a universal topology can be used. It produces optimal circuits for most of the unitaries.
- the QSD is the best algebraic method but its scalability is limited. For $n \leq 12$ it can be used in accordance with the numerical method.
- When $n > 12$ we need to use the fast Householder method. With the use of Schmidt decomposition we can transform the synthesis on $n < 20$ qubits into several synthesis on $n < 10$ qubits where we call the QSD.

We summarize this global synthesis framework in Table 5.5. For each problem size the best result is in bold.

Algorithm	CNOT counts $n=$ 1	2	3	4	5	6	7	8	Formulas
Specific operators									
$R_{x/y/z}$ multiplexor	\	2	4	8	16	32	64	128	2^{n-1}
$\mathcal{SU}(2)$ Multiplexor (up to diag. gate)	\	1	3	7	15	31	63	127	$2^{n-1}-1$
Diagonal operator	0	2	6	14	30	62	126	254	2^n-2
Generalized Toffoli	0	1	6	24	48	72	96	120	1 if $n=2$, 6 if $n=3$, $24n-72$ if $n>3$
State preparation									
Rotations mult.	0	2	8	22	52	114	240	494	$2^{n+1}-2n-2$
$\mathcal{SU}(2)$ mult.	0	1	4	11	26	57	120	247	2^n-n-1
Schmidt's decomposition									$\begin{cases} \alpha\times 2^n - 2\times 2^{n/2} - \frac{1}{3} + 2\delta_p \text{ if } n \text{ even} \\ \alpha\times 2^n - 3\times 2^{(n-1)/2} + \frac{4}{3} + 4\delta_p \text{ if } n \text{ odd} \end{cases}$
+ QSD ($p=2$)	0	1	5	9	23	47	103	215	$\alpha=\frac{23}{24},\ \delta_p=1$
+ QSD ($p=4$)	0	1	5	9	23	35	79	**137**	$\alpha=\frac{490}{768},\ \delta_p=3$
BFGS	**0**	**1**	**3**	**6**	**14**	**29**	**61**	x	$\approx\lceil\frac{1}{4}(2^{n+1}-2n-2)\rceil$
Unitary synthesis									
BFGS									
BFGS	0	**3**	**14**	**61**	**252**	x	x	x	$\lceil\frac{1}{4}(4^n-3n-1)\rceil$
BFGS (up to diag. gate)	0	2	13	58		x	x	x	$<\lceil\frac{1}{4}(4^n-3n-1)\rceil$
QSD									$\alpha_p\times 4^n - 3\times 2^{n-1} + \frac{1}{3} + \delta_p$
QSD ($p=2$)	0	3	20	100	444	1,868	7,660	31,020	$\alpha_p=\frac{23}{48},\ \delta_p=1$
QSD ($p=3$)	0	3	14	76	348	1,484	6,124	24,876	$\alpha_p=\frac{37}{96},\ \delta_p=1$
QSD ($p=4$)	0	3	14	61	282	**1,214**	**5,038**	**20,526**	$\alpha_p=\frac{245}{768},\ \delta_p=3$
Householder									
With rotations mult.	0	15	118	740	3,212	11,956	43,164	157,572	$2\times 4^n - 2n\times 2^n - 4 + (2^n-1)\#\text{Toff}_{[78]}$
With $\mathcal{SU}(2)$ mult.	0	8	74	527	2,278	8,045	27,156	92,795	$4^n-(2n-3)2^n+(2^n-1)\#\text{Toff}_{[78]}-n-5$
With Schmidt, QSD ($p=2$)	0	9	77	505	2,130	6,978	23,346	71,522	Eq. (4.37) and Eq. (4.38)
With Schmidt, QSD ($p=4$)	0	9	77	505	2,040	6,492	19,977	58,091	Eq. (4.37) and Eq. (4.38)

Algorithm	CNOT counts $n=$ 9	10	11	12	13	14	15	16	Formulas
Specific operators									
$R_{x/y/z}$ multiplexor	256	512	1,024	2,048	4,096	8,192	16,384	32,768	2^{n-1}
$\mathcal{SU}(2)$ Multiplexor (up to diag. gate)	255	511	1,023	2,047	4,095	8,191	16,383	32,767	$2^{n-1}-1$
Diagonal operator	510	1,022	2,046	4,094	8,190	16,382	32,766	65,534	2^n-2
Generalized Toffoli	144	168	192	216	240	264	288	312	1 if $n=2$, 6 if $n=3$, $24n-72$ if $n>3$
State preparation									
Rotations mult.	1,004	2,026	4,072	8,166	16,356	32,738	65,504	131,038	$2^{n+1}-2n-2$
$\mathcal{SU}(2)$ mult.	502	1,013	2,036	4,083	8,178	16,369	32,752	65,519	2^n-n-1
Schmidt's decomposition									$\begin{cases} \alpha\times 2^n - 2\times 2^{n/2} - \frac{1}{3} + 2\delta_p \text{ if } n \text{ even} \\ \alpha\times 2^n - 3\times 2^{(n-1)/2} + \frac{4}{3} + 4\delta_p \text{ if } n \text{ odd} \end{cases}$
+ QSD ($p=2$)	447	919	1,871	3,799	7,663	15,447	31,023	62,295	$\alpha=\frac{23}{24},\ \delta_p=1$
+ QSD ($p=4$)	**291**	**595**	**1,223**	**2,491**	**5,047**	**10,203**	**20,535**	**41,307**	$\alpha=\frac{490}{768},\ \delta_p=3$
BFGS	x	x	x	x	x	x	x	x	$\approx\lceil\frac{1}{4}(2^{n+1}-2n-2)\rceil$
Unitary synthesis									
BFGS									
BFGS	x	x	x	x	x	x	x	x	$\lceil\frac{1}{4}(4^n-3n-1)\rceil$
BFGS (up to diag. gate)	x	x	x	x	x	x	x	x	$<\lceil\frac{1}{4}(4^n-3n-1)\rceil$
QSD									$\alpha_p\times 4^n - 3\times 2^{n-1} + \frac{1}{3} + \delta_p$
QSD ($p=2$)	124,844	500,908	2,006,700	8,032,940	x	x	x	x	$\alpha_p=\frac{23}{48},\ \delta_p=1$
QSD ($p=3$)	100,268	402,604	1,613,484	6,460,076	x	x	x	x	$\alpha_p=\frac{37}{96},\ \delta_p=1$
QSD ($p=4$)	**82,862**	**332,974**	**1,334,958**	**5,345,966**	x	x	x	x	$\alpha_p=\frac{245}{768},\ \delta_p=3$
Householder									
With rotations mult.	588,652	2,248,532	8,736,572	34,340,644	135,970,572	540,737,268	2,155,937,500	8,608,284,356	$2\times 4^n - 2n\times 2^n - 4 + (2^n-1)\#\text{Toff}_{[78]}$
With $\mathcal{SU}(2)$ mult.	328,034	1,203,017	4,548,400	17,575,703	68,886,270	272,350,949	1,082,293,964	4,313,513,651	$4^n-(2n-3)2^n+(2^n-1)\#\text{Toff}_{[78]}-n-5$
With Schmidt, QSD ($p=2$)	262,026	848,106	3,502,634	11,909,290	52,543,162	182,508,154	825,554,090	2,886,297,322	Eq. (4.37) and Eq. (4.38)
With Schmidt, QSD ($p=4$)	197,028	616,896	2,435,360	8,161,924	**35,375,344**	**122,376,244**	**550,521,776**	**1,923,509,236**	Eq. (4.37) and Eq. (4.38)

Table 5.5: CNOT counts and formulas for different synthesis methods.

Chapter 6

Reuse Method for Quantum Circuits Synthesis

Contents

In the Chapters 4 and 5 we either improved standard algebraic methods or standard numerical methods. There exists a less typical method that focuses on a decomposition of the operator as a linear combination of other operators chosen from a given set. This method enables to reuse optimized circuits in order to implement more complex operators [104]. To our knowledge, this is the only method using such a technique. This method, which we informally call the *reuse method*, has been shown to be efficient on specific cases [104]. Our objective in this chapter is to determine whether this method can be efficiently extended to a general framework for circuit synthesis.

The chapter is organized as follows. In Section 6.1, we recall the main principles of the reuse method. In Section 6.2, we select the groups that can be used in synthesizing circuits via the reuse method. In Section 6.3, we study the potential group candidates.

6.1 Presentation of the Reuse Method

The reuse method has emerged from the following motivation: if we know how to implement circuits (supposedly efficiently), can we directly reuse these circuits in order to implement new operators?

Based on this idea, Klappenecker and Rötteler replied in the affirmative [104]. Below is a simplified version of [104, Th. 6].

Theorem 6.1.1. *Let $G \subset \mathcal{U}(2^m)$ be a group of order 2^n, and $T = (t_1, ..., t_n)$ be such that any member g of G can be written as $g = t_1^{\alpha_1} ... t_n^{\alpha_n}$ with $\alpha_i \in \{0, 1\}$). Suppose*

$$A = \sum_{g \in G} \beta_g g \tag{6.1}$$

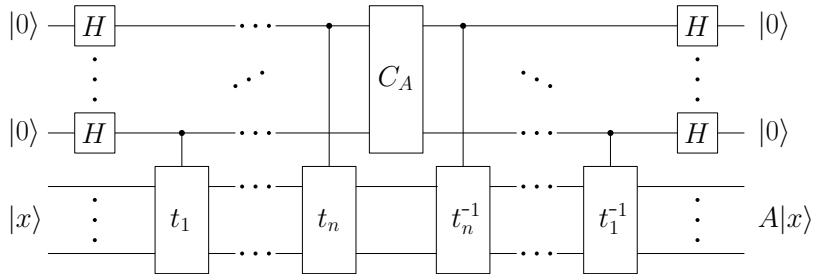


Figure 6.1: Quantum circuit implementing a linear combination of operators.

with $A \in \mathcal{U}(2^m)$ and define the coefficient matrix $C_A = (\beta_{g^{-1}h})_{g,h\in G}$. Then the coefficients $(\beta_g)_{g\in G}$ can be chosen such that C_A is unitary and the operator A can be implemented as depicted in Figure 6.1.

This remains a simplified version, sufficient for the rest of our study. An illustration of the method can be found in [104, Sec. 3], where there is an implementation of the Hartley transform via a linear combination of powers of the Fourier transform.

A key point in the use of this method is the distribution of information between the group and the matrix of coefficients. When the group contains sufficient information, such as the Fourier transform powers group, the coefficient matrix is easy to compute and the efficiency of the quantum Fourier transform synthesis is used to produce an efficient circuit on non trivial operators. An alternative would be to consider the problem in the other direction: the group is simple, contains little information but the matrix of coefficients — which now has a maximum of information — has a structure that makes its implementation efficient.

For example, if the group is circulant then the matrix of coefficients will be circulant and diagonalizable in the Fourier basis [40]. If the group is symmetric and its elements are involutive, then the matrix will be diagonalizable in the Hadamard basis (similarly to the MS gate, see Section 5.3). In these cases, information can be predominantly contained in the coefficient matrix but the implementation of the coefficient matrix, although inevitably costly in terms of gates, is much simpler than for any generic operator (see the article by Bullock and Markov for the implementation of diagonal operators [36]).

However among all the possible matrix groups, some are more suitable than others for a generic synthesis method. In the next section we narrow our research by investigating the theoretical properties of "good" groups for general synthesis.

6.2 Characterization of Candidate Groups G

In this section we discuss under which conditions the reuse method can be used as a generic method for the synthesis of circuits.

We can already eliminate the case where the group G is Abelian. Indeed, in this case the matrices of the group G commute in pairs and are therefore simultaneously diagonalizable, just like any member of the span of G. One cannot reach all unitary matrix but only those diagonalizable in a specific basis.

Ideally, the group G should be built easily for any number of qubits either with an adaptable construction for any n or with a recursive approach. In fact, we can show how to construct a solution group K and its matrix of coefficients for $n + m$ qubits from a solution group G for n qubits and a solution group H for m qubits.

We use can the properties of the tensor product to construct the group K. Indeed, by setting $K = G \otimes H$, provided that $U(2^n) \subseteq span(G)$ and $U(2^m) \subseteq span(H)$ then we have $U(2^{n+m}) \subseteq span(G \otimes H)$. Recall the identity

$$(g_1 \otimes g_2)(h_1 \otimes h_2) = (g_1 h_1) \otimes (g_2 h_2) \tag{6.2}$$

which is used to provide an expression of the coefficient matrix associated with K:

$$\begin{aligned} C_K = (\beta_{g^{-1}h})_{g,h \in G \otimes H} &= (\beta_{(g_1^{-1} \otimes h_1^{-1})^{-1}(g_2 \otimes h_2)})_{g_1,g_2 \in G, h_1,h_1 \in H} \\ &= (\beta_{(g_1^{-1} g_2) \otimes (h_1^{-1} h_2)})_{g_1,g_2 \in G, h_1,h_1 \in H} \quad . \end{aligned} \tag{6.3}$$

With an appropriate ordering of K, the matrix C_K can be expressed as

$$C_K = \begin{pmatrix} \beta_{(g_1 g_1, h_1 h_1)} & \cdots & \beta_{(g_1 g_n, h_1 h_1)} & \beta_{(g_1 g_1, h_1 h_2)} & \cdots & \beta_{(g_1 g_n, h_1 h_2)} & \cdots \\ \vdots & & \vdots & \vdots & & \vdots & \\ \beta_{(g_n g_1, h_1 h_1)} & \cdots & \beta_{(g_n g_n, h_1 h_1)} & \beta_{(g_n g_1, h_1 h_2)} & \cdots & \beta_{(g_n g_n, h_1 h_2)} & \cdots \\ \beta_{(g_1 g_1, h_2 h_1)} & \cdots & \beta_{(g_1 g_n, h_2 h_1)} & \beta_{(g_1 g_1, h_2 h_2)} & \cdots & \beta_{(g_1 g_n, h_2 h_2)} & \cdots \\ \vdots & & \vdots & \vdots & & \vdots & \\ \beta_{(g_n g_1, h_2 h_1)} & \cdots & \beta_{(g_n g_n, h_2 h_1)} & \beta_{(g_n g_1, h_2 h_2)} & \cdots & \beta_{(g_n g_n, h_2 h_2)} & \cdots \\ \vdots & & \vdots & \vdots & & \vdots & \end{pmatrix}. \tag{6.4}$$

Thus, if a series of operations P factors C_G and a series of operations Q factors C_H, then $(P \otimes I)$ block-factorizes C_K and $(I \otimes Q)$ factorizes each block of C_K. Thus *a priori* $(P \otimes Q)$ factorizes C_K.

Therefore, if a solution for one qubit has been found, we can generate a solution for an arbitrary number of qubits by successive tensor products. Now, because the available memory is limited[1], it is desired to minimize the number of auxiliary qubits per logical qubits especially if additional qubits are necessary for error correcting codes [181]. In our study the size of the group G has been fixed to a maximum of 8 elements so as to have only 3 auxiliary qubits per logic bit. This accounts for the fact that quantum memory is expensive.

Only a few potential groups then satisfy the above restrictions:

- the projective Pauli group,
- the dihedral group over 3 qubits,
- the quaternion group.

6.3 Study of the Candidates

The two 8-element groups – quaternion and dihedral group – are very similar: we only consider the latter. Indeed, the results on one of the two groups are immediately transposable to the other group.

This section analyzes first the base cases: the projective Pauli group and the dihedral group. In a second step, we discuss the behavior of the factorization mentioned in Section 6.2 for the case of two qubits in the context of the dihedral group.

[1] Simulating quantum computation on a conventional computer is known to be expensive [148] since a linear increase in the number of manipulated qubits yields an exponential increase in the size of the required memory.

6.3.1 The Projective Pauli Group

In the original publication, [104, Th. 6] has been extended to the case of projective groups in [104, Th. 7]. The particular projective group that we consider is

$$G = \{I, X, Z, XZ\}. \tag{6.5}$$

By setting $A = a_0 I + a_1 X + a_2 Z + a_3 XZ$ the associated coefficient matrix is

$$C_A = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & -a_3 & a_0 & -a_1 \\ a_3 & -a_2 & a_1 & -a_0 \end{pmatrix}. \tag{6.6}$$

Klappenecker and Rötteler [104, Eq. 12] gave the factorization

$$CNOT \times CNOT^{(2,1)} \times (H \otimes I_2) \times C_A \times (H \otimes I_2) \times CNOT = A \otimes I_2 \tag{6.7}$$

with

$$CNOT^{(2,1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

This shows that the synthesis of C_A is as difficult as the synthesis of A. No improvement can therefore be achieved with this group.

6.3.2 The dihedral group

The idea is to get rid of the projective character of the Pauli group by adding matrices to the G group, i.e., with

$$G = \{I, -I, X, -X, Z, -Z, XZ, -XZ\}. \tag{6.8}$$

The coefficient matrix then becomes

$$C_A = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \\ a_1 & a_0 & a_3 & a_2 & a_5 & a_4 & a_7 & a_6 \\ a_2 & a_3 & a_0 & a_1 & a_7 & a_6 & a_5 & a_4 \\ a_3 & a_2 & a_1 & a_0 & a_6 & a_7 & a_4 & a_5 \\ a_4 & a_5 & a_6 & a_7 & a_0 & a_1 & a_2 & a_3 \\ a_5 & a_4 & a_7 & a_6 & a_1 & a_0 & a_3 & a_2 \\ a_7 & a_6 & a_5 & a_4 & a_2 & a_3 & a_0 & a_1 \\ a_6 & a_7 & a_4 & a_5 & a_3 & a_2 & a_1 & a_0 \end{pmatrix}. \tag{6.9}$$

The best factorization that we have found is

$$P \times C_A \times P^\dagger = \begin{pmatrix} I & & & \\ & U & & \\ & & I & \\ & & & U \end{pmatrix} = I \otimes \Lambda(U) \tag{6.10}$$

where

$$P = (SWAP \otimes I) \times (I \otimes SWAP) \times (\Lambda(Z) \otimes I) \times H^{\otimes 3} \tag{6.11}$$

and where U is some arbitrary 2×2 unitary matrix, a priori not simpler to synthesize than the matrix A.

We can then conclude that no improvement can neither be found for this group.

6.3.3 Factorization for two qubits

In this section, we highlight the fact that the factorization procedure envisioned in Section 6.2 is not so simple to use, and that it does not necessarily provide a usable decomposition.

Consider indeed an operator A on 2 qubits. Using the dihedral group, the block factorization on C_A would then lead to

$$\begin{pmatrix} I & & & \\ & V & & \\ & & I & \\ & & & V \end{pmatrix} \tag{6.12}$$

with V a 2 by 2 block-matrix with blocks of size 8 by 8. Applying the same factorization on each block of V gives a matrix of the shape

$$\begin{pmatrix} I & & & & 0 & & & \\ & U_1 & & & & U_2 & & \\ & & I & & & & 0 & \\ & & & U_1 & & & & U_2 \\ 0 & & & & I & & & \\ & U_3 & & & & U_4 & & \\ & & 0 & & & & I & \\ & & & U_3 & & & & U_4 \end{pmatrix}, \tag{6.13}$$

with U_1, U_2, U_3 and U_4 arbitrary matrices of size 2×2, such that $\begin{pmatrix} U_1 & U_2 \\ U_3 & U_4 \end{pmatrix}$ is unitary. Synthesizing A therefore corresponds to synthesizing this matrix, which does not seem too less costly. This hints at the fact that extending the study to larger groups might not trivially help in getting a working solution.

Chapter 7

Conclusion and Perspectives on Generic Circuits Synthesis

We started this part of the thesis with the goal

Synthesize an operator on n qubits in a reasonable amount of time with a circuit of size $\alpha \times 4^n$

and we studied its feasibility for extreme values of n and α, i.e., the largest values possible for n and the smallest ones for α. We first proposed ZUNQRF in Chapter 4, a LAPACK-like routine for the QR factorization of a unitary matrix based on Householder transformations. We transformed the problem of the synthesis of an arbitrary operator into the synthesis of several Householder operators which itself is roughly equivalent to the preparation of quantum states. We derived analytical formulas for the flop counts and the size of the resulting quantum circuits. Considering that "a reasonable amount of time" corresponds here to one hour, we achieve our goal with the Householder framework for new problem sizes $n = 13, 14, 15$ at the cost of a twice as large α compared to the state-of-the-art method. With the help of multithreaded architectures or GPUs we can hope to reach even larger problem sizes: $n = 17$, $n = 18$ qubits, maybe even more.

On the other end of the spectrum, we used numerical optimizers to produce circuits with the lowest possible α in Chapter 5. We considered circuit skeletons with parameterized gates where a numerical optimizer computes a set of angles that minimizes the error between the target operator and the one implemented by the circuit. With this simple framework, we exhibited universal circuit skeletons, i.e., circuits from which we can empirically always find a set of angles such that the circuit instantiated implements the desired operator. We provided universal topologies for several gate sets like the standard universal gate set for trapped-ion based quantum circuits or the standard continuous gate set for superconducting qubits. More importantly, the size of those topologies is optimal in the sense that α cannot be lower. We also highlighted that the number of iterations required to converge is correlated to the extra number of entangling gates in the circuits. Therefore it is possible to trade some quantum resources to consistently decrease the classical resources. This improvement in the synthesis of small operators is useful in itself as a subtask in a global compilation framework but it also improves the state-of-the-art methods like the QSD or our Householder framework. By stopping the recursion of the QSD at 4 qubits for instance we synthesize circuits with a lower α than the one usually given by the QSD.

Our results from Chapters 4 and 5 can be summarized by the graph given in Fig 7.1. Given the amount of classical resources on the x-axis and the amount of quantum resources on the y-axis, we can place every synthesis method. The ideal method is the one closest to the origin of the graph. More realistically, we can imagine a theoretical curve such that you cannot synthesize a

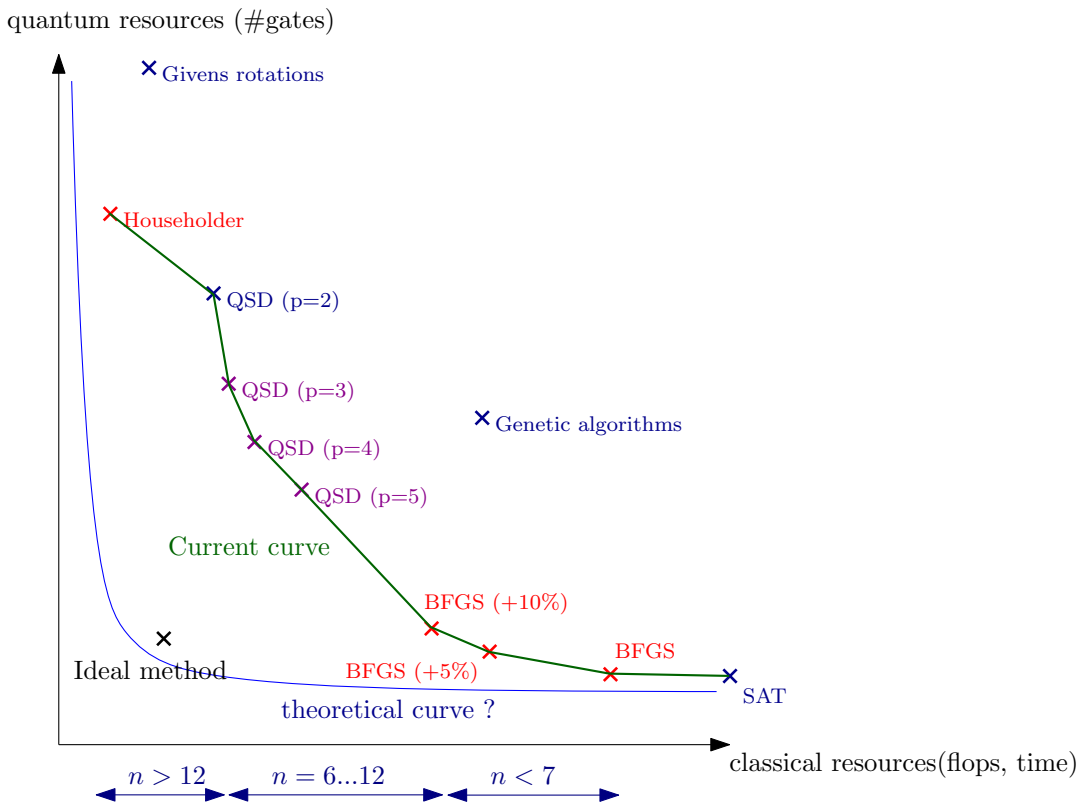


Figure 7.1: Synthesis methods in the quantum resources/classical resources plane.

circuit with a method that would lie below this curve. In other words, the curve gives the lowest quantum circuit complexity one can hope with a specific amount of classical resources. A non-exhaustive overview of the state of the art is is given by the blue crosses: SAT solvers for optimal synthesis, algebraic methods like the QSD or the QR via Givens rotations for faster but less optimal synthesis, and genetic algorithms in between that have shown encouraging but limited results. Of course, other synthesis schemes like Knill's decomposition or block-ZXZ have their own place in the graph but, for clarity, we did not put them.

With the results from this thesis we can add several methods, represented in red in the graph: first, the Householder framework from Chapter 4 which provides, to our knowledge, the fastest method while keeping a relatively low complexity. Then we have the BFGS based synthesis framework from Chapter 5 which provides asymptotically optimal circuits but at the cost of a lot of classical resources. Then we can add the variations around BFGS: by adding a percentage of the total number of CNOT gates in our topologies we can decrease the number of iterations required for BFGS to converge and thus the total of classical resources. These improvements have also consequences on the QSD method. For each size at which we stop the recursion, we have a new tradeoff quantum/classical: the larger the stopping size is the shorter the final quantum circuit is but the longer it is to compute the optimal synthesis of each small operator. The reuse method, which is a bit apart from the other methods, is more difficult to place so we put it aside. Finally, this gives us the green curve which corresponds to the current state of the art. As we do not give real figures about the number of quantum and classical resources, we have added an approximate idea of what problem sizes can be addressed with the current green curve.

Can we improve the Householder framework? In terms of classical resources, it is complicated. The strong scaling experiment has shown that the performance of ZUNQRF is almost as good as

the matrix/matrix product which is probably the most optimized routine in scientific computing. In other words, unless a new framework with less theoretical flops is found, any improvement of the Householder method is dependent on the improvements done on the matrix/matrix product. But these improvements rely on experts in the field. In terms of circuit size, the main bottleneck is the optimization of the product of isometries: when n is odd, the concatenation of state preparation and state de-preparation can we simplified with Schmidt's decomposition as the product of two operators on $\lfloor n/2 \rfloor$ qubits and two isometries on $\lceil n/2 \rceil$ qubits. For the moment we have to consider the last product as an entire operator on $\lceil n/2 \rceil$ qubits but, similarly to the product state preparation/de-preparation that can be simplified, we think it may be possible to simplify this product of isometries.

Concerning the BFGS based algorithm, there is room for improvement. The first obvious improvement is a theoretical one: can we prove that the lower bounds are tight? Can we at least show it for $n = 3$ qubits? The structure of the cost function is in fact well understood: this is a trigonometric polynomial with variables the angles of the rotations in the circuit. But with k parameters in the circuit, the polynomial contains 2^k monomials and with $k = 4^n$ it is intractable to have a complete description of the cost function even for $n = 3$. Still, we can expect:

- that some black-box optimization algorithms tailored for trigonometric polynomials exist and will give better results than the BFGS algorithm.
- Some analytical work can be done in the hope to prove that the trigonometric polynomials we encounter always have a zero.

We are not familiar with the optimization of trigonometric polynomials and we did not find something useful in the literature.

Another question regarding the BFGS framework: can we extend it to synthesize specific operators? When the operator is not random and we expect that a short circuit can implement it, is it possible to use numerical optimization? We are quite pessimistic about it. For CNOT+$\mathcal{SU}(2)$ based quantum circuits, it involves finding a correct layout of CNOT gates which implies a combinatorial optimization on top of the standard numerical optimization. This may be done for 3, maybe 4 qubits but the scalability of such a method is probably not good at all. But we do not think that this is the main problem. An even more worrying aspect that we noticed is the local minima problem: when dealing with specific operators and circuits with only one solution, it is very hard to converge to the global minimum. To give an example, we generated a random topology with a few CNOT gates and we instantiated the topology with random angles. We tried to recover the associated operator by using the BFGS algorithm and we always fell in a local minimum. Even worse: starting from the initial set of angles used to generate the operator and taking an initial perturbation of 10^{-2} of each component as a starting point of the BFGS algorithm, we still fell in a local minimum. This is the main reason why we are pessimistic. Even with MS gates that do not require to optimize a layout, the numerical optimizer has trouble finding the global minimum. We did not try global optimization algorithms yet, it would be interesting to see the scalability of such methods for small problem sizes.

Finally, we have proposed a classical algorithm for computing the error and the gradient. In terms of classical resources, this represents three classical simulations of a quantum circuit and this limits the range of application of the framework. The simulation of a quantum circuit is exponential in the number of qubits and linear in the number of gates, which is also exponential in the number of qubits. What if we can execute the circuit directly on a quantum machine? This idea is at the heart of variational algorithms used for instance to solve optimization problems [57, 140] or in quantum chemistry [151]. Notably, the idea of computing an error via a quantum computer for

compilation is developed in [99] where they propose a framework for compiling unitaries on NISQ architectures. Our work is part of the same issue but we propose a different point of view where we focused on random operators with theoretical lower bounds that we try to validate. We hope our work will give better understandings of the behavior of the classical optimization part of such variational algorithms.

Part C

Linear Reversible Circuits Synthesis

Chapter 8

Introduction to Linear Reversible Circuits Synthesis

Contents

8.1 Motivation

During Part B of this thesis, we saw that the synthesis of operators in $\mathcal{SU}(2^n)$ is very costly in classical and quantum resources. It can only be used in very specific cases and must be integrated into a more general compilation framework. To synthesize and optimize quantum circuits beyond the intrinsic limits of synthesis in $\mathcal{SU}(2^n)$ we must turn to alternative representations of an operator. It is no longer a question of using a matrix in $\mathcal{SU}(2^n)$ as input but a compact representation of the operator that we can handle more easily: this can be a Boolean formula for the synthesis of oracles, a quantum decision diagram, a quantum circuit, etc. We lose the universal character of the synthesis — not all operators can be described compactly — but we gain in scalability with the idea that a given algorithm can be divided into several sub-operators more efficient to process. This is the whole challenge of quantum compilation: navigating between the different efficient representations of an operator to extract the smallest quantum circuit possible. In this context, the synthesis of circuits focuses on specific non-universal classes of operators. We can cite the Clifford operators, the CNOT + T circuits (the corresponding operators are called phase polynomials), or the oracles which have attracted much attention in the literature. Knowing how to effectively synthesize these types of operators is crucial because today's compilers rely heavily on these particular cases. For example, the optimization of a quantum circuit can be reduced to a succession of

synthesis of CNOT + T circuits and is at the heart of the state-of-the-art algorithms Tpar [10] and Topt [80].

This part of the thesis is dedicated to the synthesis of the so-called *linear reversible* operators. They can be synthesized with the CNOT gate solely and we will use the term CNOT circuits for reversible linear circuits during this part even though other sets of gates can synthesize linear reversible operators (fan-in or fan-out gates for instance [81]). These operators are part of two classes of operators mentioned above: the Clifford operators and the phase polynomials. As we have said, phase polynomials have been the basis of modern compilers for a short decade and have allowed enormous progress in the optimization of quantum circuits. However, most of the efforts have focused on reducing the number of T gates in the circuit. We explained in Chapter 2 that considering only this metric could be misleading because the CNOT cost can equal or even exceed that of the T gates. Optimizing the synthesis of CNOT circuits will reduce the CNOT cost of CNOT + T circuits and therefore the cost of quantum circuits in general. Clifford circuits represent the second major application of the synthesis of CNOT circuits. Clifford operators play an essential role in several fields such as error correcting codes. They are therefore essential for the realization of a fault-tolerant quantum computer, among many other applications (quantum tomography, randomized benchmarking, etc.). We recall that a Clifford circuit is a circuit containing only the gates H, S, CNOT, and Pauli. The structure of this group is better and better understood. Several canonical forms have been demonstrated [2,134]. They break down a Clifford operator into several layers of operators made up of a single type of gate, i.e., for instance, layers of H gates, layers of S gates, or layers of CNOT gates. The layers of CNOT circuits in these canonical forms represent the major part of a Clifford circuit in terms of number of gates. Therefore optimizing the synthesis of CNOT circuits has an immediate impact on the synthesis of Clifford operators as well.

The outline of this introductory chapter is the following: we recall the notions about linear reversible circuits and the state of the art in Section 8.2. We end with a summary of our contributions in Section 8.3.

8.2 Background and State of the Art

8.2.1 Background on Linear Reversible Circuits

Let $\mathbb{F}_2$ be the Galois field of two elements. A Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is said to be linear if

$$f(x_1 \oplus x_2) = f(x_1) \oplus f(x_2)$$

for any $x_1, x_2 \in \mathbb{F}_2^n$ where $\oplus$ is the bitwise XOR operation. Let e_k be the k-th canonical vector of $\mathbb{F}_2^n$. By linearity we can write for any $x = \sum_{k=1}^{n} x_k e_k$ where $x_k \in \{0, 1\}$

$$f(x) = f\left(\sum_{k=1}^{n} x_k e_k\right) = \sum_{k=1}^{n} x_k f(e_k)$$

and the function f can be represented with a column vector $\alpha = [f(e_1), ..., f(e_n)]^T$, also called a *parity*, such that $f(x) = \alpha \cdot x$, where $\cdot$ stands for the scalar product on F_2^n and $(-)^T$ is the matrix-transpose operation. This easily extends to the n-input m-output functions $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ where f is defined by an $m \times n$ Boolean matrix A such that $f(x) = Ax$. In the case of linear reversible Boolean functions, $n = m$ and we have a one-to-one correspondence between the inputs and the outputs. We then consider n-input n-output functions f for which the equation $y = f(x) = Ax$ must have a unique solution for any $y \in \mathbb{F}_2^n$. In other words, the matrix A must be invertible in

$\mathbb{F}_2$ and there is a one-to-one correspondence between the linear reversible functions of arity n and the invertible Boolean matrices of size n. This was notably used to count the number of different linear reversible functions of n inputs in [152]. Namely, there are

$$\prod_{i=0}^{n-1}(2^n - 2^i)$$

linear reversible operators on n qubits. Note that the i-th row of A corresponds to the parity held by the i-th qubit after application of A. The application of two successive operators A and B is equivalent to the application of the operator product BA. The application of the inverse operator of A is equivalent to the application of A^{-1}.

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

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

where the first qubit in the state x_1 is the *control qubit* and the second qubit in the state x_2 is the *target qubit*. It is straightforward to check that the CNOT gate is a linear reversible gate. It can also be shown to be universal for linear reversible circuit synthesis: any linear reversible function of arity at least 2 can be implemented by a reversible circuit containing only CNOT gates [2]. In the following chapters, we aim at producing CNOT-based reversible circuits for any linear reversible functions. Note that the choice of the CNOT gate as a universal gate set is not a necessity. Other universal linear reversible gates may be of interest, but the CNOT gate is part of almost every universal gate sets in current architectures so it is natural to focus on CNOT circuits.

The matrix representation of the CNOT gate controlled by the line j acting on line $i \neq j$ is written $E_{ij} = I + e_{ij}$ where I is the identity matrix and e_{ij} the elementary matrix with all entries equal 0 but the component (i, j) where its value is 1. Therefore, finding a CNOT circuit implementing an operator A is equivalent to finding a sequence of matrices $(E_{i_k j_k})_{1 \leq i_k, j_k \leq n}$ such that

$$\prod_{k=1}^{N} E_{i_k, j_k} = A.$$

Using the fact that $E_{ij}^{-1} = E_{ij}$, it is more convenient to rewrite the synthesis problem as a reduction of A to the identity operator I

$$\prod_{k=N}^{1} E_{i_k, j_k} A = I$$

because we now show that the synthesis problem can be reformulated in terms of elementary operations applied on A. Given a CNOT with control j and target i applied on an operator A, the updated operator $E_{ij}A$ can be deduced from A with an elementary row operation :

$$r_i \leftarrow r_i \oplus r_j,$$

writing r_k for the k-th row of A. Therefore, if one can compute a sequence of row operations transforming A into the identity operator then one can construct a circuit implementing A by concatenating the CNOT gates associated to each row operation. With a standard Gaussian elimination algorithm it is always possible to reduce an invertible operator to the identity with row operations. This gives a simple proof that the CNOT gate is universal for linear reversible circuit

synthesis. Yet, in some cases we will also authorize the use of column operations. A column operation $c_j \leftarrow c_i \oplus c_j$ is equivalent to right multiplying A by E_{ij}. Overall by combining row and column operations we get

$$\prod_{(i_1,j_1)} E_{i_1 j_1} \times A \times \prod_{(i_2,j_2)} E_{i_2 j_2} = I$$

and finally

$$A = \prod_{(i_1,j_1)} E_{i_1 j_1} \prod_{(i_2,j_2)} E_{i2j2}$$

so we recover a CNOT circuit for the implementation of A.

Thus, synthesizing a linear reversible function into a CNOT-based reversible circuit is equivalent to transforming an invertible Boolean matrix A to the identity by applying elementary row and column operations. From now on we will privilege this more abstract point of view because it gives more freedom and often appears clearer for the design of algorithms. We note by $\text{Row}(i,j)$ the elementary row operations $r_j \leftarrow r_i \oplus r_j$ and $\text{Col}(i,j)$ the elementary column operations $c_j \leftarrow c_i \oplus c_j$.

We now review the main algorithms designed for the synthesis of CNOT circuits.

8.2.2 State of the Art

Gaussian Elimination

The first proposed algorithm is a standard Gaussian elimination algorithm for matrices in $\mathbb{F}_2$. The algorithm consists of two parts: first, the operator A is transformed into an upper triangular operator by zeroing the subdiagonal elements of each column. Then the first part is repeated on the triangular operator: the superdiagonal elements are zeroed column by column.

More precisely, given a lower triangular operator L — the upper case can be treated similarly — the Gaussian elimination algorithm can be summarized as follows: by applying elementary row operations $\text{Row}(i,j)$ with $i < j$ we zero the subdiagonal elements of L without changing its triangular shape. This process is performed column by column, starting from the first one. This way one can easily see that if the k first columns are treated, applying row operations $\text{Row}(i,j)$ with $k < i < j$ will not change the treated columns as all their elements are 0. Since L is invertible, when zeroing the k-th column the k-th row is necessarily equal to $L[k,:] = e_k^T$ hence one can always add the row k to any row $j > k$ to zero the entry $L[j,k]$. This guarantees the good behavior of the algorithm for any input matrix.

To triangularize an operator the same procedure is applied but one has to be careful to ensure that the diagonal element on the current column is not zero. If it is not the case an extra row operation is needed.

Since one row operation sets to zero only one element at a time, we need at most $\frac{n(n-1)}{2}$ row operations to zero every subdiagonal elements of L, leading to a worst-case complexity of $\mathcal{O}(n^2)$ for the synthesis of a generic linear reversible operator.

For completeness, we give the pseudo-code in Algorithm 8.1.

Patel-Markov-Hayes Algorithm (PMH)

In [152] Patel, Markov and Hayes proposed an algorithm with worst-case complexity $\mathcal{O}(n^2/\log_2(n))$. Besides, they also showed that this worst-case complexity is optimal, meaning that there exist oper-

Algorithm 8.1: Gaussian Elimination algorithm on an operator A.

Require: $n \geq 0, \quad A \in \mathbb{F}_2^{n \times n}$
Ensure: C is a CNOT-circuit implementing A

$C \leftarrow [\,]$
for i = 1 to n **do**
  if A[i,i] != 1 **then**
    $diag\_is\_zero = true$
  end if
  for j = i+1 to n **do**
    if A[j,i] = 1 **then**
      if $diag\_is\_zero$ **then**
        C.append(CNOT(j, i))
        $A[i,:] \leftarrow A[j,:] \oplus A[i,:]$
        $diag\_is\_zero = false$
      end if
      C.append(CNOT(i, j))
      $A[j,:] \leftarrow A[j,:] \oplus A[i,:]$
    end if
  end for
end for
for i = 1 to n **do**
  for j = i+1 to n **do**
    if A[n-j+1,n-i+1] = 1 **then**
      C.append(CNOT($n-i+1, n-j+1$))
      $A[n-j+1,:] \leftarrow A[n-j+1,:] \oplus A[n-i+1,:]$
    end if
  end for
end for
return reverse(C)

ators that cannot be synthesized with less than $\mathcal{O}(n^2/\log_2(n))$ gates. In other words this algorithm is *asymptotically optimal*.

To prove this lower bound on the worst case complexity, note that for each CNOT we have $2\binom{n}{2}$ different choices for the control and the target, plus a "do nothing" gate. This means that we have at most

$$\left(2\binom{n}{2}+1\right)^k$$

different operators that can be implemented with a circuit with no more than k CNOTs. We want to find the smallest k such that

$$\left(2\binom{n}{2}+1\right)^k \geq \prod_{i=0}^{n-1}(2^n-2^i) \geq 2^{n(n-1)}$$

which will ensure us that we can potentially synthesize any operator with only k CNOT gates. Taking the $\log$ on both sides, we finally get

$$k = \frac{n^2}{2\log_2(n)} + o(n^2/\log_2(n)). \tag{8.1}$$

The PMH algorithm is an improvement over the Gaussian elimination method. Patel *et al.* [152] showed that in average $\mathcal{O}(\log_2(n))$ elements can be zeroed in one row operation and not only 1 as in the Gaussian elimination algorithm. To do so they designed a block version of the Gaussian elimination algorithm. By treating a block of m columns simultaneously, it is possible to zero $n-2^m$ rows of m elements with only one row operation each as there are only 2^m different words on m qubits. After zeroing those duplicate rows the rest of the block is zeroed with a standard Gaussian elimination algorithm. Then the algorithm pursues with another block of m columns, and so on until the operator is triangular. Similarly to the Gaussian elimination algorithm, the algorithm is repeated on the triangular operator.

If m is the size of the blocks, they showed that the number of row operations required to synthesize an operator is upper bounded by

$$\text{total row op. [152]} \leq \frac{n^2}{m} + 2n2^m + \text{ negligible terms} \tag{8.2}$$

Take $m = \alpha\log_2(n)$ with $0 < \alpha < 1$ and we have the desired complexity:

$$\text{total row op. [152]} \leq \frac{n^2}{\alpha\log_2(n)} + 2n^{1+\alpha} + \text{ negligible terms} \tag{8.3}$$

By taking α arbitrarily close to 1, the multiplicative constant is arbitrarily close to 1 but the term $2n^{1+\alpha}$ is also arbitrarily close to $2n^2$. We will see that this term is not negligible and distorts the concrete complexity. In practice $\alpha = 1/2$ produces the best results for intermediate values of n ($n < 100$) [152]. A pseudo-code is given Algorithm 8.2 in the case of a triangular operator L.

To our knowledge, the PMH algorithm is the best algorithm for the synthesis of linear reversible circuits when the the connectivity between the qubits is full. It is notably used in some recent circuit optimization algorithms [9, 10].

Steiner Trees and Hardware Constraints

The current technologies for quantum computers cannot afford full connectivity between the qubits. The qubits are arranged in a certain way in the hardware and CNOTs are possible only

Algorithm 8.2: PMH algorithm on a triangular operator L.

Require: $n \geq 0,\ \ 0 < m \leq n,\ \ L \in \mathbb{F}_2^{n \times n}$ triangular
Ensure: C is a CNOT-circuit implementing L

$C \leftarrow [\,]$
$NB \leftarrow \lceil n/m \rceil$ // number of blocks
for k=1 to NB **do**
  begin_block $\leftarrow (k-1) \times m + 1$
  end_block $\leftarrow \min(k \times m, n)$
  size_block $\leftarrow$ end_block-begin_block+1
  locations $\leftarrow$ zeros($2^{\text{size\_block}}$) // create a vector of size 2^m locating the last positions of subrows
  for $i =$ begin_block to n **do**
    subrow $\leftarrow Int(A[i,$ begin_block:end_block$])$
    if locations[subrow] != 0 **then**
      $j \leftarrow$ locations[subrow]
      C.append(CNOT(j, i))
      $L[i,:] \leftarrow L[j,:] \oplus L[i,:]$
    else
      locations[subrow] $\leftarrow i$
    end if
  end for
  for $i =$begin_block to end_block **do**
    for j = i+1 to n **do**
      if A[j,i] = 1 **then**
        C.append(CNOT(i, j))
        $A[j,:] \leftarrow A[j,:] \oplus A[i,:]$
      end if
    end for
  end for
end for
return reverse(C)

between neighboring qubits. For instance, the qubits can be arranged as a line (the Linear Nearest Neighbor architecture) or a grid. Some other examples of restricted connectivities were given in Fig 2.3 in Chapter 2. The Gaussian elimination algorithm or the PMH algorithm are designed assuming that any CNOT between any pair of qubits can be performed. They cannot be directly used for a hardware with a restricted connectivity. To overcome this, two solutions are possible:

1. there exist methods for rewriting a circuit so that it verifies the connectivity constraints. The rewriting consists generally in moving the qubits in the hardware with SWAP gates to execute every CNOT gate in the original circuit. Therefore these methods, sometimes called SWAP insertion methods, introduce an overhead somehow proportional to the size of the initial circuit. In other words, such rewriting methods perform well for small input circuits but are not as effective for large circuits.

2. If the original circuit is big enough (and this notion is not easy to quantify), or if the operator is given by its matrix representation, then direct synthesis methods that take into account the connectivity constraints are preferable. Indeed, SWAP insertion methods can increase the complexity by a factor of n, so a method relying on the PMH algorithm and a SWAP insertion method provides circuits of size $\mathcal{O}(n^3/\log_2(n))$ in the worst case. With direct methods, it is possible to synthesize linear reversible circuits for any architecture with at most $\mathcal{O}(n^2)$ CNOT gates [102, 146].

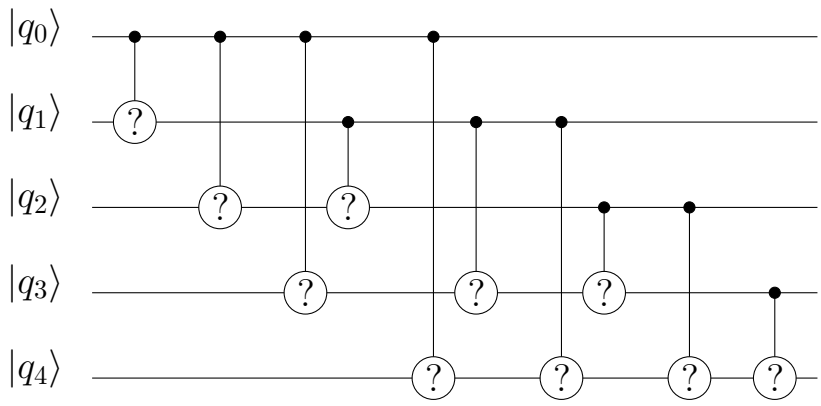
Recently, two concomitant papers proposed a modification of the Gaussian elimination algorithm to produce circuits compliant with a connectivity graph given as input of the algorithm [102, 146]. Both papers use Steiner trees to perform a custom Gaussian elimination: the operator is still synthesized column by column but the zeroing of one column is modified to respect the connectivity constraints. Given a set of vertices S in the connectivity graph G as the qubits for which the entry of the current column is 1, an efficient way to zero those entries is to follow a tree in G spanning the vertices in S. The entries of the current column are zeroed in the order given by the tree: from the leaves to the root. The two papers differ in the way they compute the sequence of row operations from the tree — we invite the reader to read the papers [102, 146] for more details.

Computing a Steiner tree with a minimum number of edges is equivalent to computing a minimum-weight Steiner tree in a graph whose edges are weighted to 1. This problem is NP-Hard but approximation algorithms with good approximation ratios exist in the literature. This algorithm relying on Steiner trees improves over rewriting methods found in modern compilers (namely, the QuilC [179] and t|ket⟩ [41] compilers). Very recently, in a preprint, Wu *et al.* improved this method with the use of non-cutting vertices [209].

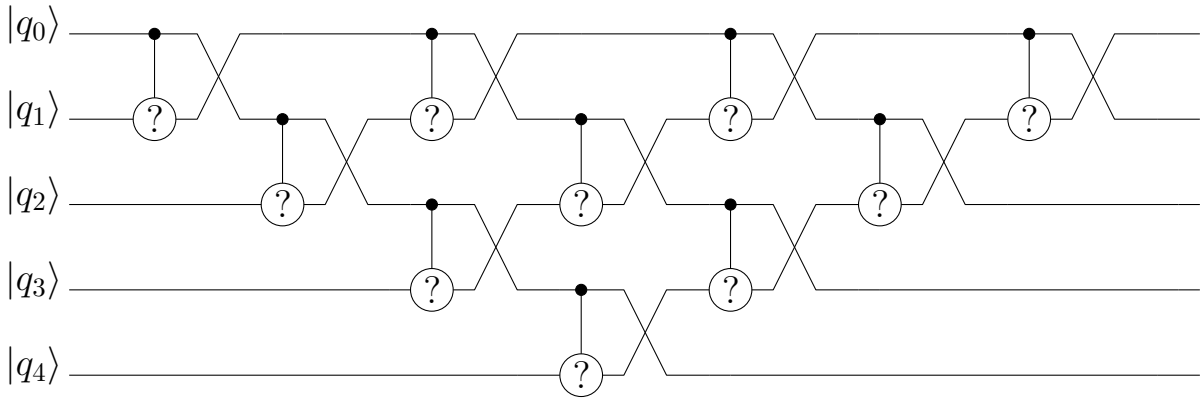
Depth Optimization Algorithms for LNN

So far we have presented state-of-the-art algorithms that optimize the size of the produced circuits. The other metric of interest is the depth of the CNOT circuits and some works proposed algorithms for this problem. In this section, we present algorithms for the LNN architecture. In Section 8.2.2 we will present algorithms for depth optimization in the case of full qubit connectivity.

We first present the algorithms for the LNN architecture because, surprisingly, the first papers on the subject directly focused on the depth optimization for the LNN architecture. In [130] Maslov showed how to reorganize the CNOTs applied during the Gaussian elimination algorithm in order to have a circuit of linear depth. More precisely, he considered a skeleton circuit with placeholders for all potential CNOTs that can be applied in the Gaussian elimination algorithm. This means that for a specific operator to synthesize, you only have to decide for each placeholder if there will be



(a) Skeleton circuit for 5 qubits. Each CNOT has a placeholder in the skeleton but its actual presence is conditioned by the operator to be synthesized. This is symbolized by the "?". The skeleton has CNOT depth $2n-3$.



(b) Skeleton circuit for 5 qubits compliant with the LNN architecture. The presence of a CNOT gate is still conditioned by the operator, however the SWAP gates are necessary for any operator. The skeleton has CNOT depth $3(2n-3)$.

Figure 8.1: Skeleton circuits on 5 qubits with a linear depth.

a CNOT in the circuit or not. An example with 5 qubits taken from his paper is given in Fig 8.1a. Maslov showed that the skeleton can be set to have depth $2n-3$. Then, with the concatenation of 3 skeleton circuits, he managed to synthesize any operator in depth $6n-9$.

Then he extended it to the LNN architecture. The general idea is to move the qubits in the hardware in such a way that the skeleton circuit can still be executed. This gives a new skeleton circuit made of necessary SWAP gates and optional CNOT gates — the CNOT gates in the original skeleton circuit. Fig 8.1b gives the new skeleton circuit from Fig 8.1a adapted to the LNN architecture. A SWAP gate needs 3 CNOTs to be implemented but combined with a CNOT gate it only needs 2 CNOTs. In any case, the worst-case depth for the new skeleton is 3 times the depth of the original skeleton circuit, i.e., $18n + \mathcal{O}(1)$.

Around the same time, Kutin *et al.* gave several constructions of specific linear reversible operations for the LNN architecture: addition, swap, permutation, generic linear reversible operator [117]. They focused on the shallowest way to do it. For a generic linear reversible operator, they relied on their construction for reversing the qubits, i.e., the image of an n-qubit state $|x_1x_2...x_n\rangle$ is $|x_nx_{n-1}...x_1\rangle$. This construction is a *sorting network* and contains only SWAP gates. The network, as a SWAP circuit, is of depth n. An example with 7 qubits is given Fig 8.2. Then they considered the same sorting network but with *boxes* replacing the SWAP gates. Each box, acting on 2 qubits,

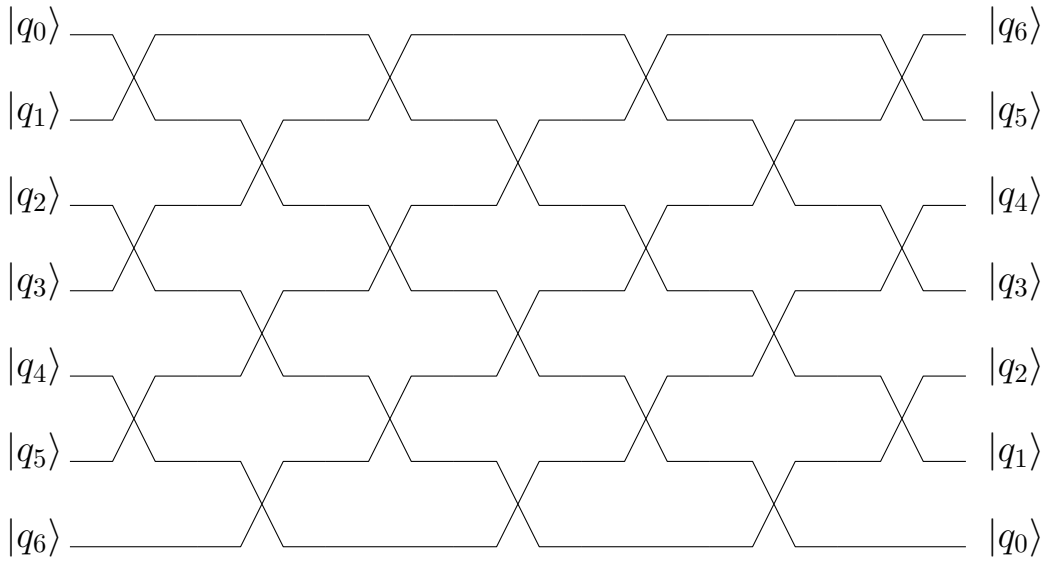


Figure 8.2: Sorting network for 7 qubits. As a SWAP circuit, the depth of the circuit is n. Replacing each CNOT by a box gives a skeleton circuit for the synthesis of triangular linear reversible operators.

can perform one of the following operations:

- $(u, v) \to (u, v)$, requiring 0 CNOT,
- $(u, v) \to (u, u \oplus v)$, requiring 1 CNOT,
- $(u, v) \to (u \oplus v, v)$, requiring 1 CNOT,
- $(u, v) \to (v, u \oplus v)$, requiring 2 CNOT,
- $(u, v) \to (u \oplus v, u)$, requiring 2 CNOT,
- $(u, v) \to (v, u)$, requiring 3 CNOT.

Kutin *et al.* [117] proved that a sorting network made of boxes can transform any operator into a northwest triangular one. Moreover, for each box, only the state of one of the two output qubits needs to be fixed after applying the box. This means that we can always choose a box that needs at most 2 CNOTs to be implemented. Consequently, the total depth of the sorting network is $2n$. Finally, they showed how to synthesize a northwest triangular operator with a similar sorting network except that in this case for each box the states of the two output qubits need to be fixed. Therefore we may need at most 3 CNOTs for some boxes (if we only need to swap the qubits) and the depth of this second part is upper bounded by $3n$. Overall this gives a generic method for synthesizing any linear reversible operator for the LNN architecture in depth at most $5n$. To our knowledge, this is the best result in the literature for any given graph connectivity. This result can only be improved by a constant factor as Kutin *et al.* also showed that some operators need at least circuits with depth $2n$ to be implemented. So the best possible method for synthesis for the LNN architecture should provide circuits of depth comprised between $2n$ and $5n$. For other architectures, the bounds are not clear. Obviously, if an architecture contains a Hamiltonian path in it then one can apply the algorithm for the LNN case, giving an upper bound of $5n$ for the depth. To our knowledge, lower bounds are not known but Maslov computed lower bounds for 2

simplified models in the case where each qubit has k neighbors [130]. The first model is the case where we have to execute every gate given by the Gaussian elimination algorithm in a given order; the second model is less restrictive as we have to execute every gate but we assume that they all commute. In both cases, the depth is lower bounded linearly in n.

Depth Optimization Algorithms for Full Qubit Connectivity

Although it was not done in their paper, the algorithm proposed by Kutin *et al.* [117] can be extended to the full connectivity case: this is what we show in this paragraph. To our knowledge, such an extension has never been proposed in the literature.

In the original Kutin *et al.*'s algorithm [117], each box corresponds to an interaction between a pair of qubits and it can be decomposed into two parts: first, we execute the interaction strictly speaking between the two qubits with a CNOT gate, secondly, we move the qubits in the hardware by swapping them. If we consider that the connectivity is full then we do not need to move the qubits anymore in the hardware. This means that we can replace each box by a CNOT gate and we get rid of the SWAP gates. We end with a new skeleton circuit that is functionally equivalent to the one given by Kutin *et al.* except that each box is now a single CNOT. The skeleton circuit from Kutin *et al.*, as a box-based circuit, is of depth $2n$. Therefore our now skeleton, as a CNOT-based circuit, is also of depth $2n$. To our knowledge, this was the best result until the asymptotically optimal algorithm proposed recently in [90]. The pseudo-code of this new algorithm is given Algorithm 8.3. For simplicity we only show the case for a lower triangular operator, the generalization to any operator is done via an LU decomposition [66]:

$$A = PLU$$

where A is the operator to synthesize, P is a permutation matrix, and L, resp. U, are lower, resp. upper, triangular operators. With full qubit connectivity, a permutation can be implemented with a circuit of constant depth 6 [141]. Each triangular operator can be synthesized with a circuit of depth n, leading to a total depth of $2n + 6$ for the synthesis of an arbitrary operator. Given that we do not move the qubits anymore, most of the algorithm consists in tracking what would be the positions of the qubits in the hardware to determine which interactions need to be done at a given time step. Then it is easy to decide if, for a given pair of qubits (i, j), a CNOT gate needs to be added. If the operator is lower triangular, we only have to decide if we add the CNOT $(i \to j), i < j$ as we must not ruin the triangular structure by adding a CNOT $(j \to i)$. Then if the i-th component of the j-th row is 1 then add a CNOT $(i \to j)$. The reason why it works is not straightforward: we have to note that when we decide to apply or not a CNOT $(i \to j)$, either the components $k < i$ have already been treated for qubit i so it cannot modify the components of qubit j, or such components have not been treated on both qubits, so modifying them on qubit j is not a problem as they will be zeroed later in the algorithm.

So far we have presented state-of-the-art algorithms that can only synthesize circuits with linear depth. But given that:

- any operator can be synthesized with a circuit of size $\mathcal{O}(n^2/\log_2(n))$,
- it is possible to apply $\mathcal{O}(n)$ CNOT gates in parallel,

we must ask: can we synthesize any linear reversible operator with circuits of depth $\mathcal{O}(n/\log_2(n))$? For some restricted connectivities — LNN for instance — we know that the answer is no but what about the unconstrained case? This question was answered in the already mentioned article [90] and the answer is yes. The authors propose an algorithm based on the LU decomposition and a

Algorithm 8.3: Adaptation of Kutin *et al.*'s algorithm [117] on a triangular operator L for a full qubit connectivity.

Require: $n \geq 0, \quad L \in \mathbb{F}_2^{n \times n}$ triangular
Ensure: C is a CNOT-circuit implementing L with depth at most n

```
C ← [ ]
perm ← [[1, n]]
for j = 1 to n do
  if j ≡ 1[2] then
    start ← 1
  else
    start ← 2
  end if
  while start < n do
    if L[perm[start+1], perm[start]] = 1 then
      C.append(CNOT(perm[start], perm[start+1]))
    end if
    perm[start], perm[start+1] ← perm[start+1], perm[start]
    start ← start + 2
  end while
end for
return reverse(C)
```

divide-and-conquer approach. The proof of the optimal depth complexity is quite hard to summarize but the principle of the algorithm is simpler so we give a brief description of it. First, the algorithm starts with an LU decomposition $A = PLU$. Again P can be synthesized in constant depth 6, therefore we only need to treat the triangular case. We illustrate with the lower triangular case. The synthesis of L consists in a divide-and-conquer algorithm, the operator L is decomposed as

$$L = \begin{pmatrix} L_{\lfloor n/2 \rfloor} & \\ A & L_{\lceil n/2 \rceil} \end{pmatrix}$$

where $L_{\lfloor n/2 \rfloor}$ and $L_{\lceil n/2 \rceil}$ are triangular operators of size $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ respectively and A is any boolean matrix of size $\lfloor n/2 \rfloor \times \lceil n/2 \rceil$. The algorithm initially synthesizes in parallel both triangular suboperators by applying recursively the algorithm. Then we are left with the operator

$$L' = \begin{pmatrix} I & \\ A' & I \end{pmatrix}$$

to synthesize. This is done by considering the following blocks in L:

- the northwest identity operator is seen as a block diagonal operator with $n/\log_2(n)$ blocks of size $\log_2(n)/2$, noted $B_1, ..., B_{n/\log_2(n)}$,
- A is divided into $\log_2(n)/2$ blocks of $n/\log_2(n)$ rows, noted $A_1, ..., A_{\log_2(n)/2}$. For simplicity we consider each A_i as a matrix $C_i \in (F_2^{\log_2(n)/2})^{\frac{n}{\log_2(n)} \times \frac{n}{\log_2(n)}}$, i.e., we see A_i as a $\frac{n}{\log_2(n)} \times \frac{n}{\log_2(n)}$ matrix with elements from $F_2^{\log_2(n)/2}$.

The specific structure of L is summarized in Fig 8.3. The synthesis of L consists in successive applications of two stages of row operations:

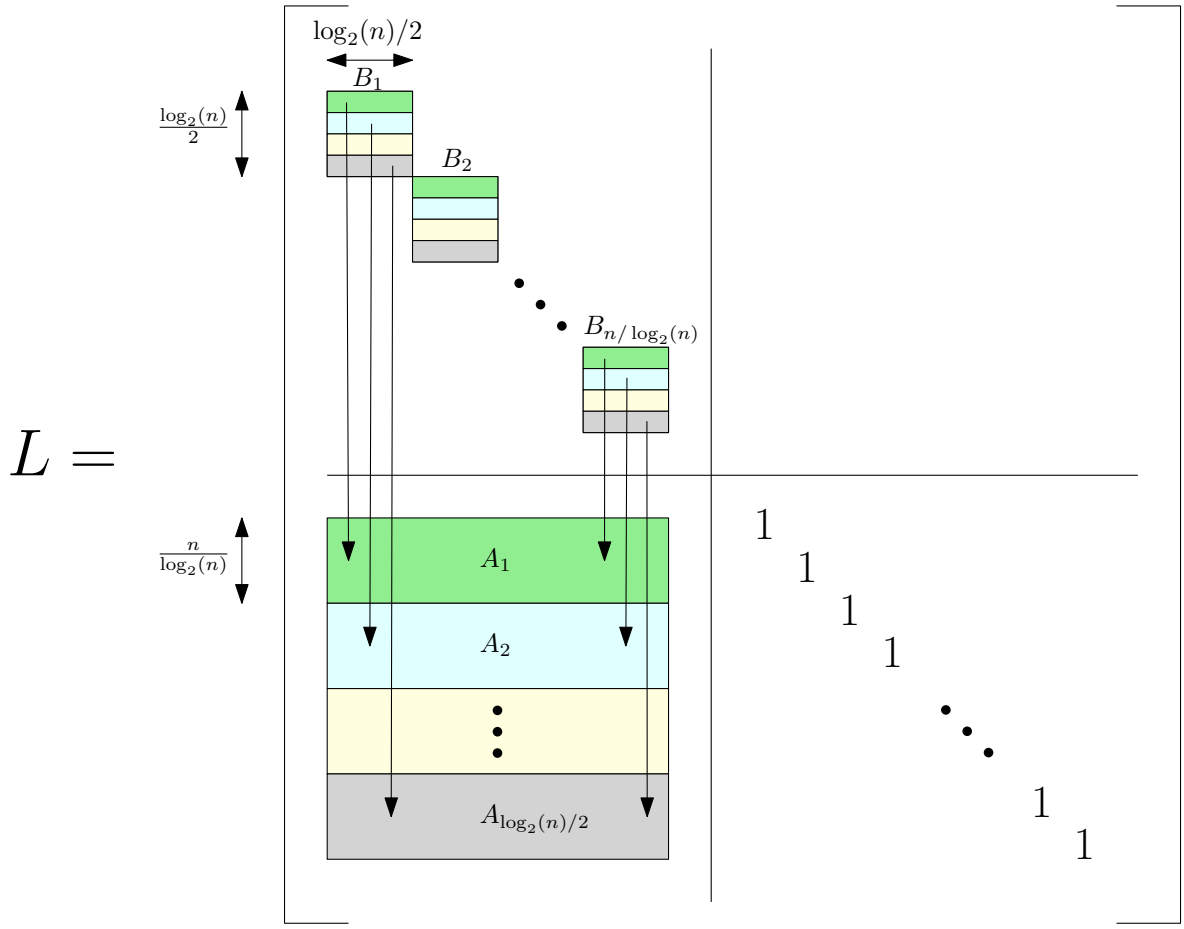


Figure 8.3: Block structure of the triangular operator L for Jiang *et al.*'s algorithm.

1. row operations on the B_i's such that specific words of $\log_2(n)/2$ bits appear on each row,
2. row operations between the B_i's and the A_i's to zero words of $\log_2(n)/2$ bits of A.

More precisely, the k-th row of a B_i has to zero the i-th column of C_k. For that a sequence of operators on $\log_2(n)/2$ qubits is computed such that the property "Every word on $\log_2(n)/2$ bits appears on each row of the B_i's" is verified. Such sequence of operators is called a row traversal sequence. The operators of the row traversal sequence are computed in parallel on each B_i via the row operations during Stage 1. Once we have the desired operator on each B_i, we need to compute the appropriate row operations of Stage 2. Each row of a B_i act on a specific C_k so the corresponding row operations can be done in parallel. So all we need is to see how to coordinate the row operations acting on the same C_k. For simplicity consider the case $k = 1$, i.e., all the first rows of each B_i are used to zero C_1. Given that all B_i's are identical, the choice of applying a row operation from block B_i to the k-th row of C_1 is: is $C[k, i]$ equal to $B_*[1, :]$ (where the index on B has been omitted to emphasize that all B_i's are the same) ? Therefore the matrix C_1 can be seen as the adjacency matrix P of a bipartite graph G where $P[k, i] = 1$ if $C_1[k, i] = B_*[1, :]$. A sequence of parallel row operations between the B_i's and C_1 corresponds to a matching in G and a "good" sequence of parallel row operations is given by a matching decomposition of G. A central theorem that we will also use in our own work is the following: if the maximum number of 1 in a row or a column of P is p then there exists a decomposition of G into p matchings, i.e., a sequence of p parallel row operations is necessary to zero all the entries of C equal to $B_*[1, :]$. Given that

each word appears on each row of the B_i's we are ensured that A will be zero at the end of the algorithm. Finally we need to assume that A is sufficiently random, if it is not the case one can decompose $A = A' \oplus A''$ with A', A'' sufficiently random and do the process two times, the first time for adding A' and the second time for adding A''. A high level pseudo-code of their algorithm is given Algorithm 8.4.

The depth $d(n)$ for the synthesis of one triangular operator is therefore given by

$$d(n) = d(n/2) + 2\times \text{ length row traversal sequence} \times \left(\underbrace{d(\log_2(n))}_{\text{synthesize the operator } B_i} + \text{size matching decomposition} \right)$$

Finally the authors have shown that the length of the row traversal sequence is $\mathcal{O}(\sqrt{n})$ and if A is sufficiently random at each iteration the matching decomposition is of size $\mathcal{O}(\sqrt{n}/\log_2(n))$. Therefore

$$d(n) = d(n/2) + \mathcal{O}(\sqrt{n}) \times (d(\log_2(n)) + \mathcal{O}(\sqrt{n}/\log_2(n))) = \mathcal{O}(n/\log_2(n))$$

hence the result. A better estimation of the total depth is given in Chapter 10.

8.3 Contributions

In the first three following chapters we propose different algorithms for the synthesis of CNOT circuits. Each chapter focuses on a specific metric and connectivity:

1. Chapter 9 focuses on the size optimization of CNOT circuits with full qubit connectivity.
2. Chapter 10 is dedicated to the depth optimization of CNOT circuits with full qubit connectivity.
3. Chapter 11 deals with the size optimization of CNOT circuits for hardware with constrained connectivity.

Chapter 12 is dedicated to the benchmarks. The method we propose in Chapter 11 gives also good results in the full qubit connectivity case, we need to compare it with the methods of Chapter 9. Besides the types of benchmarks are very similar in Chapters 9,10 and 11. For these reasons, we believe it appears clearer to show all the benchmarks in one shot in a dedicated chapter. Finally, Chapter 13 concludes Part C with a summary of our works and proposes tracks for future research.

8.3.1 Contributions of Chapter 9

The contributions of Chapter 9 are as follows:

- We propose GreedyGE, an optimized version of the Gaussian elimination algorithm for synthesizing any linear reversible operator. When we zero out the subdiagonal entries of the matrix, like in a standard Gaussian elimination, we perform the operations that minimize an ad-hoc cost function, resulting in fewer operations for the synthesis. We study the theoretical worst-case behavior of our algorithm. Overall, our algorithm is asymptotically optimal with an improvement over [152, Algo. 1] . In terms of computational time, our algorithm is faster than the standard Gaussian elimination algorithm.

Algorithm 8.4: Jiang *et al.*'s algorithm for CNOT circuit synthesis.

Require: $n \geq 0, \quad L \in \mathbb{F}_2^{n \times n}$ lower triangular
Ensure: C is a CNOT-circuit implementing L^{-1}
Function *Synthesis(L, n)*
$C \leftarrow [\,]$
$k_1 \leftarrow \lceil n/2 \rceil$
$k_2 \leftarrow \lfloor n/2 \rfloor$
$L_1 \leftarrow L[1 : k_1, 1 : k_1]$
$L_2 \leftarrow L[k_1 + 1 : n, k_1 + 1 : n]$
C_1 = Synthesis(L_1, k_1)
C_2 = Synthesis(L_2, k_2)
C.append(C_1)
C.append(C_2)
applyCircuit(L, C)

$A \leftarrow L[k_1 + 1 : n, 1 : k_1]$
$A', A'' \leftarrow$ ComputeRandomSum(A)
$m \leftarrow \log_2(n)/2$
$nb \leftarrow n/\log_2(n)$
$R \leftarrow$ RowTraversalSequence(m)
R.pushfirst(I_m)

for M = {A', A"} **do**
  for $i = 2$ to length(R) **do**
    $Circ \leftarrow$ StandardSynthesis($R_i R_{i-1}^{-1}$)
    for $j = 1$ to nb **do**
      C.append(shift($Circ$, $(j-1)m$))
    end for
    for $j = 1$ to m **do**
      $G \leftarrow$ GenerateGraph($M[(j-1) \times m + 1 : j \times m, :]$, $R_i[j, :]$)
      $\mathcal{M} \leftarrow$ MatchingDecomposition(G)
      $Circ \leftarrow$ ExtractRowOperations($\mathcal{M}$)
      C.append($Circ$)
    end for
  end for
end for
return C

- We propose a new way to compute the LU decomposition of a linear reversible operator. This provides us triangular operators L and U whose synthesis yield shorter circuits, especially when the circuit is expected to be "small enough". We will detail this notion of "small enough" in Chapter 12.
- We show that cost minimization techniques are useful for finding circuits whose expected size is "small enough". We briefly review the existing algorithm and we propose our own cost function that increases the range of validity of this kind of technique.

8.3.2 Contributions of Chapter 10

The contributions of Chapter 10 are as follows:

- We present a generalization of the ideas developed in [90]. This gives DaCSynth: a simple divide-and-conquer framework that divides the synthesis into parallelizable sub-problems that can be solved with several strategies.
- We give strict upper bounds on the depth of the circuits. First, we prove that, in all generality, the depth is upper bounded by $2n + 2\lceil \log_2(n) \rceil$. Then we present a specific strategy that gives the upper bound $\text{depth}(n) < \frac{8}{5}n + 8\lceil \log_2(n) \rceil$, this is an improvement over the best algorithm in the literature for medium sized registers. In practice, we propose a greedy algorithm to solve our sub-problem.
- We extend our framework to the case where parities on ancillary qubits need to be synthesized. We show that the depth increases logarithmically in the number of ancillae.

8.3.3 Contributions of Chapter 11

The contributions of Chapter 11 are as follows:

- We present a new method for the synthesis of CNOT circuits relying on solving a well-known cryptographic problem: the syndrome decoding problem [19]. Our algorithm transforms the synthesis problem into a series of syndrome decoding problems.
- We propose several methods to solve the syndrome decoding problem. The first one is an exact method relying on integer programming. Then to reach larger problem sizes we propose approximate methods using well-known greedy methods for pathfinding problems.
- This method, initially designed for full qubit connectivity, is robust enough to be extended to hardware having a Hamiltonian path in their connectivity graph.

The results of Chapter 11 have been presented at RC 2020 and they are published by Springer LNCS [48].

8.3.4 Contributions of Chapter 12

In Chapter 12:

- We provide benchmarks of our methods optimizing the size with full qubit connectivity and compare them to state-of-the-art algorithms. First, we test our algorithms on random circuits of various sizes. Overall, we can synthesize circuits with more than 200 qubits in a few seconds while giving circuits that are 25% smaller on average than the current state of the art and up to 50% for some specific operators.

Metric	Size		Depth	
Connectivity	Full	Arbitrary	Full	Arbitrary
Algorithm	Patel-Markov-Hayes	Kissinger et al. Mosca et al.	1. Jiang et al. 2. Kutin et al.	Kutin et al.
Complexity	$O\left(n^2/\log_2(n)\right)$	$O(n^2)$	1. $O\left(n/\log_2(n)\right)$ 2. $2n$	$5n$
Idea	Block Gaussian elimination	Steiner tree + GE	1. Divide-and-conquer + LU 2. Sorting network	Sorting network
Our contributions	Greedy Gaussian elimination (GreedyGE)	Syndrome decoding	Greedy Divide-and-conquer (DaCSynth)	x
Complexity	$O\left(n^2/\log_2(n)\right)$	Unknown	$\approx 0.85n$	x
Practical savings	25%	$> 15\%$	50%	x
Chapter	9	11	10	x

Table 8.1: State of the art and summary of contributions for CNOT circuits synthesis.

- We also provide benchmarks for our syndrome decoding based method for specific NISQ architectures. Except for small architectures where the variance of our results is very large or the LNN architecture where our algorithm performs poorly, we consistently outperform the state-of-the-art method with gains between 10% and 20% on average.
- We give benchmarks of our divide-and-conquer framework to support our theoretical results and compare them to state-of-the-art algorithms. Our best method provides circuits of depth smaller than n where n is the number of qubits. This improves the state-of-the-art algorithms by a factor of 2.
- We include our methods into a more general Clifford+T quantum compiler: the Tpar algorithm proposed in [10] and we show how our algorithms perform. On a set of reversible functions from a standard library of functions, we successfully reduce the total number of CNOTs and the total circuit depth while keeping other metrics of interest (T-count, T-depth) as low as possible.

A summary of the state-of-the-art algorithms and our contributions is given in Table 8.1.

Chapter 9

Size Optimization on an Unconstrained Architecture

Contents

This chapter focuses on the size optimization of CNOT circuits for an unconstrained architecture, i.e., with full qubit connectivity. We improve on two results in the literature: first, we improve [152, Algo. 1] which remains, to our knowledge, the best algorithm for size optimization in unconstrained architecture. We provide both theoretical and practical improvements. Namely, the upper bound of our algorithm proposal is slightly better and we give insights into why we are strongly confident that our algorithm always performs better than [152, Algo. 1] . The benchmarks in Chapter 12 are consistent with these theoretical improvements. Secondly, we improve the work in [169] where a cost-minimization approach was studied. Notably, we propose a new cost function that performs better than the cost function considered in [169] in some specific cases that we will highlight in Chapter 12.

The outline of this chapter is the following: in Section 9.1 we first present GreedyGE, a greedy version of the Gaussian elimination algorithm for the synthesis of triangular matrices. Then in Section 9.2 we extend GreedyGE to the synthesis of any operator in two ways: we either triangularize the operator with a slight modification of GreedyGE and then we apply GreedyGE, or we compute an LU decomposition of our operator and apply GreedyGE twice. Finally, we investigate the use of pure greedy algorithms in Section 9.3.

9.1 GreedyGE: a Greedy Gaussian Elimination Algorithm for Triangular Boolean Matrices

9.1.1 General Presentation of the Algorithm

Given a lower triangular operator L — the upper case can be treated similarly — the Gaussian elimination algorithm can be summarized as follows: by applying elementary row operations $\text{Row}(i, j)$ with $i < j$ we zero the sub-diagonal elements of L without changing its triangular shape. This process is performed column by column, starting from the first one. This way one can easily see that if the k first columns are treated, applying row operations $\text{Row}(i, j)$ with $k < i < j$ will not change the treated columns as all their elements are 0. Since L is invertible, when zeroing the k-th column the k-th row is necessarily equal to $L[k, :] = e_k^T$ hence one can always add the row k to any row $j > k$ to zero the entry $L[j, k]$. This guarantees the good behavior of the algorithm for any input matrix.

Usually, a Gaussian elimination algorithm always performs row operations $\text{Row}(k, j)$ when zeroing the k-th column. As one row operation ensures to zero only one element at a time, we need at most $\frac{n(n-1)}{2}$ row operations to zero every sub-diagonal elements of L, leading to a worst-case complexity of $O(n^2)$ for the synthesis of a generic linear reversible operator.

A first straightforward improvement is to authorize any row operation $\text{Row}(i, j)$, $i < j$. This was exploited in [152, Algo. 1] : by partitioning the matrix of size n into blocks of size m, the authors use the fact that there can only be at most 2^m different vectors in each block. So for $n > 2^m$ it is possible to zero $n - 2^m$ full vectors of size m by applying one row operation for each vector. Using this method some row operations can zero up to m entries at a time, diminishing the total number of row operations. By choosing $m = \lfloor \alpha \log_2(n) \rfloor$ (for arbitrary $0 < \alpha < 1$) they reach an asymptotic complexity of $n^2/(\alpha \log_2(n))$ in the worst case which meets the theoretical lower bound.

Our proposed algorithm can be seen as a direct improvement of [152, Algo. 1] . It comes from the following simple observation: given a row i of our triangular operator and assuming that we follow a standard Gaussian elimination process, an upper bound on the number of row operations that will be applied to row i during the synthesis is given by

$$\#\text{CNOTs} = i - \min\{j \mid L[i, j] = 1\}. \tag{9.1}$$

In other words, when zeroing the entries of a row, we have to focus first on the entries on the first columns and this can be done by minimizing the cost function given by Eq. (9.1). Once an entry is zeroed, we are sure that it is “treated” and will never be modified. Note on the other hand how zeroing entries in the last columns do not give us the guarantee that they would not be modified again during the synthesis. This simple observation is at the core of the Gaussian elimination algorithm: entries are acted upon one by one. It is also at the core of [152, Algo. 1] which improves on regular Gaussian elimination: entries are acted upon by fixed-size blocks. In this paper, we improve on [152, Algo. 1] by allowing for varying-size blocks. In other words, we do not fix a block size but at each iteration we let the greedy component of our algorithm choose the largest suitable block size.

Therefore we propose the following 3-step method:

1. choose among the rows with entries "1" in the left-most columns the two rows i and j that have the largest number of common left-non-zero-elements,

2. apply the row operation $\text{Row}(\min(i, j), \max(i, j))$,

3. repeat the first two steps until L is the identity.

The pseudo-code of the algorithm is given in Algorithm 9.1. The time complexity of the algorithm mostly depends on step 1, i.e., the function SelectRowOperation. Actually, this step can be easily implemented iteratively on the columns :

- We start with the set of all the rows and, during the kth step, we have in memory a set of rows having the same first k-1 entries.
- Then we separate those rows into two sets: those which have the k-th entry respectively equal to 0 and 1, corresponding respectively to the sets set0 and set1 in Algorithm 9.1. The rows in either set0 or set1 have the same first k entries.
- We want to zero the maximum of 1 in priority so if set1 contains 2 or more rows we continue with this set, otherwise we choose set0 and go to step k+1.

By this process, the size of our set of lines always remains greater than 2 until we have only 2 lines: these are the two lines on which to do our row operation. Clearly, the algorithm is polynomial in the size of the matrix.

Algorithm 9.1: GreedyGE : Greedy Gaussian Elimination of a triangular operator L.

Require: $n \geq 0, \quad L \in \mathbb{F}_2^{n \times n}$ lower triangular
Ensure: C is a CNOT-circuit implementing L
Function *Synthesis(L, n)*
  $C \leftarrow [\,]$
  while $L \neq I_n$ **do**
    set $\leftarrow$ SelectRowOperation(L, n)
    $i_0 \leftarrow \min(\text{set}[0], \text{set}[1])$
    $j_0 \leftarrow \max(\text{set}[0], \text{set}[1])$
    C.append(CNOT(i_0, j_0))
    $L[j_0, :] \leftarrow L[i_0, :] \oplus L[j_0, :]$
  end while
  return reverse(C)
Function *SelectRowOperation(L, n)*
  set $\leftarrow [\![1, n]\!]$
  $j \leftarrow 0$
  while $|\text{set}| > 2$ **do**
    $\text{set}_0 \leftarrow \{i \in \text{set} \mid L[i, j] = 0\}$
    $\text{set}_1 \leftarrow \{i \in \text{set} \mid L[i, j] = 1\}$
    if $|\text{set}_1| < 2$ **then**
      set $\leftarrow \text{set}_0$
    else
      set $\leftarrow \text{set}_1$
    end if
    $j \leftarrow j + 1$
  end while
  return set

Our algorithm belongs to the same family as the Gaussian elimination method and [152, Algo. 1] because we synthesize the operator column by column. However, this is an improvement over [152,

Algo. 1] because we do not work with a fixed block size. At each step, we somewhat choose the largest block size possible to zero the largest number of elements in one row operation. We are then ensured that these entries will not be modified anymore. Thus this is also an improvement over cost minimization methods [169] that can take as cost function the number of ones in the matrix for instance. With such greedy methods, it may be possible that in one step more elements are zeroed but it is not guaranteed that theses elements will be left untouched thereafter. We keep a structure in the synthesis which enforces the convergence of our greedy algorithm.

9.1.2 Improving the Time Complexity

When executing Algorithm 9.1 we find at each iteration the next row operation by selecting the rows according to their number of common first elements. As a first approach, we keep the set of the most promising rows until the size of the set is equal to 2. With a little more work we can in fact order all the rows in one run and get the whole set of row operations required to zero the sub-diagonal elements of the current column. It is more efficient than redoing the sorting work after each new row operation because the selection algorithm will repeat the same calculations again and again but without the presence of the modified rows. The new pseudo-code for a more efficient version of our method is given Algorithm 9.2.

The new version of SelectRowOperation, the function SelectAllRowOperations, is based on the idea that row operations should always be done, if possible, between rows belonging to the same sets. We implement this idea recursively on the columns: first, we only select the rows with a "1" entry in the first column. Then, after the creation of set0 and set1, we call recursively SelectAllRowOperations on both sets: we have two sets of row operations that treat separately the rows with first entries "10" and "11". After the execution of SelectAllRowOperations, both set0 and set1 contains the unmodified rows. In both cases, there can be at most one row unmodified. If both sets have one row unmodified then we add another row operation $\text{Row}(\min(\text{set}_0[0], \text{set}_1[0]), \max(\text{set}_0[0], \text{set}_1[0]))$ and the set returned contains $\min(\text{set}_0[0], \text{set}_1[0])$. If only one set has an unmodified row then the set returned contains that row. We illustrate the behavior of our algorithm with the example given in Fig 9.1.

9.1.3 Bounding the CNOT Count

We denote with $C(n)$ the number of CNOTs required to synthesize a triangular operator L of size n with Algorithm 9.1. Given $1 \leq k \leq n$, we consider $c(n,k)$ the number of row operations required to synthesize the first k columns.

Once the first k columns are synthesized the $n \times k$ block matrix $L[:, 1:k]$ is equal to the matrix $\begin{pmatrix} I_k \\ 0 \end{pmatrix}$ and we are left with the sub-matrix $L[k+1:end, k+1:end]$ to synthesize. Then we have

$$\forall k \geq 1,\ C(n) \leq c(n,k) + C(n-k)$$

and the total number of row operations to synthesize L is upper bounded by

$$C(n) \quad \leq \quad \sum_{i=1}^{m} c\left(n - \sum_{j=1}^{i-1} k_j,\ k_i\right) \tag{9.2}$$

where $k_1, k_2, ..., k_m$ can represent any partition of L into m sub-blocks, i.e., m positive integers such that $\sum_{i=1}^{m} k_i = n$. By considering the worst-case scenario for each block we do not take into consideration the side effects of the row operations that were applied on the first blocks and

Algorithm 9.2: FastGreedyGE : Greedy Gaussian Elimination of a triangular operator L.

Require: $n \geq 0, \quad L \in \mathbb{F}_2^{n \times n}$ lower triangular
Ensure: C is a CNOT-circuit implementing L

Function *OptimizedSynthesis(L, n)*
- $C \leftarrow [\,]$
- **for** j = 1 to n-1 **do**
  - set $\leftarrow \{i \in [j, ..., n] \mid L[i, j] = 1\}$
  - pairs, set = SelectAllRowOperations(L, j+1, set)
  - **for** pair = pairs **do**
    - $i_0 \leftarrow \min(\text{pair}[0], \text{pair}[1])$
    - $j_0 \leftarrow \max(\text{pair}[0], \text{pair}[1])$
    - C.append(CNOT(i_0, j_0))
    - $L[j_0, :] \leftarrow L[i_0, :] \oplus L[j_0, :]$
  - **end for**
- **end for**
- **return** reverse(C)

Function *SelectAllRowOperations(L, j, set)*
- **if** |set| < 2 **then**
  - **return** $\emptyset$, set
- **end if**
- $\text{set}_0 \leftarrow \{i \in \text{set} \mid L[i, j] = 0\}$
- $\text{set}_1 \leftarrow \{i \in \text{set} \mid L[i, j] = 1\}$
- $\text{pairs}_0, \text{set}_0 \leftarrow \text{SelectAllRowOperations}(L, j + 1, \text{set}_0)$
- $\text{pairs}_1, \text{set}_1 \leftarrow \text{SelectAllRowOperations}(L, j + 1, \text{set}_1)$
- $\text{pairs} \leftarrow [\text{pairs}_0, \text{pairs}_1]$
- **if** $|\text{set}_0| > 0$ **and** $|\text{set}_1| > 0$ **then**
  - $i_0 \leftarrow \text{set}_0[0]$
  - $i_1 \leftarrow \text{set}_1[0]$
  - $\text{pairs} \leftarrow [\text{pairs}, [\min(i_0, i_1), \max(i_0, i_1)]]$
  - $\text{set} \leftarrow \min(i_0, i_1)$
- **else**
  - **if** $|\text{set}_0| > 0$ **then**
    - $\text{set} \leftarrow \text{set}_0[0]$
  - **else**
    - $\text{set} \leftarrow \text{set}_1[0]$
  - **end if**
- **end if**
- **return** pairs, set

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

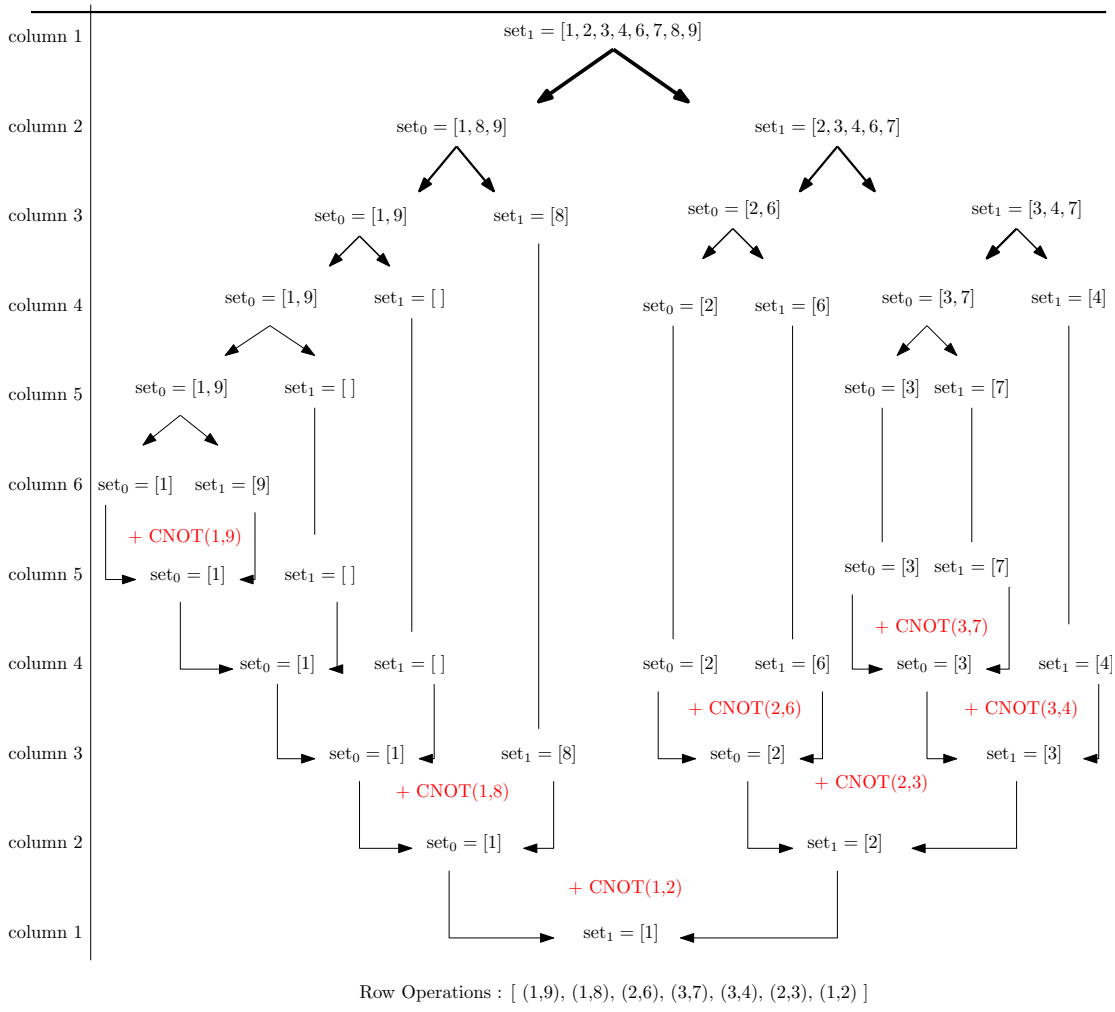


Figure 9.1: Illustration of the SelectAllRowOperations function on a specific example.

that could have an impact on the remaining matrix. Especially when looking strictly at one block, one can easily see that some row operations will only zero one element at a time in a block — the rightmost elements — but will certainly have an impact on the next block: by not considering this effect, we weaken our upper bound but the proof of the inequality (9.2) is straightforward.

We now turn to an estimation of an upper bound of $c(n,k)$. Considering a block of size (n,k) and assuming that $k < \log_2(n)$, the algorithm process is the following :

- as there can be at most 2^k different bitstrings of k bits, $n-2^k$ row operations at most will be used to zero the duplicate rows.
- We are left with a rectangular block of size $(2^k,k)$. Using a similar argument we can zero $k-1$ bits of 2^{k-1} different rows by using one row operation for each, then the remaining bit on these rows can be zeroed with one row operation. We are now left with a block of size $(2^{k-1},k)$, we zero $k-2$ bits of 2^{k-2} rows with one row operation each and the two remaining bits on these rows can be zeroed with two row operations. Repeating this process until the whole block has been reduced to $\begin{pmatrix} I \\ 0 \end{pmatrix}$, we perform at most

$$2^{k-1}\times(1+1)+2^{k-2}\times(1+2)+2^{k-3}\times(1+3)\ldots = 2^{k+1}\times\sum_{j=2}^{k+1} j/2^j \leq 3\times 2^k$$

  row operations. We can save some row operations because the right upper triangular part of the block is already zeroed (as L is triangular) but we neglect them for simplicity.

So we have the estimation

$$c(n,k) \leq n + 2^{k+1}. \tag{9.3}$$

From this we can derive an upper bound for $C(n)$. We replace Eq. (9.3) in Eq. (9.2) :

$$C(n) \leq \sum_{i=1}^{m}\left(n - \sum_{j=1}^{i-1} k_j + 2^{k_i+1}\right).$$

As $n-\sum_{j=1}^{i-1}k_j = \sum_{j=i}^{m}k_j$ for any i we rewrite the upper bound

$$C(n) \leq \sum_{i=1}^{m}\left(2^{k_i+1} + \sum_{j=i}^{m} k_j\right)$$

$$C(n) \leq \sum_{i=1}^{m}\left(2^{k_i+1} + ik_i\right). \tag{9.4}$$

For simplicity we assume that $k_1 = k_2 = \ldots = k_{m-1} = k$. We also assume that $k_m = n - \sum_{i=1}^{m-1} k_i = k$ and $m = n/k$ because it does not affect the results. Replacing in Eq. (9.4):

$$C(n) \leq m2^{k+1} + k\frac{m(m+1)}{2} = \frac{n^2}{2k} + \frac{n}{k}2^{k+1} + \frac{n}{2} \tag{9.5}$$

We now prove two things:

1. for one block we show that our upper bound is better that the one given in [152].

2. we prove that the asymptotic complexity is at most $n^2/\log_2(n)$.

1. Taking $k = \alpha \log_2(n)$ with $0 < \alpha < 1$, we have

$$2 \times C(n) < \frac{n^2}{\alpha \log_2(n)} + \frac{4n^{1+\alpha}}{\alpha \log_2(n)} + n. \tag{9.6}$$

where the multiplication by 2 gives an upper bound for the synthesis of a generic operator. This is a direct improvement over the upper bound in [152] given by

$$\text{total row op. [152]} \leq \frac{n^2}{\alpha \log_2(n)} + 2n^{1+\alpha} + \text{ negligible terms} \tag{9.7}$$

provided $\frac{4}{\alpha \log_2(n)} < 2$ which is the case if n is sufficiently large. For instance, with $\alpha = 1/2$ then $n > 16$ is sufficient.

2. The coefficient of the leading term in Eq (9.6) is $1/\alpha$, yet taking the limit $\alpha \to 1$ in Eq. (9.6) leads to a complexity of $5n^2/\log_2(n)$. Instead, taking $k = \log_2(n) - p$ for some p, we have

$$2 \times C(n) < \frac{n^2}{\log_2(n)} \times \frac{1}{1 - p/\log_2(n)} \times \left(1 + \frac{4}{2^p}\right) + n.$$

By choosing carefully p, for instance $p = \sqrt{\log_2(n)}$, we get

$$2 \times C(n) < a(n) \times \frac{n^2}{\log_2(n)} + n$$

with $a(n) = \frac{1}{1-1/\sqrt{\log_2(n)}} \times \left(1 + \frac{4}{2^{\sqrt{\log_2(n)}}}\right)$. As $a(n) \to 1$ this proves that the asymptotic complexity is $n^2/\log_2(n)$.

We improve the upper bound Eq. (9.7) in two ways: we ensure that the coefficient of the leading term is 1 and we avoid other terms that distort the real complexity. Especially if one wants to have the best asymptotic complexity with [152, Algo. 1] by choosing α arbitrarily close to 1, the term $n^{1+\alpha}$ becomes arbitrarily close to n^2 and its impact is not negligible until very large values of n. Even with intermediate values of n, for example $n = 1000$, then $a(n) \approx 2.12$. To have the same leading term we have to choose $\alpha = 1/2.12$. With these values we now have

$$\frac{2n^{1+\alpha} + n^2/(\alpha \log_2(n))}{n^2/(\alpha \log_2(n))} \approx 1.25$$

which means that the term $2n^{1+\alpha}$ increases the upper bound by 25%.

Discussion. The main result we want to emphasize is that the upper bound in Eq. (9.4) is true for any partitioning of the matrix L. This means it performs better than any partitioning method unless a more efficient method for synthesizing one block is found. Notably, given $k_1 = k_2 = ... = k_{m-1} = k$ the size of the block used for [152, Algo. 1] , it is clear that the row operations done in [152, Algo. 1] to zero an entire k-bits row will also be performed by our algorithm. Then we are left with a sub-matrix to synthesize for which [152, Algo. 1] will use a Gaussian Elimination method whereas our algorithm will perform recursively by first zeroing the bitstrings of size $k-1$. In the worst case our algorithm will only zero one element at a time like the Gaussian elimination algorithm.

This gives insights about the advantage of GreedyGE over [152, Algo. 1] . When [152, Algo. 1] performs a standard Gaussian elimination process to zero some elements, we somewhat perform [152, Algo. 1] on a smaller sub-matrix. Thus, at these steps of the algorithm, we expect to have the same gain in the number of row operations than [152, Algo. 1] has compared to the classical Gaussian elimination algorithm. Consequently, although we cannot theoretically claim that our algorithm always generates shorter circuits than [152, Algo. 1] — as we cannot claim that [152, Algo. 1] always performs better than standard Gaussian elimination — we have strong confidence that our algorithm provides better results most of the time — for the same reasons that [152, Algo. 1] gives most of the time better results than the Gaussian elimination algorithm. This assertion is empirically verified in our benchmarks, see Chapter 12.

9.2 Extending GreedyGE to General Operators

So far we have presented an algorithm for the synthesis of triangular operators. To extend this to generic linear reversible operators A we give two strategies :

- we can modify our algorithm for triangular operators such that it transforms A into an upper triangular operator U on which we apply our unmodified algorithm,
- we can rely on an LU decomposition to factorize A into the product of two triangular operators. Then the concatenation of the circuits implementing U and L yields a circuit implementing A.

9.2.1 Modification of FastGreedyGE

Algorithm 9.2 works even if the input operator is not lower triangular. In the outer for-loop, during the j-th iteration the algorithm will zero the sub-diagonal elements of the j-th column provided that the element $A[j, j]$ is equal to 1. So all we need to do is adding an extra row operation to make sure that $A[j, j] = 1$ is satisfied. If $A[j, j] = 1$ then we do nothing otherwise we choose among the rows $[\![j + 1, n]\!]$ whose j-th element is 1 the one that zeroes the largest number of right elements of the j-th row so that when executing FastGreedyGE on the future U operator we save some row operations. Overall this increases the CNOT count by at most n so it is negligible in the derivation of the upper bound above.

9.2.2 Optimizing the Choice of L and U

An LU decomposition provides a way to decompose an operator into two simpler sub-operators. Ideally, the path from the identity to the target operator given by the concatenation of the paths of two sub-operators would be close to the optimal path. In our case, it is very unlikely that an LU decomposition would give such an ideal decomposition because the structure of triangular operators is too specific. Nonetheless, the matrices L and U in the LU decomposition are not unique and we can optimize the factorization by choosing appropriate L and U operators such that we minimize the overall complexity of the final reversible circuit. In this section, we report several strategies to choose the matrices L and U.

If we allow row and column permutations, we can compute the LU decomposition of an operator A following this algorithm:

1. Choose a row vector $v = A[i, :]$ and a column j for which $A[i, j] = 1$.

2. Set the first line of U to v and set the first column of L to $A[:, j]$.

3. Update the matrix A, first by left-applying the gate SWAP$(i, 1)$ and right-applying the gate SWAP$(j, 1)$. Then add the vector v to every row $A[k, :]$ where $A[k, 1] = 1$. Finally we are left with a matrix A with zeroed first column and first row.

4. Repeat the algorithm on this updated A until we obtain the null matrix.

At the end of the algorithm we have the relation

$$P_1 A P_2 = LU$$

where P_1, P_2 are permutation matrices. P_1 is the concatenation of the SWAP gates that were left-applied to A and P_2 is the concatenation of the SWAP gates that were right-applied to A. After the synthesis of L and U we can commute the gates with P_2 and have

$$A = P_1^{-1} P_2^{-1} L' U'$$

and we transfer the permutation matrix $P_1^{-1} P_2^{-1}$ to the end of the total circuit. L' and U' are triangular operators up to a permutation of their rows and columns.

At each step of the algorithm, we have several choices for the row and column vectors. We aim to construct L and U such that their synthesis will give short circuits. To do so we tried different strategies :

- First, if we believe that sparse L and U matrices can be implemented with shorter circuits, a first approach consists in choosing the row and column with smallest Hamming weights: this is the “sparse” approach.

- Secondly, we can adopt a "minimizing cost" approach. If the updated matrix A' minimizes a cost function then the synthesis of A' will require a shorter circuit. We choose the cost function to be the number of ones and we choose among all the possible rows and columns the pair that minimizes the number of ones in A'.

These decompositions transform a general operator A into two operators L, U with specific structures. Triangular operators are easier to synthesize but this also limits the optimality of the solution. Because we force the algorithm to synthesize two matrices with a specific structure, it is very unlikely that we can perform an optimized synthesis by going through this forced path. This also gives insights into the question about the global optimality of our algorithm: even if we use an optimal algorithm for synthesizing a triangular operator it is very unlikely that we can get an optimal solution for the whole synthesis. Our algorithm gives asymptotic optimal results in the worst case but will be suboptimal for specific operators. Notably, if the operators can indeed be synthesized with a small circuit, direct greedy methods have shown encouraging results. In the next section, we will review the methods used and propose an extension of this framework to improve the scalability of this kind of algorithms.

9.3 Pathfinding Based Algorithms

The problem of linear reversible circuit synthesis can fit into a more generic problem called "pathfinding". In a pathfinding problem, we are generally given a starting state, a target state, and a set of operators that dictates the reachable states from a specific state. Then the goal is to find

a path from the starting state to the goal state and the shorter the path is, the better it is. The reformulation of linear reversible circuit synthesis into a pathfinding problem is straightforward: the starting state is the operator A we want to implement, the goal state is the identity operator and the available operators are all the elementary row and column operations.

This field of research in AI has been very active for the last decades and we can rely on many algorithms to solve our problem. The most famous one is probably the A* algorithm and all its derivatives (IDA* [113], Weighted A* [115], Anytime A* [122], etc.). These algorithms have been successfully used to solve toy problems (Rubik's cube, Tile puzzles, Hanoi towers, Top Spin puzzle) but also concrete problems in video games [4], robotics [4,122], genomics [76,83,211,214], planning [27,76], etc. Yet we are quickly faced with some crucial limitations :

- First the size of the search space grows much faster in our case than in any of the cases mentioned above. To give an order of magnitude, in [206] they compare greedy search algorithms - a subclass of pathfinding algorithms known for providing suboptimal solutions very quickly - and they highlight time and memory limitations of some algorithms especially on the largest problem they ran on: the 48-puzzle. The 48-puzzle and its variations (8-puzzle, 15-puzzle) are sliding puzzles. In the case of the 48-puzzle, 48 tiles are arranged in a 7×7 square with one tile missing. The goal of the puzzle is to order the tiles by moving the empty space around. For this specific problem the authors of [206] show that the computational time for most of the pathfinding algorithms they consider increases substantially while providing solutions of poor quality. The number of different positions in the 48-puzzle is equal to $49!/2 \approx 3 \times 10^{62}$. This is less than the number of linear reversible functions on 15 qubits. As we expect to be able to synthesize circuits on 50 qubits and even more, it is clear that we cannot use all the pathfinding algorithms in the literature.

- Secondly we need a function that will evaluate the distance between the current state and the target state. The properties of this function, usually called the heuristic, will deeply influence the performance of the pathfinding algorithms. Its goal is to guide the search by giving priority to more promising states and prune a whole part of the search space. More generally there are two sources of improvement for a pathfinding algorithm: the accuracy of the heuristic function and the management of the path discovery, i.e., what strategy do you employ to explore the search space. If the latter can be quite independent of the problem, the former is very problem-specific and unfortunately to our knowledge no good heuristics exist that satisfy good theoretical properties. For instance, for A* to provide an optimal solution we need the heuristic to be admissible and consistent, i.e., it must always underestimate the length of the optimal path and the value of the heuristic should always decrease of at most one after the application of one elementary operation. By taking a function that always returns 0 we have an admissible and consistent heuristic but the search will result in a brute-force search. Generally to obtain an admissible heuristic we can consider a simpler version of our problem and the optimal solution of this problem gives an underestimation of the distance to the target state. In the case of linear reversible circuits, the closest simplified subproblem we found is the Shortest Linear Straight-Line Program [28] but this problem is also NP-hard and cannot be approximated exactly by a polynomial algorithm.

For all these reasons, we believe that using the standard A* algorithm or its derivatives is a dead end. Instead of that, we focused our work on greedy best-first search algorithms that can be seen as a cost minimization/discrete gradient descent approach. It has been shown that the heuristics that are efficient for this kind of search are different than the ones for A* algorithms [205], so we expect that we can exploit the properties of the matrix A of our operator to guide our search.

To our knowledge, the cost minimization approach for linear reversible circuits synthesis has only been investigated in [169]. Yet, with their methods, the authors can only synthesize circuits on at most 25 qubits. More crucially, they did not investigate the behavior of their methods when the resulting circuits are supposed to be small. According to us, this is a behavior that has to be considered because as we will see the results of the methods drastically change depending on the size of the optimal circuits.

We consider two different heuristic functions to guide our search. The first one, already used in [169], is the function

$$h_{sum}(A) = \sum_{i,j} a_{i,j}$$

that is to say the number of ones in the matrix A. We also propose a new cost function to solve this problem, it is defined by

$$h_{\text{prod}}(A) = \sum_{i=1}^{n} \log\left(\sum_{j=1}^{n} a_{i,j}\right)$$

which corresponds to the log of the product of the number of ones on each row. An important improvement given in [169] is to add the value of the cost function applied to the inverse as well, i.e.,

$$H_{sum}(A) = h_{sum}(A) + h_{sum}(A^{-1}),$$

$$H_{\text{prod}}(A) = h_{\text{prod}}(A) + h_{\text{prod}}(A^{-1}).$$

As we will see this tightens the possible paths and can improve the results because we escape more easily from local minima during the search.

We now give a few details on our implementation. We directly work on a cost matrix that stores the impact of each possible CNOT on the cost function. Namely let M be such matrix, then $M[i,j] = h(E_{ij}A) - h(A)$ where $h \in \{h_{sum}, h_{\text{prod}}\}$. We can define similar cost matrices for A^{-1} and for column operations. Overall we have four matrices $M_{A,\text{row}}, M_{A,\text{col}}, M_{A^{-1},\text{row}}, M_{A^{-1},\text{col}}$. Note that $(E_{ij}A)^{-1} = A^{-1}E_{ij}$ so a row operation $\text{Row}(i,j)$ on A is a column operation $\text{Col}(j,i)$ on A^{-1}. The cost minimization algorithm is simple if we work with such matrices: at each iteration we look for the pair (i,j) that minimizes $\min(M_{A,\text{row}}[i,j] + M_{A^{-1},\text{col}}[j,i], M_{A,\text{col}}[i,j] + M_{A^{-1},\text{row}}[j,i])$, we update A and A^{-1} and we pursue the algorithm until A is a permutation matrix. The advantage of working on cost matrices rather than directly on A is that the update of the M's is cheaper than updating A and computing the costs. For instance, suppose we apply $\text{Row}(i,j)$ to A, then if $h = h_{\text{sum}}$ the impact on the cost function of any row operation that does not involve row j will be the same than before the application of $\text{Row}(i,j)$. This means that we only need to update $M_{A,\text{Row}}[i,j], M_{A,\text{Row}}[j,i], M_{A^{-1},\text{Col}}[i,j], M_{A^{-1},\text{Col}}[j,i]$ for $i = 1..n$. The updates of $M_{A,\text{Col}}$ and $M_{A^{-1},\text{Row}}$ are also simpler because only one element of each column of A has been modified. However, for our new cost function the updates are not that simple and it will have an impact on the computational time.

With cost minimization techniques it is not rare to fall into local minima. In that case we simply select among the best operations a random one and pursue the search. If the number of iterations exceeds a certain number then we consider that the algorithm is stuck in a local minimum and we stop it.

9.4 The Benchmarks in a Nutshell

We provide benchmarks of GreedyGE and purely greedy methods in Chapter 12, Section 12.1. In a few words, GreedyGE is fast and produces the shortest circuits for operators that represent the worst cases. Purely greedy methods give the best results when the number of qubits is smaller than 30 or when the operators are expected to be synthesized with a small circuit. Otherwise, their scalability is limited: their performance deteriorates rapidly with great variance in the results whether with the number of qubits or the "complexity" of the operator to synthesize.

In other words, GreedyGE is the algorithm of choice for synthesizing operators in a worst-case scenario or when the number of qubits is large. Purely greedy methods perform well when the number of qubits is small or when the operators are "simple enough" to be synthesized with a "short circuit". These notions of "simple enough" and "short circuit" are hard to quantify, but some experiments are done in Chapter 12 to give numerical insights about these notions.

Chapter 10

Depth Optimization on an Unconstrained Architecture

Contents

After studying how to optimize the size of CNOT circuits we are now interested in optimizing their depth, still for an architecture with full qubit connectivity. First, we propose an algorithm without ancillary qubits, then we extend our algorithm in the case where extra parities have to be synthesized on ancillary qubits. Our work builds on the paper [90]. It contains insightful theoretical works on how to use divide-and-conquer methods to optimize the depth of CNOT circuits both with and without ancillary qubits. Notably, in the non-ancillary case, the authors show that an n-qubit CNOT circuit can be parallelized to depth $\mathcal{O}(n/\log_2(n))$. However, their result is probably not useful for practical use, at least until very large problem sizes. An upper bound of the depth is given approximately by

$$\text{depth [90]} \leq \alpha \frac{n}{\log_2(n)} + \beta\sqrt{n}\log_2(n).$$

Let us now compute an estimation for α, β. As a divide-and-conquer framework, the authors have derived a recursive formula in the case of triangular operators. Noting $d(n)$ for the depth we have

$$d(n) \leq d(n/2) + 2 \times \mathcal{O}(\sqrt{n}) \times (\mathcal{O}(\log_2(n)) + \mathcal{O}(\sqrt{n}/\log_2(n))).$$

Note that we think there is a typo in their formula, the term $\mathcal{O}(\log_2(n))$ being $\mathcal{O}(\log^3{}_2(n))$ in their paper. This term corresponds to the synthesis of an operator of size $\log_2(n)/2$. Assuming that we use the best algorithm, i.e., the adaptation of Kutin *et al.*'s algorithm [117] we proposed in Section 8.2.2, each of these operators can be synthesized with a circuit of depth at most $2 \times \log_2(n)/2 = \log_2(n)$. We may have missed something but this improves the real complexity so we keep our proposed modification. The second term $\mathcal{O}(\sqrt{n}/\log_2(n)))$ corresponds to the matching decomposition of a graph and the authors showed that the leading coefficient is $\sqrt{e}$. The third term, $\mathcal{O}(\sqrt{n})$, corresponds to the length of the row-traversal sequence on k qubits that gives a sequence

of k-qubit operators such that for any bitstring of size k and any integer $j \in [\![1, k]\!]$, there is an operator in the sequence whose j-th row equals the bitstring. The authors proved that there exists a row-traversal sequence on k qubits of length $3 \times 2^{k-1} - k + 1$. Here we have $k = \log_2(n)/2$ and $\mathcal{O}(\sqrt{n}) = 3/2 \times \sqrt{n}$. Therefore, we finally get

$$d(n) \leq d(n/2) + 3 \times \sqrt{n} \times \left(\log_2(n) + \sqrt{ne}/\log_2(n)\right) = d(n/2) + 3\sqrt{n}\log_2(n) + 3\frac{n\sqrt{e}}{\log_2(n)}$$

and

$$d(n) \leq 3 \times \left(\sum_{j=0}^{\log_2(n)-1} \sqrt{\frac{n}{2^j}} \log_2\left(\frac{n}{2^j}\right) + \frac{\sqrt{e}\frac{n}{2^j}}{\log_2\left(\frac{n}{2^j}\right)} \right).$$

After simplification we have

$$d(n) \leq 3 \times \left(2\sqrt{e}\frac{n}{\log_2(n)} + 3.3\sqrt{n}\log_2(n) \right)$$

and we have to do it for the two triangular operators given by the LU decomposition. Overall

$$\text{depth [90]} \leq 20 \left(\frac{n}{\log_2(n)} + \sqrt{n}\log_2(n) \right).$$

Although it is only an upper bound, in practice there is little simplification one can make when synthesizing a specific operator: the row-traversal sequence still needs to be synthesized and the matching decomposition is done on random graphs so we cannot expect the exact complexity to be that lower compared to the upper bound. To give an idea, it is cheaper to use our modification of Kutin *et al.*'s algorithm [117] — producing circuits of depth at most $2n$, see Section 8.2.2 — until $n = 170000$. In their article [90], the authors acknowledge that their method is not effective for practical register sizes. For practical use, they propose a simpler algorithm that produces circuits of depth at most $4n$. Thus the adaptation of Kutin *et al.*'s algorithm is still, to our knowledge, the best algorithm when the number of qubits ranges from 1 to several thousands.

Nevertheless, we think the use of a divide-and-conquer method is a promising idea and we propose a practical implementation of a divide-and-conquer framework. This framework transforms the synthesis problem into successive zeroing of binary matrices with given elementary operations. We show that the work in [90] fits in our framework as a special case. With a specific strategy for zeroing the matrices we also improve the upper bound to $\frac{8}{5}n + 8\log_2(n)$, making it the best upper bound for $150 < n < 480000$, see Table 10.1 for more details. In practice, we use a greedy method for zeroing the matrices and the benchmarks show that the depth of the produced circuits lies between $0.85n$ and n. We also extend our framework when additional parities need to be synthesized on ancillary qubits. We show that this can be done with a logarithmic overhead in the number of extra parities.

The outline of the chapter is as follows: first, we present DaCSynth — the divide-and-conquer framework used — and we derive upper bounds on the depth in Section 10.1. Then we present the practical greedy method we use for executing the zeroing process in Section 10.2. Finally, in Section 10.3 we extend our framework to take into account extra parities.

10.1 DaCSynth: a Divide-and-Conquer Framework

Given an operator $A \in F_2^{n \times n}$ to synthesize, our proposed algorithm is a divide-and-conquer algorithm and consists in the following Steps :

Method	Gaussian elimination	[117]	[90]	Our algorithm
Upper bound	$4n$	$2n$	$\mathcal{O}\left(\frac{n}{\log_2(n)}\right)$	$\frac{8}{5}n + 8\log_2(n)$
Best result for	-	$n < 150$	$n > 480000$	$150 < n < 480000$

Table 10.1: Synthesis algorithms and theoretical upper bounds with the approximate ranges of validity for each method.

1. First compute a permutation matrix P such that $PA = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix}$ and $A_1 \in F_2^{\lceil n/2 \rceil \times \lceil n/2 \rceil}$ is invertible,
2. Apply row operations on A to zero the block A_3 such that the resulting matrix is $A' = \begin{pmatrix} A'_1 & A'_2 \\ 0 & A'_4 \end{pmatrix}$,
3. Apply row operations on A' to zero the block A'_2 such that the resulting matrix is $A'' = \begin{pmatrix} A''_1 & 0 \\ 0 & A''_4 \end{pmatrix}$,
4. Call recursively the algorithm on A''_1 and A''_4. When $n = 1$ return an empty set of row operations.

Step 1 is straightforward: consider the rows of the sub-matrix $A[:, 1 : \lceil n/2 \rceil]$ (using Matlab notation). Start from an empty set and at each Step add a row to the set. If the rank of the set is increased, keep the row, otherwise remove it. If the resulting set is not of rank $\lceil n/2 \rceil$, this would mean that the first $\lceil n/2 \rceil$ columns of A are not linearly independent which is impossible by invertibility of A. In addition, we assume that the qubits are fully connected so we can avoid to apply P by doing a post processing on 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. Hence the core of the algorithm lies in Steps 2 and 3. We now give the details for processing Step 2. This can be easily transposed to do Step 3 as well.

Theorem 10.1.1. *Given $A = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \in F_2^{n \times n}$ with $A_1 \in F_2^{\lceil n/2 \rceil \times \lceil n/2 \rceil}$ invertible, zeroing A_3 by applying row operations on A is equivalent to zeroing the matrix $B = A_3 A_1^{-1}$ by applying any row and column operations on B or flipping any entry of B.*

Proof. First, by assumption, A_1 is invertible so the matrix B exists.

- Applying an elementary row operation $\text{Row}(i, j)$ on A_3 gives the matrix $E_{ij}A_3$ and B is updated by $E_{ij}B$. Thus a row operation on A_3 is equivalent to a row operation on B.
- Applying an elementary row operation $\text{Row}(i, j)$ on A_1 gives the matrix $E_{ij}A_1$ and B is updated by BE_{ij}. Thus a row operation on A_1 is equivalent to a column operation on B.
- B is a $\lfloor n/2 \rfloor \times \lceil n/2 \rceil$ matrix. The k-th row of B gives the decomposition of the k-th row of A_3 in the basis given by the rows of A_1. Thus any row operation $\text{Row}(k_1, \lceil n/2 \rceil + k_2)$ will flip the entry (k_2, k_1) of B.

With these three types of operations available on B, the invertibility of A_1 is preserved. Thus when B is zero necessarily A_3 is also zero. □

Obviously flipping all the 1-entries of B is enough to reduce A_3 to the null matrix, but we are concerned with the shallowest way of doing this. In the following we show how to compute the optimal depth of the circuit zeroing B using only the flipping operation.

Theorem 10.1.2. *With the same notations, let k be the maximum number of 1 entries in one row or one column of B. Then if we use only the flipping operation we need a circuit of depth k to zero B.*

Proof. We exploit a theoretical result about bipartite graph already used in [90]. Consider the bipartite graph $G = (V_1, V_2, E)$ where each vertex of V_1 is a row of A_1, each vertex of V_2 is a row of A_3 and B is the adjacency matrix of G. Any matching in G represents a series of row operations that can be executed in parallel and that will zero some entries in B. If there is at most k non-zero entries in each row and column of B this means that the degree of G is k as well. Any bipartite graph of degree d can be decomposed into exactly d matchings [96]. Hence a circuit of depth k is needed to transform B into the null matrix. □

We are now able to give a strict upper bound on the worst-case result of our algorithm.

Corollary 10.1.2.1. *The depth of the circuits given by our algorithm is upper bounded by $2n + 2\lceil \log_2(n) \rceil$ with n the number of qubits.*

Proof. A first straightforward formula for the depth of the circuit output by our algorithm is

$$d(n) = d(\lceil n/2 \rceil) + 2 \times d^*(\lceil n/2 \rceil)$$

where d^* is the depth of the circuits computing Steps 2 and 3. Using the result of the previous theorem we have

$$d^*(n) \leq n$$

So overall the depth of our circuit is upper bounded by

$$d(n) \leq d(\lceil n/2 \rceil) + 2\lceil n/2 \rceil$$

As $d(1) = 0$ and by exploiting the fact that $\lceil \lceil n/2 \rceil / 2 \rceil = \lceil n/4 \rceil$ we have

$$d(n) \leq 2 \times \left(\sum_{k=1}^{\lceil \log_2(n) \rceil} \lceil n/2^k \rceil \right) \leq 2 \times \left(\sum_{k=1}^{\lceil \log_2(n) \rceil} n/2^k + 1 \right)$$

After calculation we have

$$d(n) \leq 2n + 2\lceil \log_2(n) \rceil$$

□

A Block algorithm for Steps 2 and 3

In order to improve the upper bound of our framework, we propose a block method for Steps 2 and 3. Given an $n \times n$ matrix B to zero and an integer $k < n$ such that $n = bk + r$, we divide B into a matrix of $\lceil \frac{n}{k} \rceil \times \lceil \frac{n}{k} \rceil$ blocks:

- $\lfloor \frac{n}{k} \rfloor^2$ such blocks are of size k,

- $\lfloor \frac{n}{k} \rfloor$ such blocks are of size $k \times r$,
- $\lfloor \frac{n}{k} \rfloor$ such blocks are of size $r \times k$
- and the lower right one is of size $r \times r$.

If B is of size $n \times (n+1)$ or $(n+1) \times n$ (which can happen if A is of odd size) then some blocks on the edge will be of size $k \times (r+1)$ or $(r+1) \times k$. In any case, as $r < k$, $r+1 \leq k$ and the critical point is that all of these rectangular blocks are smaller than the $k \times k$ blocks.

Now we consider each nonzero block as a 1-entry in a $\lceil \frac{n}{k} \rceil \times \lceil \frac{n}{k} \rceil$ binary matrix that can be mapped to a bipartite graph G as above. Then it is clear that a matching in G corresponds to a subset of blocks on which we can apply row and column operations in parallel.

Considering one such matching, we assume that we can reduce the maximum number of 1-entries in each row and column of one block to an integer p in depth at most D. Then all the blocks are matrices with at most p nonzero entries per row and column and we can flip all of these nonzero entries in p sequences of row operations as they belong to different rows and columns in B. After that all the blocks given by the matching are zero and we can repeat the process with another matching without modifying the nullified blocks. G can be decomposed into at most $\lceil \frac{n}{k} \rceil$ matchings, each of them requires a depth of at most $D+p$ to zero all the blocks so the total depth for Step 2 (or Step 3) is $(D+p) \times \lceil \frac{n}{k} \rceil$. Again using the formula $\lceil \frac{\lceil n/m \rceil}{k} \rceil = \lceil \frac{n}{mk} \rceil$ an upper bound for the total depth is given by

$$d(n) \leq 2(D+p) \times \left(\sum_{j=1}^{\lceil \log_2(n) \rceil} \lceil n/(k2^j) \rceil \right).$$

After calculation we get

$$d(n) \leq \frac{2(D+p)}{k} n + 2(D+p)\lceil \log_2(n) \rceil. \tag{10.1}$$

We notice that with $k = 1$ then $D = 0, p = 1$ and we recover the result of Theorem 10.1.2. We can now prove our main result.

Corollary 10.1.2.2. *The depth of the circuits given by our algorithm is upper bounded by* $\frac{8}{5}n + 8\lceil \log_2(n) \rceil$ *with* n *the number of qubits.*

Proof. To improve our first result we need to find an efficient synthesis for bigger problem sizes. We performed a brute-force search for square matrices of size $k = 1, 2, 3, 4, 5, 6$. The search consisted in a breadth-first search: starting from the partial permutations, i.e., binary matrices with at most one entry equal to 1 per row and column, row/columns operations were applied in a growing depth manner. We explored the set of binary matrices and computed the minimum depth required to reduce them to a partial permutation. To reduce the size of the search space we only considered matrices up to row and column permutations. For this purpose, we used standard techniques involving graph isomorphisms to compute a canonical representative for each class [63]. The results are given in Table 10.2. We recall that row and column operations can be performed in parallel. It is clear that for smaller rectangular matrices the worst-case depth cannot be larger. So by considering blocks of size 5 the depth in the worst case is $D = 3$ and $p = 1$. Replacing in Eq. (10.1) gives the result. □

Depth \ Number of qubits	1	2	3	4	5	6
0	2	3	4	5	6	7
1		4	17	58	160	447
2			15	237	3960	105039
3				17	1498	144575
4						1531
5						11

Table 10.2: Number of binary matrices reachable for different number of qubits and circuit depth (up to row/column permutations).

We want to insist on the fact that the current upper bound is to be improved. In fact, any improvements in the synthesis of "small" quantum circuits can significantly improve the theoretical upper bound and its range validity given in Table 10.1 on page 145. For instance, computing the worst-case depth for $k = 7, 8, 9$, if possible, may lead to a better upper bound. Especially, what happens if we stop the row and column operations once the maximum number of 1-entries in each row and column is below an integer $p > 1$? If D decreases faster than p increases this would represent another improvement.

Besides, although the upper bound requires to use the worst case scenario for zeroing the blocks, we can expect a better behavior in average. With blocks of size $k = 6$ the chances that among $\lceil n/k \rceil$ blocks there is one block requiring a depth of 4 or 5 is given by $p = 1 - \left(\frac{250068}{251610}\right)^{\lceil n/6 \rceil}$ so with probability $q = \left(\frac{250068}{251610}\right)^{\lceil n/6 \rceil} \approx 0.9939^{\lceil n/6 \rceil}$ we have $D = 3$. On average, for computing Step 2 or 3, we can expect that the depth will be $D = (3q + 5p) = (5 - 2q)$. For example, with $n = 128$ the calculations give an average depth of $1.42n + 56$ which represents an improvement over the strict worst case.

As we already mentioned, the synthesis algorithm proposed in [90] can in fact be seen as a special case where $A_1 = I$ and their strategy is also a block algorithm with blocks of size $\log_2(n)/2 \times \log_2(n)/2$ and $n/\log_2(n) \times \log_2(n)/2$. Yet, translated in our framework, they only use the operations on the columns and the flipping entries operations.

10.2 Zeroing Binary Matrices with a Greedy Approach

In practice, we use a greedy algorithm to perform Steps 2 and 3. We recall that we work on a matrix B that we want to zero with the following three available operations: (1) row operations, (2) column operations, (3) flipping one entry. Note that row and column operations can be performed in parallel as this corresponds to CNOT circuits on two disjoint subsets of qubits.

At each Step we compute a sequence of row and column operations on B that minimizes the number of ones in B and that can be done in parallel. If we only consider row or column operations then the optimal sequence can be computed in a polynomial time. To do so we create a directed graph $G_{\text{row}}/G_{\text{col}}$ whose nodes are the rows/columns of B and the edges $(i \to j)$ are weighted by the gain in the number of ones if we apply the row operation $i \to j$. The optimal sequence of row/column operations is given by the maximum weight matching in such graph which can be computed in polynomial time using the blossom algorithm [55].

However, when considering both row and column operations, things are not that simple. A row operation on B modifies G_{col} and a column operation modifies G_{row} so we cannot solve in-

dependently (or one after the other) the two problems in order to have an optimal sequence. The maximum weight matching problem can be reformulated as the linear program

$$\begin{aligned} \text{maximize} \quad & \sum_{e} x_e w(e) \\ \text{such that} \quad & \sum_{e=\{u,v\}} x_e \leq 1 \quad \text{for all vertices } u, v \\ & x_e \in \{0,1\} \quad \text{for all edges } e. \end{aligned} \tag{10.2}$$

Taking into account both row and column operations adds quadratic terms in the cost function and this complicates the search for an optimal solution. Namely, this new problem can be reformulated as

$$\begin{aligned} \text{maximize} \quad & \sum_{e_{\text{row}}} x_{e_{\text{row}}} w(e_{\text{row}}) + \sum_{e_{\text{col}}} x_{e_{\text{col}}} w(e_{\text{col}}) \\ & + \sum_{e_{\text{col}}, e_{\text{row}}} x_{e_{\text{row}}} x_{e_{\text{col}}} \delta(e_{\text{row}}, e_{\text{col}}) \\ \text{such that} \quad & \sum_{e_{\text{row}}=\{u,v\}} x_{e_{\text{row}}} \leq 1 \quad \text{for all vertices } u, v \in G_{\text{row}} \\ & \sum_{e_{\text{col}}=\{u,v\}} x_{e_{\text{col}}} \leq 1 \quad \text{for all vertices } u, v \in G_{\text{col}} \\ & x_e \in \{0,1\} \quad \text{for all edges } e. \end{aligned} \tag{10.3}$$

where the δ's are the quadratic terms. Each quadratic term corresponds to a specific entry in B so we have $\delta(e_{\text{row}}, e_{\text{col}}) \in \{-1, 0, 1\}$ for any row and column. It turns out that solving a generalization of Problem 10.3 is NP-Hard.

Theorem 10.2.1. *With the same notations, finding the sequence of parallel row and column operations that minimizes the number of ones in B is NP-Hard.*

Proof. We reduce the Maximum Clique Problem to problem 10.3. Given a graph $G = (V, E)$, we construct the following two graphs A, B as follows: for any vertex $v \in V$ create two vertices v_{A_1}, v_{A_2} in A and two vertices v_{B_1}, v_{B_2} in B. For both pair of vertices set the weight of the edge at 0. Set the weights of all the other edges to $-\infty$. Now for any pair of vertices $(v, v') \in V \times V$, consider the associated four vertices $v_{A_1}, v_{A_2}, v'_{B_1}, v'_{B_2}$. If $v = v'$ or v, v' are adjacent in V then set the quadratic term to +1, otherwise set the quadratic term to $-\infty$.

A clique of size k in G gives a solution of weight k^2 for problem 10.3. Conversely, given a solution to problem 10.3 whose weight is not $-\infty$, it is always possible to extend this solution to a symmetric solution in A and B, i.e., the matching in A and the matching in B are of equal size. By construction this correspond to the same vertices in V. If the matching in A and B is of size k then the total weight of the solution to problem 10.3 is of size k^2 and this corresponds to a clique of size k in G. □

Given the NP-Hardness of the problem, we compute a sequence of row and column operations greedily. We first choose the best row or column operation that minimizes the number of ones and we keep in memory the operation applied. Then we determine the best row or column operation among the operations that can be performed in parallel with the previously stored operation and we repeat the process. Finally if some rows and columns are left untouched we may complete the sequence of operations by flipping some entries. The best sequence of flipping operations is

computed as described in the proof of Theorem 10.1.2 using the blossom algorithm [55]. If no row or column operation can reduce the number of 1 in B then only the flipping operation is used.

10.3 An Extension to Utilize Extra Memory

The quantum compiler Tpar [10] efficiently reduces the T-depth by computing subsets of T gates that can be applied in parallel. Each T gate is associated to a parity, i.e., a linear combination of the input qubits. Linearly independent parities can be computed at the same time and the T gates are applied in parallel to each qubit carrying one of these parities.

With ancillary qubits, the parallelization can be even more efficient because the ancillary qubits can carry any parity, i.e., it can be a linear combination of the parities carried by non-ancillary qubits. In terms of CNOT circuits synthesis, we need to synthesize a bigger linear reversible operator. Namely, we have to synthesize a boolean matrix of size $p \times n$ where $p-n$ is the number of additional qubits that will carry a parity. We extend our framework to treat this particular case. First, we prove in Section 10.3.1 that the depth for the synthesis of an operator $A \in F_2^{p\times n}, p \geq 2n$ is equal, up to additive logarithmic terms, to the depth of the synthesis on an operator $A \in F_2^{p\times n}, n \leq p \leq 2n$. This result shows that the total depth barely increases with the number of ancillae after a certain threshold. Then, in Section 10.3.2, we extend our framework to deal with an arbitrary number of ancillae to treat the case where $A \in F_2^{p\times n}$ and $n \leq p \leq 2n$.

10.3.1 A Block Extension Algorithm

Given $p = kn + r$, we partition the operator A into k blocks of n rows and one block of r rows. We assume that the first block is of full rank and we merge the first block with the block of r rows. The strategy we propose is divided into three parts:

1. perform some row operations on A such that the $(k-1)$ ancillary blocks are also of full rank.
2. synthesize each block in parallel.
3. zero each $(k-1)$ ancillary block.

Step 1 can be performed with a circuit of depth $\lceil \log_2(k) \rceil$ using the following lemma:

Lemma 10.3.1. *Given two matrices $A, B \in F_2^{n\times n}$ with A of full rank, there exists a partial permutation[1] P such that $B + PA$ is of full rank.*

Proof. Suppose B is of rank $k, k < n$. We can write $B = CD$ where $C \in F_2^{n\times k}, D \in F_2^{k\times n}$ are of rank k. One can always add a set of $n-k$ canonical vectors to the columns of C to create a basis of F_2^n. We can complete as well the rows of D into a basis of F_2^n by adding row vectors of A. We get two new extended matrices C', D' such that $B' = C'D'$ is now invertible. We can always add zero columns in C' and the remaining rows of A in D' and reorder the columns of C' and D' to get $C' = [C|P]$ and $D' = [D|A]$ with P a partial permutation matrix. Rewriting $B' = [C|D] \times [P|A] = CD + PA = B + PA$ proves the result. □

To perform Step 1, we first make sure that the second block is of full rank by adding the appropriate rows of the first block. Using the lemma, this operation can be done with a circuit of depth

[1] We remind that a partial permutation is a boolean matrix with at most one entry equal to 1 per row and column.

1. Then we make sure that the third and fourth block are of full rank by adding the appropriate rows of the first and second blocks, etc. It is clear that we only need $\lceil \log_2(k) \rceil$ iterations to treat all of the blocks.

Step 2 requires a call to our framework for the square blocks and the extension of our framework for the rectangular block. All synthesis are performed simultaneously so the total depth for Step 2 is given by the maximum depth required for the synthesis of one of the blocks.

After Step 2 each block is equal to a permutation matrix. It is clear that we can zero in one iteration half of the blocks, then half of the remaining blocks, etc. Similarly to Step 1 we need a circuit of depth $\lceil \log_2(k) \rceil$ to zero all the blocks and complete the synthesis.

The total depth $d(n, p)$ is given by

$$d(n, p) = 2\log_2(\lfloor p/n \rfloor) + d^*(n, n + r) \leq 4n + 2\log_2(\lfloor p/n \rfloor)$$

where $d^*(n, n+r)$ is the depth required to synthesize the block of size $(n+r) \times n$, which informally represents the maximum depth required when synthesizing all the blocks in parallel. The upper bound is not the tightest possible. The result we want to emphasize is that the depth only depends logarithmically on the number of ancillae.

10.3.2 Extension of the Divide-and-Conquer Framework

We now show how to treat the rectangular block in Step 2 of the block extension algorithm. What follows can be performed for any number of ancillae even though we only need $n \leq p \leq 2n$.

Again, we partition the matrix

$$A = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \\ A_5 & A_6 \\ A_7 & A_8 \end{pmatrix}$$

where A_5, A_6, A_7, A_8 encode the ancillary parities. The algorithm consists in similar operations to the case without ancillae:

1. Find a permutation matrix P such that the updated A_1, A_4 are of full rank.
2. Apply row operations on A to zero the blocks A_3, A_7 such that the resulting matrix is $A' = \begin{pmatrix} A'_1 & A'_2 \\ 0 & A'_4 \\ A'_5 & A'_6 \\ 0 & A'_8 \end{pmatrix}$,
3. Apply row operations on A' to zero the blocks A'_2, A'_6 such that the resulting matrix is $A'' = \begin{pmatrix} A''_1 & 0 \\ 0 & A''_4 \\ A''_5 & 0 \\ 0 & A''_8 \end{pmatrix}$,
4. Call recursively the algorithm on $[A''_1; A''_5]$ and $[A''_4; A''_8]$.

The main difference is that in Steps 2 and 3 we now deal with rectangular matrices. Suppose we are performing Step 2. For clarity we set

$$C_1 = [A_1; A_5],$$

$$C_2 = [A_3; A_7],$$

and

$$n_1 = \lceil n/2 \rceil + \left\lceil \frac{p-n}{2} \right\rceil,$$

$$n_2 = \lfloor n/2 \rfloor + \left\lfloor \frac{p-n}{2} \right\rfloor.$$

With these notations, $C_1 \in F_2^{n_1 \times \lceil n/2 \rceil}$ and $C_2 \in F_2^{n_2 \times \lceil n/2 \rceil}$. We face two problems if we try to simply copy the method without ancillae:

1. First we cannot create the matrix B as before because the matrix C_1 not invertible. Instead we create a matrix $B \in F_2^{n_2 \times n_1}$ where $B[i,:]$ is a solution of

   $$B[i,:]C_1 = C_2[i,:].$$

   In other words

   $$BC_1 = C_2.$$

   We still want to zero B and one can see that the three elementary available operations (row operation, column operation, bit flip) are still valid. If B is zero then C_2 is zero but the converse is not true anymore: we can still apply our greedy algorithm but it may be less optimal because one nonzero row of B might be in the kernel of C_1 and we will add useless row operations to zero it.

   We need a way to explicitly form B. We have many choices for each row of B, we said that any solution to

   $$xC_1 = C_2[i,:] \tag{10.4}$$

   is a valid row for B. In fact Eq 10.4 is an instance of a cryptographic problem called the syndrome decoding problem. Finding a solution with minimal Hamming weight is NP-Hard. In [48] a greedy solution of the problem has been proposed to optimize the size of CNOT circuits but it requires the presence of the canonical vectors to ensure the convergence of the algorithm (see Chapter 11 for more details). This condition can be met if we set

   $$C_1 \leftarrow C_1 A_1^{-1}$$

   $$C_2 \leftarrow C_2 A_1^{-1}$$

   so that the first $\lceil n/2 \rceil$ vectors of C_1 are now canonical vectors. Then we can compute the rows of B by solving each instance of the syndrome decoding problem.

2. Secondly, we are not ensured after Step 2 that $\begin{pmatrix} A'_4 \\ A'_8 \end{pmatrix}$ will be of full rank because $\begin{pmatrix} A'_1 & A'_2 \\ A'_5 & A'_6 \end{pmatrix}$ may be of rank $> \lceil n/2 \rceil$. Ideally, we would like to ensure that $\begin{pmatrix} A_1 & A_2 \\ A_5 & A_6 \end{pmatrix}$ is of rank $\lceil n/2 \rceil$ before Step 2. This would mean that after Step 2 this sub-matrix is still of rank $\lceil n/2 \rceil$ because the only modifications made on it were internal row operations and consequently the desired matrix $\begin{pmatrix} A'_4 \\ A'_8 \end{pmatrix}$ is necessarily of rank $n - \lceil n/2 \rceil$. Unfortunately this is not always possible.

Take for instance the matrix

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix},$$

there is no subset of 3 rows of rank 2. This means that we have to restrict the available operations we perform on A during Step 2. The following modifications do not need to be applied during Step 3 though. First, we ensure when computing the permutation P that A_4 is also of full rank, then we have to prevent any row operation on A that could change the rank of A_4. The easiest way to do so is to zero A_3 as if we were in the case with no ancillae and we zero A_5 with all kinds of operations possible. The consequence of this simple method is that the depth required to synthesize the whole operator can only be greater than the depth required to zero the sub-operator without ancillae.

To be sure that our greedy algorithm will converge, the rows of A_3 have to be written in the basis of the rows of A_1 only because these will be the only rows we will use to zero A_3. In other words if B_1 is a solution of $B_1C_1 = A_7$, then we create $B = \begin{pmatrix} A_3A_1^{-1} \\ B_1 \end{pmatrix}$.

Although each row of B is of size n_1 we have the guarantee that the Hamming weight of each row will not exceed $\lceil n/2 \rceil$. However, we cannot say anything on the Hamming weight of the column of B so we can roughly upper bound the depth of Steps 2 and 3 only by $n_2 \approx p/2$. This means that in theory the depth can increase linearly with the number of ancillae. In practice we will see in Chapter 12 that it is not the case.

10.3.3 The Benchmarks in a Nutshell

We provide benchmarks of DaCSynth and its extensions in Chapter 12, Section 12.2. In a few words, DaCSynth outperforms the state-of-the-art methods [117,130]. The empirical complexity of the depth is approximately n which is a factor or 2 better than the adaptation of Kutin's algorithm [117] (Algorithm 8.3) and a factor of 4 better than the standard Gaussian elimination algorithm. The results with ancillae are consistent with the theoretical results of Section 10.3. For the block version we recover the logarithmic overhead in the number of ancillae shown in Section 10.3.1. For the extension of the divide-and-conquer framework presented in Section 10.3.2 the depth increases linearly with the number of ancillae but with a relatively small slope.

Chapter 11

Syndrome Decoding Based Algorithm for Size Optimization with Hardware Compliant Connectivities

Contents

Chapters 9 and 10 focused on optimizing the circuits when the qubit connectivity is full. This is only a theoretical case since for the moment current architectures allow only limited interactions between qubits. Yet it is still important to work on ideal cases as sometimes a method working for a simplified problem can be transposed to take into account some constraints. This is the case of the method we present in this chapter. It was initially designed for full qubit connectivity and it appears to also work for some connectivity graphs. The algorithm consists in transforming the synthesis of a triangular operator into a series of syndrome decoding problems to solve. The syndrome decoding problem is NP-Hard and inapproximable so we cannot afford to use an exact algorithm unless the problem size is small enough. Yet, although this is a standard problem in cryptography, we were unable to find practical algorithms to compute approximate solutions. We had to come up with ad-hoc greedy solvers and the benchmarks show that it turns out to be efficient. In the unconstrained case, we improve [152, Algo. 1] but also GreedyGE, introduced in Chapter 9 when the number of qubits is below a certain threshold (around 200 qubits). This gives insights about when to use each method. Then we show that the method is robust enough to be naturally extended to a constrained connectivity case. However, the generalization is not done for all possible connected connectivity graphs but only for those that have a Hamiltonian path. This delicate transition from graphs with a Hamiltonian path to all graphs is not inherent in our problem. The method using Steiner trees in [102] also required an extra effort to adapt it to all connected graphs. Nevertheless, many architectures have a Hamiltonian path and the benchmarks show that we outperform the state of the art for those particular architectures. The outline of the chapter is the following: we first present our algorithm in the case of an all-to-all connectivity in Section 11.1. Then we extend

it to the case of restricted connectivity in Section 11.2.

11.1 Syndrome Decoding Based 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 applying 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 $i \to n$ 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 $i \to n$. 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

$$\begin{aligned} &\underset{x\in F_2^m}{\text{minimize}} && wt(x) \\ &\text{such that} && Hx = s^T. \end{aligned} \tag{11.1}$$

Problem 11.1 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 [9], 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 detail in Section 11.1.1.

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.

11.1.1 Syndrome Decoding Problem

In its general form, the syndrome decoding problem is known to be NP-Hard [19] and cannot be approximated by a constant factor [14]. A good overview of how difficult the problem is can be found in [193].

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 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 program

$$\begin{aligned} \min_{x \in F_2^m, t \in \mathbb{N}^n} \quad & c^T \cdot [x; t] \\ \text{such that} \quad & A[x; t] = s. \end{aligned} \tag{11.2}$$

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 also give exact results, but 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 pathfinding algorithms like Real-Time A* [114]. Instead of directly choosing the parity that minimizes the Hamming weight, we look up to

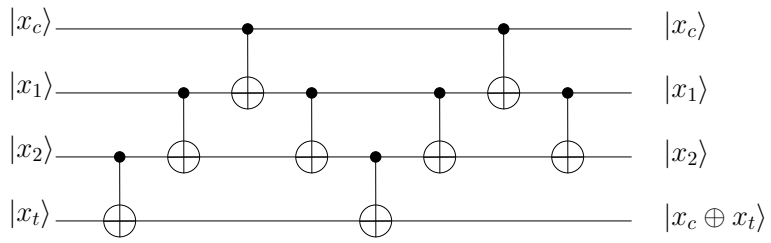


Figure 11.1: CNOT in LNN architecture.

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 a 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 [156], even though the complexity of these algorithms remains exponential.

11.2 Extension to Any Connectivity with an Hamiltonian Path

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

11.2.1 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. A 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 $q_c \rightarrow q_t$ with $\max(1, 4k)$ CNOTs. An example for 4 qubits (with $k = 2$) is given Fig 11.1.

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

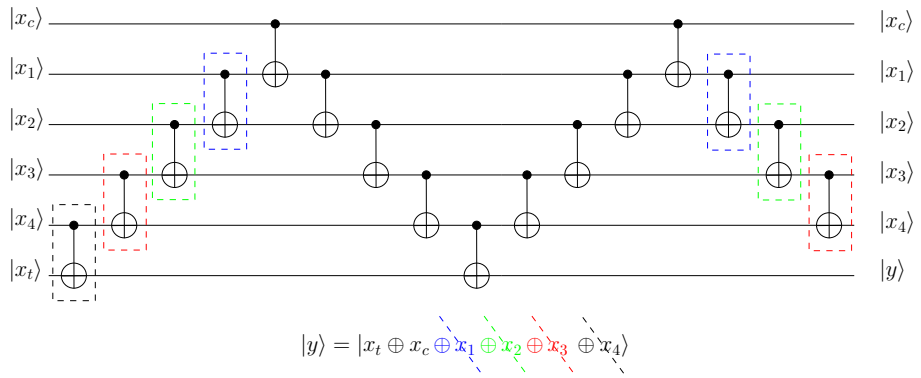


Figure 11.2: Fan-in CNOT in LNN architecture.

of parities in a matrix H and a cost vector $c \in \mathbb{N}^m$, we look for the solution of the optimization problem

$$\begin{aligned} &\underset{x \in F_2^m}{\text{minimize}} && c^T \cdot x \\ &\text{such that} && Hx = s^T. \end{aligned} \tag{11.3}$$

Problem 11.3 can be recast again as an integer linear program: we only have to change the value of c. In what follows, 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 $\text{bc}(s)$ 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

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

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. 11.1 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. 11.1.
- There is not necessarily one unique path in G between the control qubit and the target qubit.

More precisely, the template shown in Fig. 11.1 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 fewer CNOTs than required for applying only q_c. In fact the less costly linear combination of parities is the complete combination $q_c \oplus q_1 \oplus ... \oplus q_k$: $2k + 1$ 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 in Fig. 11.2. 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.

11.2.2 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 (with $P = I$) 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 sub-matrix 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

$$\underset{\pi}{\text{minimize}} \quad \sum_{(u,v)\in E} w_{uv} \, |\pi(u) - \pi(v)| \tag{11.4}$$

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 [154] and the problem is in general NP-Hard [65]. 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 manually chose a suitable ordering of the qubits. Fig. 2.3 features the choices we made for some architectures: the 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.

11.3 The Benchmarks in a Nutshell

We provide benchmarks of the syndrome decoding based algorithms in Chapter 12, Sections 12.1 and 12.3. In the unconstrained case, the better the solver of the syndrome decoding problem the shorter the circuits generated are. All our algorithms always outperform the PMH algorithm [152, Algo. 1] but they only outperform GreedyGE (presented in Section 9.1) when $n < 150$. This is in fact due to the bad scalability of the method: complex solvers do not scale well with the number of qubits and when the solver is too simple we think the quality of the solution of the syndrome decoding problem deteriorates with the problem size.

In the constrained case, except for the LNN architectures, we consistently outperform the state-of-the-art algorithm [102]. Especially, the more connected the architectures, the larger the savings.

Chapter 12

Benchmarks

Contents

This chapter presents the experimental results related to the synthesis of CNOT circuits. We have the following algorithms to benchmark:

- GreedyGE for size optimization with full qubit connectivity from Chapter 9,
- cost minimization techniques for size optimization with full qubit connectivity from Chapter 9,
- DaCSynth for depth optimization with full qubit connectivity from Chapter 10,
- the syndrome decoding based algorithms for size optimization for both full and restricted connectivities from Chapter 11.

The state-of-the-art algorithms for each category are the following:

- The PMH algorithm [152, Algo. 1] for size optimization and full qubit connectivity, see Algorithm 8.2,
- The extension of Kutin *et al.*'s algorithm [117] for depth optimization with full qubit connectivity, see Algorithm 8.3.
- Steiner tree based algorithm [102, 146] for size optimization on constrained architectures.

Two kinds of datasets are used to benchmark our algorithms:

- First, a set of random operators. The test on random operators gives an overview of the average performance of our algorithm. We generate random operators by creating random CNOT circuits. Our routine takes two inputs: the number of qubit n and the number of

CNOT gates k in the random circuit. Each CNOT is randomly placed by selecting a random control and a random target and the simulation of the circuit gives a random operator. Empirically we noticed that when k is sufficiently large – $k = n^2$ is enough – then the operators generated have strong probability to represent the worst case scenarii. The same can be done with the depth as parameter instead of the number of gates.

- Secondly, a set of reversible functions, given as circuits, taken from Matthew Amy's github repository [8]. This experiment shows how our algorithms can optimize useful quantum algorithms in the literature like the Galois Field multipliers, integer addition, Hamming coding functions, the hidden weighted bit functions, etc.

To evaluate the performance of our algorithms for the random set, two types of experiments are conducted:

1. a worst-case asymptotic experiment, namely for increasing problem sizes n we generate circuits with n^2 gates/depth $2n$ and we compute the average number of gates/depth for each problem size. This experiment reveals the asymptotic behavior of the algorithms and gives insights about strict upper bounds on their performance.

2. a close-to-optimal experiment, namely for one specific problem size we generate operators with different number of gates/depth to show how close to optimal our algorithms are if the optimal circuits are expected to be smaller than the worst case.

All our algorithms are implemented in Julia [22] and executed on the ATOS QLM (Quantum Learning Machine) whose processor is an Intel Xeon(R) E7-8890 v4 at 2.4 GHz.

The outline of the chapter is the following: first, we present the benchmarks about the size optimization in the full qubit connectivity case in Section 12.1. Secondly, in Section 12.2 we show the benchmarks for depth optimization with full connectivity. Next, we give the benchmarks for the size optimization for constrained architectures in Section 12.3. In these three sections only benchmarks on random operators are considered. We present in Section 12.4 the benchmarks of the library of reversible functions, both for size and depth optimizations. Finally, we also count the number of reversible functions up to row and column permutation in Section 12.5.

12.1 Size Optimization with Full Qubit Connectivity

First, we present the worst-case asymptotic experiment with the following algorithms:

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

Algorithm	Average computational time (s)				
	$n = 10$	$n = 100$	$n = 500$	$n = 1000$	$n = 5000$
Gaussian elimination	0.00004	0.00125	0.14	1.2	140
PMH algorithm	0.00015	0.006	0.75	7	980
GreedyGE	0.00015	0.0035	0.1	0.6	45

Table 12.1: Computational time of GreedyGE vs PMH and standard Gaussian elimination algorithms.

The results are given in Fig 12.1. For the sake of clarity, instead of showing the circuit size we plot the ratio between the circuit size given by our method and the circuit size given by the PMH algorithm. So if the ratio is below 1 this means that we outperform the state-of-the art method [152, Algo. 1] . We also have a better view of which algorithm is better in which qubit range. We stopped the calculations when the running time was too large for producing benchmarks within several hours.

Overall, for the considered range of qubits, all our algorithms outperform [152, Algo. 1] . The integer programming solver gives the best results with with a maximum gain of more than 55% but its scalability is limited: beyond 50 qubits it requires too much computational time. Using commercial softwares for addressing larger problem sizes would be interesting to confirm the tendency toward an increasing gain.

Concerning the cost minimization heuristic for solving the syndrome decoding problem, it seems better to increase the depth of search rather than the width. With depth 3 and width 15 we have the best results for the range 70-125 with at least 30% of gain. Surprisingly, the Information Set Decoding 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 Information Set Decoding method cannot compete with the other versions of the cost minimization heuristic.

As the number of qubits increases the syndrome decoding based algorithm performs worse and GreedyGE eventually produces the shortest circuits when $n > 120$. Notably, the average gain over the PMH algorithm stabilizes around 25% whereas the syndrome decoding based algorithms performances continue to deteriorate such that for n large enough the PMH algorithm is better. We cannot tell if it is due to the method in itself or to the solution of the syndrome decoding problem that becomes less and less optimal as the problem size increases. We leave this question as a future work. For large n, GreedyGE is the best method so far. GreedyGE is also fast: we compare the computational time of GreedyGE against the standard Gaussian elimination and the PMH algorithm. The results are given in Table 12.1. Not only GreedyGE produces the smallest circuits but in addition it is the fastest method to our knowledge.

Next, we show the impact of the choice of the LU decomposition. We provide results with GreedyGE considering it can be directly transposed to the syndrome decoding based algorithms as well. We perform the close-to-optimal experiment with the GreedyGE and three different LU decompositions:

- a standard LU decomposition for matrix in $\mathbb{F}_2$, taken from the Julia package Nemo [60],
- the LU decomposition algorithm of Chapter 9 with the "sparse" strategy,

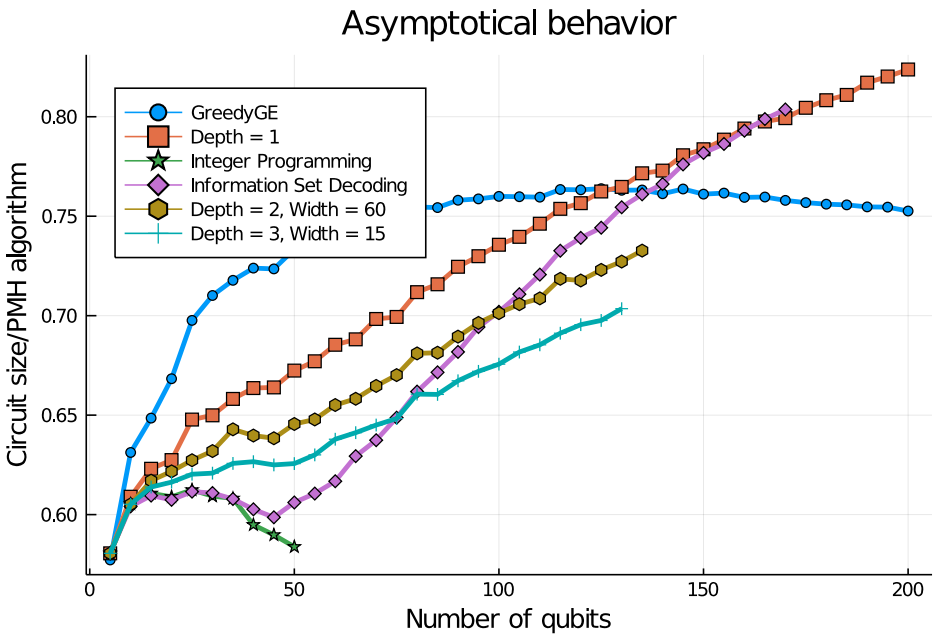


Figure 12.1: Average performance of GreedyGE and the Syndrome Decoding based algorithms versus the PMH algorithm.

- the LU decomposition algorithm of Chapter 9 with the "cost minimization" strategy.

The experiment is done on 60 qubits. We also add the best method for this problem size: the syndrome decoding based algorithm with the "Information Set Decoding" strategy. The results are given in Fig 12.2. Computing more efficient LU decompositions has almost no effect on the worst-case results but provides some improvements when the input circuits are smaller. There is no significant difference between the two strategies "sparse" and "cost minimization" in terms of circuit sizes but the running time is much lower for the sparse approach so we would privilege it. This improvement also benefits the syndrome decoding based method as it also relies on an LU decomposition.

Similarly to the worst case, the syndrome decoding based algorithm, in this range of qubits, is better even when the input circuits are expected to be small.

12.1.1 Path Finding Methods

We now evaluate the performance of purely greedy methods. We remind that we have four cost functions to study:

- $h_{sum}(A) = \sum_{i,j} a_{i,j}$,
- $h_{\text{prod}}(A) = \sum_{i=1}^{n} \log\left(\sum_{j=1}^{n} a_{i,j}\right)$,
- $H_{sum}(A) = h_{sum}(A) + h_{sum}(A^{-1})$,
- $H_{\text{prod}}(A) = h_{\text{prod}}(A) + h_{\text{prod}}(A^{-1})$.

First, we show the asymptotic behavior of those four methods. The results are given in Fig 12.3. The first graph in Fig 12.3a shows that the cost function h_{sum} does not scale well with the number

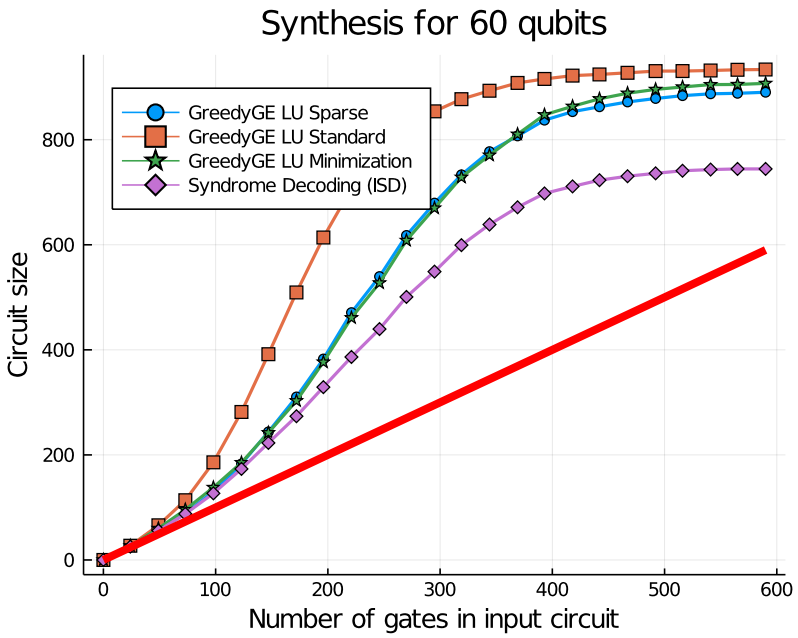


Figure 12.2: Performance of GreedyGE Syndrome Decoding on 60 qubits for different input circuit sizes.

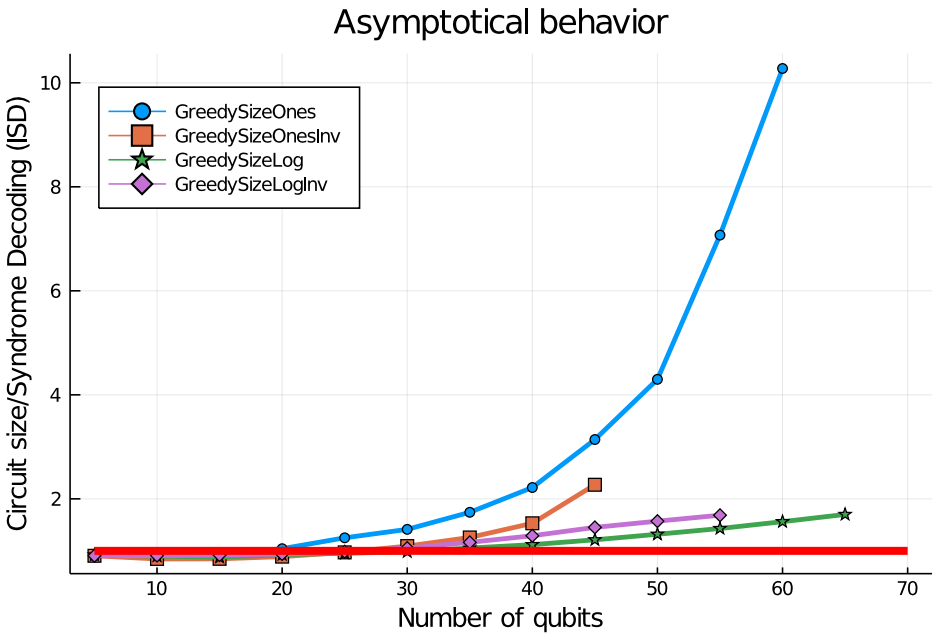
of qubits so we decide to remove it for clarity. The new graph is given in Fig 12.3b. We notice two things: first, the cost function H_{prod} always underperforms H_{sum}, secondly both H_{sum} and h_{prod} outperform our syndrome decoding based algorithm for small problem sizes ($n < 30$) but with an advantage for H_{sum} when $n < 25$. There is a thin window — between 25 and 30 qubits — where it is preferable to use h_{prod} instead of H_{sum}.

The results given in Fig 12.3 are mainly there to discredit two of the four cost functions: h_{sum} and H_{prod}. The result that really interests us is that of the close-to-optimal experiment, given in Fig 12.4. For this problem size on the worst case the syndrome decoding based algorithm outperforms our greedy methods. But when we are not in the worst case the results are completely different: the greedy methods follow more faithfully the bound $y = x$ than the syndrome decoding based algorithm. We have to wait input circuits of size 300 for the syndrome decoding algorithm to be better. The greedy algorithm based on the cost function h_{prod} produces much more stable results than the one with the cost function H_{sum}. For the cost function H_{sum} the variance is extremely large because the method struggles finding a global minimum. However, it is H_{sum} that produces the best results most of the time in the range of size $0-300$. Again there is a thin window where it may be relevant to use h_{prod} instead of H_{sum}.

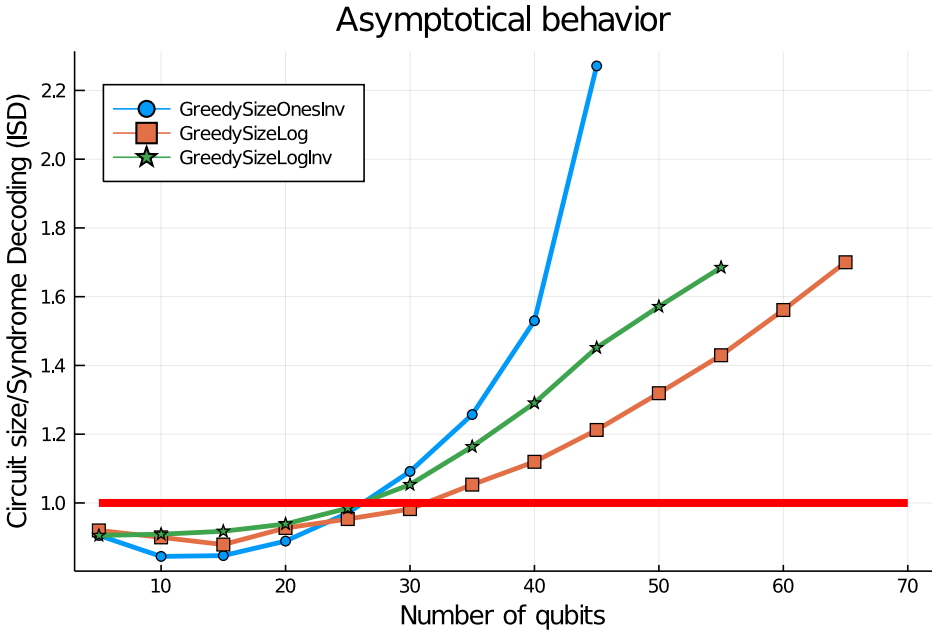
12.1.2 Conclusion: Combining the Methods

Similarly to what we have seen during Part B of this thesis with the synthesis in $\mathcal{SU}(2^n)$, each method has its own range of validity:

- GreedyGE is suited when the number of qubits exceeds 120, it outperforms the PMH algorithm and our other algorithms in both circuit size and computational time.
- The syndrome decoding based algorithms must be used for intermediate problem sizes ($30 < n < 120$) with the best possible solver of the syndrome decoding problem (in order: inte-



(a) Average performance of $\{h_{sum}, h_{prod}, H_{sum}, H_{prod}\}$ vs Syndrome decoding (Information Set Decoding).



(b) Average performance of $\{h_{prod}, H_{sum}, H_{prod}\}$ vs Syndrome decoding (Information Set Decoding).

Figure 12.3: Average performance of Cost minimization techniques vs Syndrome decoding (Information Set Decoding).

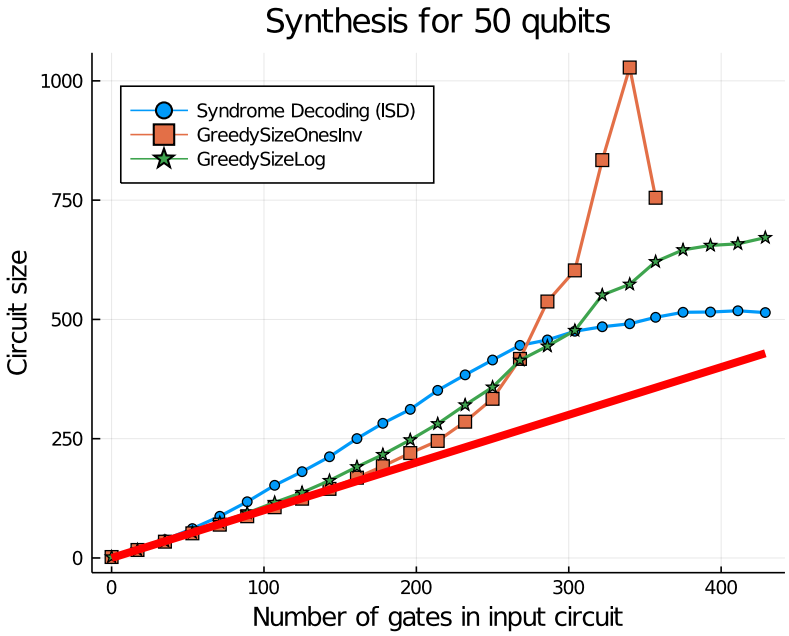


Figure 12.4: Performance of Cost minimization techniques vs Syndrome Decoding on 50 qubits for different input circuit sizes.

ger programming, Information Set Decoding if $n < 70$, maximum depth otherwise). Both GreedyGE and the family of syndrome decoding based algorithms are state of the art in a worst case scenario.

- When the output circuit is expected to be small, or when $n < 30$, then direct greedy methods have shown to produce the best results. The proposal of a new cost function paid off as this offers a more scalable direct greedy search. However, the moment when our new custom function outperforms the one proposed in [169] coincides with the moment when the syndrome decoding based algorithm also outperforms the direct greedy search. More investigation needs to be done to clarify which cost function should be preferred and when.

Given the variety of methods, each providing the best results in particular cases, the best method when trying to synthesize an operator would be simply to test each method and to keep the best result. Except for the direct greedy methods, the computational time of each method is well understood. Given a specific problem size, it is easy to know which method can be used and how long it will take. It is more delicate with the direct greedy methods, we have to set a limit in the maximum number of iterations before considering that the search will not converge to a solution.

12.2 Depth Optimization with Full Qubit Connectivity

First, we need an explicit way to compute the depth. This task is not trivial compared to the number of gates in the circuits. The most common way to perform this computation is to create a Directed Acyclic Graph representation of the circuit: the vertices of the graph are the gates and the edges represent their inputs/outputs. The depth of the circuit is then given by the longest path in the graph which can be computed by doing a topological sorting of the vertices for example. An interesting feature of this procedure is that we recover the skeleton circuit outlined in [130]

by computing the depth of a circuit returned by the standard Gaussian elimination algorithm. We therefore only benchmarked the Gaussian elimination method and the adapted method from [117].

We did the worst-case asymptotic experiment for $n = 1...100$ and the close-to-optimal experiment on $n = 40$ qubits with input circuits depth from 1 to $2n$. The results are plotted in Figure 12.5 and in Figure 12.6. In the worst case, for medium values of n ($n < 50$) the depth is slightly bigger than n but then it tends to approximately $0.85n$. This is a factor of 4 smaller than the standard Gaussian elimination results and a factor of 2 smaller than the method given in [117]. This also highlights that there may still be room for improvements in the theoretical upper bound we derived in section 10.1.

When the input circuit is already shallow, our algorithm still is able to re-synthesize a circuit with small depth, although it cannot give optimal results: this is represented by the gap with the line $y = x$ in Figure 12.6. We are again better than the state-of-the-art algorithms but this gap can still be improved.

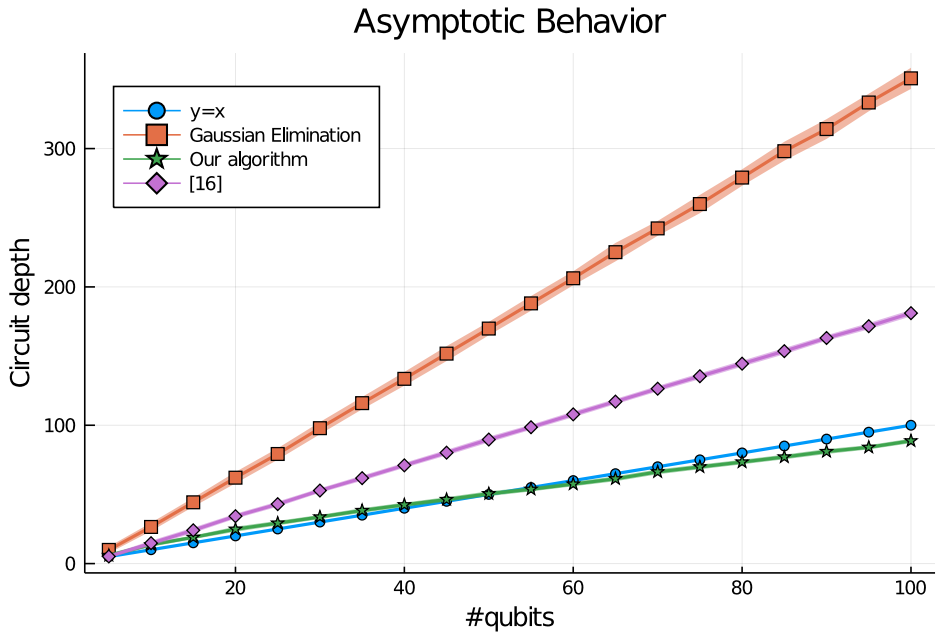


Figure 12.5: Average performance of our algorithm vs Gaussian elimination algorithm and [117].

With Ancillae

We repeat the first experiment but this time with various number of ancillary qubits. Namely, for different problem sizes we generated 100 random operators. Then we add progressively some random parities to our operator and we do the synthesis again. We are not particularly interested this time in the average depth required to synthesize those bigger operators but rather in the average increase in the depth as we add more and more ancillae. The results are given in Figure 12.7 for $n = 5, n = 15$ and $n = 50$. We present the results for both our extension of the divide-and-conquer framework and the block strategy described in Chapter 10. Overall, we cannot avoid an increase of the depth but this increase seems independent of the size of the problem and is mostly moderate with respect to the number of extra parities. For the extension of our framework, especially, when the number of ancillae is large, the increase of the depth seems to stabilize to a linear growth with a slope of at most 0.15. For the block strategy, we recover the theoretical results: the

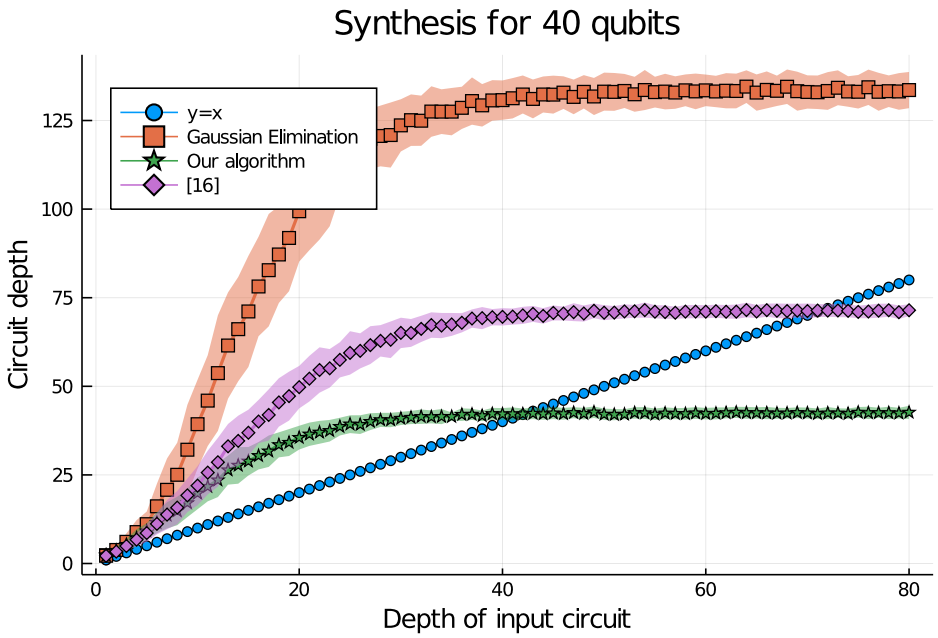


Figure 12.6: Performance of our algorithm vs Gaussian elimination algorithm and [117] on 40 qubits for different input circuits depths.

depth oscillates following an average increase given by a logarithmic overhead in p/n. If the number of ancillae is extremely large, it is preferable to use the extension rather than the block version because the logarithmic overhead is more important than the natural increase of our extension. Such promising results tend to encourage the use of ancillary qubits when synthesizing CNOT+T circuits.

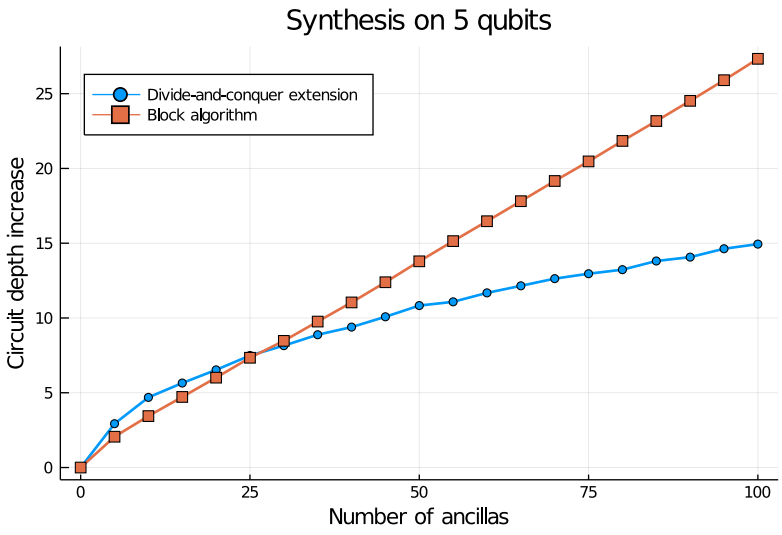
12.3 Size Optimization on Constrained Architecture

We compare the method we presented in Section 11.2 against the best algorithm in the literature [102] whose source code is available on the PyZX Github repository [101]. The code provided gives a way to generate random operators and compute several synthesis methods. For each architecture considered in the authors' implementation, we generate a set of 100 random operators and perform the synthesis using the Steiner trees. The algorithm from [102] also 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 a version of the algorithm from [102] without this extra optimization.

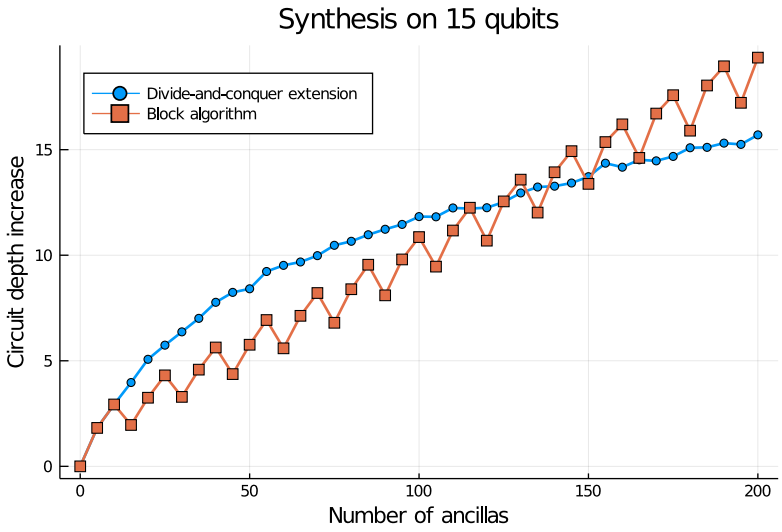
For our own implementation we set a time limit of 10 minutes for the synthesis of an operator. We recall that:

- $P_{\max}$ 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.
- N_{iter} is the number of times that the synthesis has been repeated.

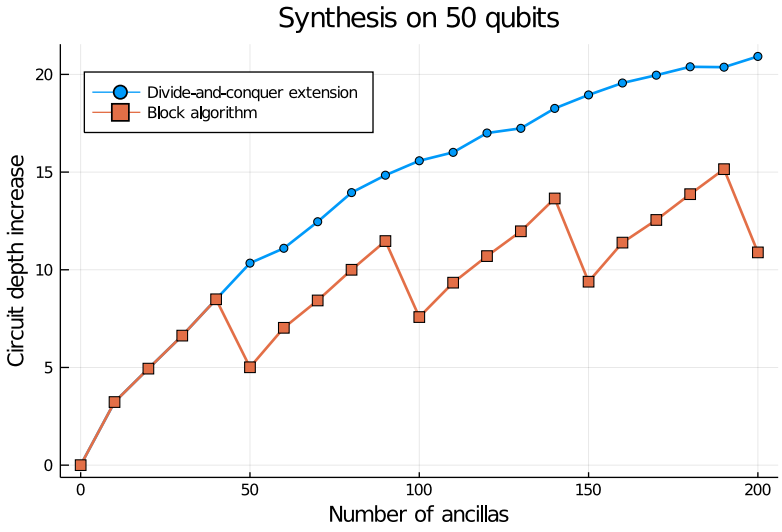
The values of the parameters for the different problem sizes are the following:



(a) 5 qubits



(b) 15 qubits



(c) 50 qubits

Figure 12.7: Average increase of the depth with the number of ancillae for different problem sizes.

Architecture	#	Steiner [102]	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 12.2: Performance of our Syndrome Decoding based algorithm vs Steiner trees algorithm [102] for several architectures.

- $n < 36$, $P_{\max} = \text{Inf}$, $N_{\text{iter}} = 100$ and $N_{\text{iter\_syndrome}} = 1$,
- $n = 36$, $P_{\max} = \text{Inf}$, $N_{\text{iter}} = 20$ 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 12.2. Columns 3 and 4 give the average size of the generated circuits for the method using Steiner trees in [102] 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.

12.4 Benchmarks on Reversible Functions

We apply our CNOT circuit synthesis to the optimization of quantum algorithms. A standard library to evaluate the quality of a quantum compiler is a library of reversible functions consisting of adders, Hamming coding functions, multipliers, etc. The circuits are from Matthew Amy's github Feynman repository [8].

Recent quantum compilers mainly focused on the T-count or T-depth optimization of quantum circuits but these optimizations often lead to an increased CNOT count. Although it is experimentally more costly to implement a T gate, the total number of CNOT gates in a quantum circuits should not be too large otherwise the CNOT cost of the total circuit will not be negligible. To give a rough overview, if the T gate is 50 times more costly than the CNOT gate but we have 20 more CNOT gates in the circuit then then the CNOT cost will be 1/3 of the total cost of implementing the circuit. In practice, the CNOT cost cannot be neglected.

In [9] the algorithm Gray-Synth for the optimization of the CNOT count in CNOT+T circuits is proposed. It efficiently reduces the total number of CNOT gates but it comes at the price of an increased T-depth, i.e., a longer circuit runtime. Here we show that, using our methods for CNOT circuit synthesis, one can reduce significantly the CNOT count without paying the price of increasing the T depth. We used the C++ implementation of Matthew Amy's Tpar algorithm to produce optimized circuits with low T-count and T-depth. We post process the circuits by re-synthesizing any chunk of purely CNOT circuits. It is possible to further optimize the synthesis of the CNOT+T circuits, using for instance the TODD optimizer [80] but as we are concerned about showing the impact of our method on the CNOT count and the T-depth we did not pursue the optimization to have a lower T-count so that we can directly compare against the results from [9].

The results are given in Table 12.4. For each reversible function we provide the results (T-count, T-depth, Total depth) of different circuits implementing it. Namely, the original circuits, the circuits optimized solely with the Tpar algorithm and GraySynth as the state-of-the-art methods and the circuits optimized with the Tpar method and our own methods: the one for optimizing the size and DaCSynth. For Gray-Synth we used the Haskell implementation, still from [8]. The savings given always compare against the circuits given by solely the Tpar algorithm. We notice several interesting points:

- we cannot decrease the CNOT count as much as with GraySynth, but we still manage to decrease the CNOT count consistently and above all we do not modify the T-depth. Our circuits therefore represent a tradeoff between the T-cost the CNOT-cost.
- We also considerably reduce the total depth of the circuits, making their execution faster on NISQ architectures.
- Surprisingly, the depth is better optimized when we optimize the size rather than the depth. This is explained by the fact that most of the CNOT circuits involved in those reversible functions are elementary operators, requiring only a few CNOTs to be implemented. Therefore

the size and the depth are very close and greedy methods find a close-to-optimal implementations while DaCSynth may have some overhead. Overall if among 100 CNOT subcircuits DaCSynth has an overhead of 1 in the depth for 50 of them, the total depth is increased by at most 50 which can be huge and this can explain the not as good results we have with DaCSynth.

To illustrate our last statement, we kept track of the best algorithm that was used to compute each CNOT sub-circuit that appeared in the synthesis of one reversible function. The results are given in Table 12.5. Most of the time, both pure greedy methods and the syndrome decoding based methods provide the best results. According to us, this highlights the simplicity of the operators to synthesize otherwise there would be many more cases where we would observe a difference of even one CNOT. Overall, for this library of reversible functions, the use of methods other than the pure greedy methods is useless. This is not surprising as the number of qubits never exceeds 40 except for three operators. For such small problem sizes, we saw that pure greedy methods always outperform the other methods, in addition to the fact that we have strong suspicions that the majority of the operators encountered are easy to synthesize. We note the interest of using the cost function $h_{\log}$ as an alternative because in some cases it allows to synthesize operators more efficiently.

12.5 Database of Optimal Reversible Circuits

In this section, we develop the idea of using a database to store the optimal circuits for linear reversible operators of small size. This idea is not new and occurred multiple times in the literature. In [67] by exploiting some symmetries in the reversible functions representation the authors stored optimal results for all the 4-bits reversible functions. In [11] a meet-in-the-middle method was used to store optimal quantum circuits up to a certain depth for small number of qubits. Similarly to our problem, in [106] these methods were applied to store Clifford quantum circuits and were also applied to the specific case of linear reversible circuits. Yet they were limited to 6 lines before exhausting the available memory.

First, it is only necessary to construct a database that stores the minimum number of CNOT gates required for the implementation of any linear reversible function. Indeed, given a specific operator to synthesize, we already know how many CNOTs are required for its implementation. Then all we need is to try every possible row operation/CNOT gate and find a new operator that needs one less CNOT for its implementation. We store the chosen CNOT and we repeat on the new operator until we reach the identity operator.

Storing blindly all the possible linear reversible functions is not efficient at all. In Table 12.3 we computed the number of linear reversible functions up to 9 lines and we see that the problem becomes already intractable for 6/7 lines. A common way to reduce the total size of the database is to exploit the symmetries of linear reversible functions to avoid storing redundant informations. For instance, if we know how to synthesize an operator A then we know how to implement A^{-1} by reading the circuit in reverse order. This reduces the size of the database by a factor of 2. Then, if we know how to implement an operator A we know how to implement any operator P_1AP_2, where P_1, P_2 are permutation matrices, by independently renaming the inputs and outputs. Using this symmetry, we can approximately reduce the size of the database by a factor of $(n!)^2$ where n is the number of lines. We give the estimated new sizes of the databases in Table 12.3 and we see that the problem is tractable up to 8 lines. The key to use efficiently this symmetry lies in the possibility to compute a canonical representative for the equivalence relation "inputs/outputs independent renaming".

n	Number of linear reversible functions (exact)	Simultaneous I/0 renaming (estimated)	Independent I/0 renaming (estimated)	Inverse and independent I/0 renaming (estimated)
1	1	1	1	1
2	6	3	2	1
3	168	28	5	2
4	20,160	840	35	18
5	9,999,360	83,328	694	347
6	20,158,709,760	27,998,208	38,886	19,443
7	163,849,992,929,280	32,509,919,232	6,450,381	3,225,190
8	5,348,063,769,211,699,200	132,640,470,466,560	3,289,694,208	1,644,847,104
9	699,612,310,033,197,642,547,200	1,927,943,976,061,501,440	5,312,896,759,429	2,656,448,379,714
10	66,440,137,299,948,128,422,802,227,200	100,981,078,400,558,897,823,744	27,827,678,130,665,481	13,913,839,065,332,741

Table 12.3: Sizes or estimated sizes of different databases.

Function	#n	Original				Tpar				Tpar + GraySynth				Tpar + CNOT size opt.				Tpar + CNOT depth opt.			
		T-count	T-depth	CNOT count	Depth	T-count	T-depth	CNOT count	Depth	T-count	T-depth	CNOT count	Depth	T-count	T-depth	CNOT count	Depth	T-count	T-depth	CNOT count	Depth
Adder_8	24	399	69	466	223	213	30	741	302	215	74 (147%)	399 (−46%)	257 (−15%)	213	30	491 (−34%)	204 (-32%)	213	30	683 (−8%)	254 (−16%)
barenco_tof_10	19	224	96	224	288	100	43	332	272	100	81 (88%)	146 (−56%)	264 (−3%)	100	43	188 (−43%)	217 (-20%)	100	43	262 (−21%)	267 (−2%)
barenco_tof_3	5	28	12	28	36	16	8	52	60	16	16 (100%)	20 (−62%)	40 (−33%)	16	8	26 (−50%)	34 (-43%)	16	8	34 (−35%)	41 (−32%)
barenco_tof_4	7	56	24	56	72	28	13	96	96	28	26 (100%)	38 (−60%)	64 (−33%)	28	13	50 (−48%)	61 (-36%)	28	13	67 (−30%)	78 (−19%)
barenco_tof_5	9	84	36	84	108	40	18	134	123	40	34 (89%)	56 (−58%)	88 (−28%)	40	18	73 (−46%)	89 (-28%)	40	18	98 (−27%)	108 (−12%)
csla_mux_3	15	70	21	90	67	62	8	379	210	62	28 (250%)	115 (−70%)	91 (−57%)	62	8	187 (−51%)	81 (-61%)	62	8	312 (−18%)	128 (−39%)
csum_mux_9	30	196	18	196	59	84	6	366	153	84	16 (167%)	160 (−56%)	78 (−49%)	84	6	179 (−51%)	70 (-54%)	84	6	315 (−14%)	110 (−28%)
cycle_17_3	35	4739	2001	4742	5974	1944	562	6608	5231	1955	1857 (230%)	3040 (−54%)	5698 (9%)	1944	562	4267 (−35%)	4229 (-19%)	1944	562	6111 (−8%)	5102 (−2%)
GF(2^{10})_mult	30	700	108	709	290	410	16	2206	1026	410	109 (581%)	648 (−71%)	324 (−68%)	410	16	955 (−57%)	307 (-70%)	410	16	2294 (4%)	401 (−61%)
GF(2^{16})_mult	48	1792	180	1837	489	1040	24	6724	2551	1040	585 (2338%)	1691 (−75%)	1488 (−42%)	1040	24	2542 (−62%)	678 (-73%)	1040	24	7724 (15%)	788 (−69%)
GF(2^{32})_mult	96	7168	372	7292	1001	4128	47	34244	11520	4128	2190 (4560%)	6636 (−81%)	5391 (−53%)	4128	47	10953 (−68%)	2113 (-82%)	4128	47	49895 (46%)	2397 (−79%)
GF(2^4)_mult	12	112	36	115	99	68	6	307	173	68	39 (550%)	106 (−65%)	117 (−32%)	68	6	135 (−56%)	80 (-54%)	68	6	285 (−7%)	116 (−33%)
GF(2^5)_mult	15	175	48	179	130	115	9	502	259	115	51 (467%)	166 (−67%)	152 (−41%)	115	9	209 (−58%)	107 (-59%)	115	9	391 (−22%)	138 (−47%)
GF(2^6)_mult	18	252	60	257	163	150	9	660	350	150	63 (600%)	235 (−64%)	189 (−46%)	150	9	310 (−53%)	137 (-61%)	150	9	694 (5%)	180 (−49%)
GF(2^7)_mult	21	343	72	349	195	217	12	996	490	217	75 (525%)	319 (−68%)	224 (−54%)	217	12	442 (−56%)	184 (-62%)	217	12	940 (−6%)	228 (−53%)
GF(2^8)_mult	24	448	84	469	233	264	13	1254	619	264	87 (569%)	428 (−66%)	266 (−57%)	264	13	588 (−53%)	224 (-64%)	264	13	1386 (11%)	293 (−53%)
GF(2^9)_mult	27	567	96	575	258	351	15	1712	810	351	95 (533%)	526 (−69%)	283 (−65%)	351	15	753 (−56%)	266 (-67%)	351	15	1805 (5%)	341 (−58%)
grover_5	9	336	144	336	457	154	51	499	477	154	130 (155%)	232 (−54%)	440 (−8%)	154	51	331 (−34%)	384 (-19%)	154	51	424 (−15%)	432 (−9%)
ham15-high	20	2457	996	2500	3026	1019	380	3427	2956	1019	839 (121%)	1588 (−54%)	2583 (−13%)	1019	380	2183 (−36%)	2226 (-25%)	1019	380	3087 (−10%)	2731 (−8%)
ham15-low	17	161	69	259	263	97	33	471	360	97	83 (152%)	221 (−53%)	271 (−25%)	97	33	280 (−41%)	223 (-38%)	97	33	415 (−12%)	281 (−22%)
ham15-med	17	574	240	616	750	230	84	759	682	230	194 (131%)	368 (−52%)	622 (−9%)	230	84	481 (−37%)	506 (-26%)	230	84	672 (−11%)	632 (−7%)
hwb6	7	105	45	131	152	75	24	270	248	75	63 (162%)	111 (−59%)	188 (−24%)	75	24	172 (−36%)	164 (-34%)	75	24	250 (−7%)	209 (−16%)
hwb8	12	5887	2139	7970	7956	3531	860	22670	15838	3531	2752 (220%)	6841 (−70%)	9338 (−41%)	3531	860	13353 (−41%)	9261 (-42%)	3531	860	20860 (−8%)	12290 (−22%)
mod5_4	5	28	12	32	41	16	6	48	57	16	15 (150%)	28 (−42%)	51 (−11%)	16	6	32 (−33%)	39 (-32%)	16	6	40 (−17%)	47 (−18%)
mod_adder_1024	28	1995	831	2005	2503	1011	258	3650	2560	1011	864 (235%)	1390 (−62%)	2474 (−3%)	1011	258	2369 (−35%)	1913 (-25%)	1011	258	3323 (−9%)	2392 (−7%)
mod_adder_1048576	58	17290	7292	17310	21807	7298	1927	29794	19975	7323	6577 (241%)	11080 (−63%)	20089 (1%)	7298	1927	19122 (−36%)	15286 (-23%)	7298	1927	27522 (−8%)	18799 (−6%)
mod_mult_55	9	49	15	55	50	35	7	106	75	35	20 (186%)	40 (−62%)	52 (−31%)	35	7	73 (−31%)	61 (-19%)	35	7	88 (−17%)	57 (−24%)
mod_red_21	11	119	48	122	158	73	25	223	207	73	59 (136%)	102 (−54%)	179 (−14%)	73	25	136 (−39%)	144 (-30%)	73	25	183 (−18%)	174 (−16%)
qcla_adder_10	36	238	24	267	73	162	13	648	195	162	33 (154%)	225 (−65%)	94 (−52%)	162	13	362 (−44%)	86 (-56%)	162	13	548 (−15%)	128 (−34%)
qcla_com_7	24	203	27	215	81	94	12	371	154	94	31 (158%)	150 (−60%)	89 (−42%)	94	12	202 (−46%)	76 (-51%)	94	12	274 (−26%)	106 (−31%)
qcla_mod_7	26	413	66	441	197	231	28	813	296	237	67 (139%)	386 (−53%)	212 (−28%)	231	28	479 (−41%)	188 (-36%)	231	28	701 (−14%)	238 (−20%)
qft_4	5	69	48	48	142	67	44	96	185	67	60 (36%)	48 (−50%)	163 (−12%)	67	44	56 (−42%)	150 (-19%)	67	44	78 (−19%)	165 (−11%)
rc_adder_6	14	77	33	104	104	47	22	165	157	47	39 (77%)	79 (−52%)	111 (−29%)	47	22	100 (−39%)	103 (-34%)	47	22	120 (−27%)	113 (−28%)
tof_10	19	119	51	119	153	71	27	236	190	71	61 (126%)	86 (−64%)	172 (−9%)	71	27	130 (−45%)	143 (-25%)	71	27	185 (−22%)	173 (−9%)
tof_3	5	21	9	21	27	15	6	35	46	15	12 (100%)	16 (−54%)	32 (−30%)	15	6	21 (−40%)	31 (-33%)	15	6	26 (−26%)	36 (−22%)
tof_4	7	35	15	35	45	23	9	63	71	23	18 (100%)	26 (−59%)	51 (−28%)	23	9	37 (−41%)	49 (-31%)	23	9	49 (−22%)	54 (−24%)
tof_5	9	49	21	49	63	31	12	97	104	31	24 (100%)	36 (−63%)	70 (−33%)	31	12	50 (−48%)	67 (-36%)	31	12	70 (−28%)	78 (−25%)
vbe_adder_3	10	70	24	80	79	24	9	120	88	24	18 (100%)	54 (−55%)	71 (−19%)	24	8	61 (−49%)	45 (-49%)	24	9	75 (−38%)	52 (−41%)
Mean savings											+391.39%	-60.21%	-29.66%			-45.03%	-41.26%			-12.61%	-27.68%
Minimum savings											+36%	-81%	-68%			-68%	-82%			-38%	-79%
Maximum savings											+4560%	-42%	+9%			-31%	-19%			+46%	-2%

Table 12.4: CNOT optimization of a library of reversible functions with several CNOT circuits synthesis methods.

Function	#n	#CNOT sub-circuits	PMH		GreedyGE		Syndrome (Information Set Decoding)		Syndrome (Depth 3)		Syndrome (IP)		Greedy Ones		Greedy Log	
			Best choice	Only Choice	Best choice	Only Choice	Best choice	Only Choice	Best choice	Only Choice	Best choice	Only Choice	Best choice	Only Choice	Best choice	Only Choice
Adder_8	24	60	12 (20%)	0 (0)	13 (22%)	0 (0%)	42 (70%)	0 (0%)	42 (70%)	0 (0%)	42 (70%)	0 (0%)	60 (100%)	2 (3%)	58 (97%)	0 (0%)
barenco_tof_10	19	99	29 (29%)	0 (0)	33 (33%)	0 (0%)	83 (84%)	0 (0%)	83 (84%)	0 (0%)	83 (84%)	0 (0%)	99 (100%)	0 (0%)	99 (100%)	0 (0%)
barenco_tof_3	5	14	8 (57%)	0 (0)	8 (57%)	0 (0%)	12 (86%)	0 (0%)	12 (86%)	0 (0%)	12 (86%)	0 (0%)	13 (93%)	0 (0%)	13 (93%)	0 (0%)
barenco_tof_4	7	27	12 (44%)	0 (0)	12 (44%)	0 (0%)	24 (89%)	0 (0%)	24 (89%)	0 (0%)	24 (89%)	0 (0%)	26 (96%)	0 (0%)	26 (96%)	0 (0%)
barenco_tof_5	9	39	16 (41%)	0 (0)	16 (41%)	0 (0%)	32 (82%)	0 (0%)	32 (82%)	0 (0%)	32 (82%)	0 (0%)	38 (97%)	0 (0%)	38 (97%)	0 (0%)
csla_mux_3	15	15	0 (0%)	0 (0)	0 (0%)	0 (0%)	8 (53%)	0 (0%)	8 (53%)	0 (0%)	8 (53%)	0 (0%)	15 (100%)	2 (13%)	10 (67%)	0 (0%)
csum_mux_9	30	14	1 (7%)	0 (0)	1 (7%)	0 (0%)	6 (43%)	0 (0%)	6 (43%)	0 (0%)	6 (43%)	0 (0%)	13 (93%)	4 (29%)	9 (64%)	0 (0%)
cycle_17_3	35	1436	219 (15%)	0 (0)	330 (23%)	0 (0%)	1094 (76%)	0 (0%)	1094 (76%)	0 (0%)	1094 (76%)	0 (0%)	1429 (100%)	23 (2%)	1412 (98%)	6 (0%)
GF(2^{10})_mult	30	20	0 (0%)	0 (0)	0 (0%)	0 (0%)	1 (5%)	0 (0%)	1 (5%)	0 (0%)	1 (5%)	0 (0%)	16 (80%)	13 (65%)	6 (30%)	3 (15%)
GF(2^{16})_mult	48	28	0 (0%)	0 (0)	0 (0%)	0 (0%)	2 (7%)	0 (0%)	2 (7%)	0 (0%)	2 (7%)	0 (0%)	18 (64%)	15 (54%)	11 (39%)	8 (29%)
GF(2^{32})_mult	96	51	0 (0%)	0 (0)	0 (0%)	0 (0%)	2 (4%)	0 (0%)	2 (4%)	0 (0%)	2 (4%)	0 (0%)	30 (59%)	28 (55%)	21 (41%)	19 (37%)
GF(2^{4})_mult	12	10	0 (0%)	0 (0)	0 (0%)	0 (0%)	3 (30%)	0 (0%)	3 (30%)	0 (0%)	3 (30%)	0 (0%)	9 (90%)	3 (30%)	6 (60%)	0 (0%)
GF(2^{5})_mult	15	13	1 (8%)	0 (0)	0 (0%)	0 (0%)	3 (23%)	0 (0%)	3 (23%)	0 (0%)	3 (23%)	0 (0%)	10 (77%)	6 (46%)	6 (46%)	2 (15%)
GF(2^{6})_mult	18	13	0 (0%)	0 (0)	0 (0%)	0 (0%)	2 (15%)	0 (0%)	2 (15%)	0 (0%)	2 (15%)	0 (0%)	9 (69%)	7 (54%)	5 (38%)	3 (23%)
GF(2^{7})_mult	21	16	0 (0%)	0 (0)	0 (0%)	0 (0%)	3 (19%)	0 (0%)	3 (19%)	0 (0%)	3 (19%)	0 (0%)	12 (75%)	7 (44%)	8 (50%)	3 (19%)
GF(2^{8})_mult	24	17	0 (0%)	0 (0)	0 (0%)	0 (0%)	2 (12%)	0 (0%)	2 (12%)	0 (0%)	2 (12%)	0 (0%)	15 (88%)	10 (59%)	6 (35%)	1 (6%)
GF(2^{9})_mult	27	19	0 (0%)	0 (0)	0 (0%)	0 (0%)	4 (21%)	0 (0%)	4 (21%)	0 (0%)	4 (21%)	0 (0%)	14 (74%)	10 (53%)	8 (42%)	4 (21%)
grover_5	9	123	39 (32%)	0 (0)	39 (32%)	0 (0%)	110 (89%)	0 (0%)	110 (89%)	0 (0%)	110 (89%)	0 (0%)	122 (99%)	0 (0%)	121 (98%)	0 (0%)
ham15-high	20	852	220 (26%)	0 (0)	256 (30%)	0 (0%)	698 (82%)	0 (0%)	698 (82%)	0 (0%)	698 (82%)	0 (0%)	852 (100%)	3 (0%)	849 (100%)	0 (0%)
ham15-low	17	69	21 (30%)	0 (0)	18 (26%)	0 (0%)	57 (83%)	0 (0%)	57 (83%)	0 (0%)	57 (83%)	0 (0%)	69 (100%)	2 (3%)	66 (96%)	0 (0%)
ham15-med	17	189	56 (30%)	0 (0)	63 (33%)	0 (0%)	167 (88%)	0 (0%)	167 (88%)	0 (0%)	167 (88%)	0 (0%)	189 (100%)	1 (1%)	188 (99%)	0 (0%)
hwb6	7	48	11 (23%)	0 (0)	12 (25%)	0 (0%)	40 (83%)	0 (0%)	40 (83%)	0 (0%)	40 (83%)	0 (0%)	47 (98%)	0 (0%)	47 (98%)	0 (0%)
hwb8	12	2130	270 (13%)	0 (0)	343 (16%)	0 (0%)	1248 (59%)	0 (0%)	1245 (58%)	0 (0%)	1247 (59%)	0 (0%)	2115 (99%)	159 (7%)	1938 (91%)	10 (0%)
mod5_4	5	14	8 (57%)	0 (0)	8 (57%)	0 (0%)	13 (93%)	0 (0%)	13 (93%)	0 (0%)	13 (93%)	0 (0%)	14 (100%)	0 (0%)	14 (100%)	0 (0%)
mod_adder_1024	28	716	173 (24%)	0 (0)	214 (30%)	0 (0%)	564 (79%)	0 (0%)	564 (79%)	0 (0%)	564 (79%)	0 (0%)	716 (100%)	28 (4%)	688 (96%)	0 (0%)
mod_adder_1048576	58	5615	857 (15%)	0 (0)	1335 (24%)	0 (0%)	4275 (76%)	0 (0%)	4276 (76%)	0 (0%)	4276 (76%)	0 (0%)	5593 (100%)	114 (2%)	5500 (98%)	21 (0%)
mod_mult_55	9	14	4 (29%)	0 (0)	4 (29%)	0 (0%)	9 (64%)	0 (0%)	9 (64%)	0 (0%)	9 (64%)	0 (0%)	14 (100%)	0 (0%)	14 (100%)	0 (0%)
mod_red_21	11	46	19 (41%)	0 (0)	19 (41%)	0 (0%)	40 (87%)	0 (0%)	40 (87%)	0 (0%)	40 (87%)	0 (0%)	45 (98%)	0 (0%)	44 (96%)	0 (0%)
qcla_adder_10	36	24	3 (12%)	0 (0)	4 (17%)	0 (0%)	17 (71%)	0 (0%)	17 (71%)	0 (0%)	17 (71%)	0 (0%)	23 (96%)	2 (8%)	20 (83%)	0 (0%)
qcla_com_7	24	26	4 (15%)	0 (0)	4 (15%)	0 (0%)	15 (58%)	0 (0%)	15 (58%)	0 (0%)	15 (58%)	0 (0%)	26 (100%)	0 (0%)	26 (100%)	0 (0%)
qcla_mod_7	26	52	5 (10%)	0 (0)	5 (10%)	0 (0%)	35 (67%)	0 (0%)	35 (67%)	0 (0%)	35 (67%)	0 (0%)	51 (98%)	2 (4%)	48 (92%)	0 (0%)
qft_4	5	29	9 (31%)	0 (0)	12 (41%)	0 (0%)	28 (97%)	0 (0%)	28 (97%)	0 (0%)	28 (97%)	0 (0%)	29 (100%)	0 (0%)	29 (100%)	0 (0%)
rc_adder_6	14	49	33 (67%)	0 (0)	34 (69%)	0 (0%)	43 (88%)	0 (0%)	43 (88%)	0 (0%)	43 (88%)	0 (0%)	48 (98%)	0 (0%)	48 (98%)	0 (0%)
tof_10	19	52	21 (40%)	0 (0)	21 (40%)	0 (0%)	48 (92%)	0 (0%)	48 (92%)	0 (0%)	48 (92%)	0 (0%)	51 (98%)	0 (0%)	50 (96%)	0 (0%)
tof_3	5	12	7 (58%)	0 (0)	8 (67%)	0 (0%)	11 (92%)	0 (0%)	11 (92%)	0 (0%)	11 (92%)	0 (0%)	11 (92%)	0 (0%)	11 (92%)	0 (0%)
tof_4	7	17	10 (59%)	0 (0)	11 (65%)	0 (0%)	16 (94%)	0 (0%)	16 (94%)	0 (0%)	16 (94%)	0 (0%)	16 (94%)	0 (0%)	16 (94%)	0 (0%)
tof_5	9	22	11 (50%)	0 (0)	11 (50%)	0 (0%)	18 (82%)	0 (0%)	18 (82%)	0 (0%)	18 (82%)	0 (0%)	21 (95%)	0 (0%)	21 (95%)	0 (0%)
vbe_adder_3	10	19	4 (21%)	0 (0)	4 (21%)	0 (0%)	18 (95%)	0 (0%)	18 (95%)	0 (0%)	18 (95%)	0 (0%)	18 (95%)	0 (0%)	18 (95%)	0 (0%)

Table 12.5: Frequency of best performance of each algorithm during the optimization of reversible circuits. For each algorithm, the first column gives the number of times it has returned the best result (possibly other algorithms returned circuits of same size). The second column reports the number of times it was the only one to provide the best possible circuit.

In [106] they evoke the possibility of using the independent renaming of the inputs and outputs and propose a way to compute the canonical representative. It is unclear if they applied this strategy. They stopped to a database for 6 lines although with an independent input/output renaming we estimate we can reach 8 qubits. Besides the way they compute the canonical representative can be improved because they compute a minimum over all possible row (or column) permutations, introducing a huge overhead such that it cannot be used to compute efficiently the database.

We use an efficient way of computing the canonical representative. This will enable us to generate quickly the database up to 8 lines. This is the same technique as in the proof of Corollary 10.1.2.2 where we did a BFS search on the set of binary matrices and reduced the size of the search space by considering equivalence class for operators where equivalence means permutation of rows or columns. In the proof, we briefly explained that the way to compute a canonical representative relies on graph isomorphisms. For completeness we give more details about this technique as it is central in the construction of the database.

An isomorphism of graphs G and H is a bijection between the vertex sets of G and H $f : V(G) \to V(H)$ such that two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H. Any binary matrix A of size n corresponds to the bi-adjacency matrix of a balanced bipartite graph $G = (U, V, E)$ with $|U| = |V| = n$. Given a binary matrix A and its associated bipartite graph $G = (U, V, E)$, applying a row, resp. column, permutation on A is equivalent to a renaming of the vertices in U, resp. V. In other words we get a new graph $G' = (U', V', E)$ where G' is isomorphic to G. We have an injection between a set of binary matrices that are equal up to row, resp. column, permutations and an isomorphic class of graphs but not a bijection: there may be a bijection f where a node in U has its image in V and f still preserves the adjacency relations. Consider a graph where one node in U and one node in V are not connected to any node: switching the two nodes gives a new graph isomorphic to the first but it does not correspond to a permutation of the rows or columns of the associated bi-adjacency matrix. In fact, you transpose the row and the column corresponding to the two nodes. As we do not want our operators to be equivalent up to some transpositions of rows and columns, we have to restrict the way we decide if two graphs are isomorphic. To overcome this we consider *colored* bipartite graphs: the vertices in U are colored in one color and the vertices in V are colored in another color. We impose the bijection to also preserve the color of the nodes. This enables to consider both row and column permutations as the bijections can only mix nodes of same color. This defines a new equivalence relation. Computing a canonical representative for a set of binary matrices equal up to row or column permutations is therefore equivalent to computing a canonical representative of an isomorphic class of graphs given by the custom equivalence relation. Determining whether two graphs are isomorphic is the graph isomorphism problem. This problem is believed not to be solvable in polynomial time nor to be NP-complete. Fortunately, to solve this problem, many efficient algorithms exist and directly compute a canonical representative. We used the nauty program [136] and linked it to our Julia code. Once we have a canonical representative we convert it into an integer on 64 bits by reading the binary sequence given by the boolean matrix of the canonical representative.

We performed a breadth-first search until we reached all the possible linear reversible functions. In a sequential run we can only generate a database for 7 qubits in approximately 10 minutes. With the exponential growth of the problem sizes it is impossible to generate a database for 8 qubits without a parallel implementation. For each integer k we iterated through a queue containing all the functions implementable with k gates in order to generate the queue $k+1$. By this way we can divide the work equally between each thread without the risk that a thread adds a function with a wrong optimal size (for instance, a "fast" thread reaches a function in K gates whereas another thread would have reached it with only K-1 gates). We gave one queue and one database for each

n	Independent I/0 renaming (exact)	Inverse and independent I/0 renaming (exact)
1	1	1
2	2	2
3	7	6
4	51	35
5	885	486
6	44,206	22,456
7	6,843,555	3,425,932
8	3,373,513,302	1,686,852,696

Table 12.6: Exact databases sizes.

thread and between two queues iteration we merged the results. This approach may not be the optimal one but it is enough to generate the database in less than 48h with 24 threads. We give in table 12.6 the number of different linear reversible functions up to independent inputs/outputs renaming. This sequence follows the sequence A224879 and we are able to add one value to the sequence for the case $n = 8$. The distribution of the sizes of the circuits is given table 12.7.

We realized after performing the computation that a similar method was done in [63] for counting the number of linear codes. Yet the procedure was stopped at $n = 7$.

We can exploit the fact than if we know how to implement a function f optimally then we know how to implement f^{-1} as well. Indeed it is sufficient to read the circuit in reverse. When computing the canonical representative of f we can take the minimum of the two 64-bits integers given by the canonical representative of f and f^{-1}. Again after performing a breadth-first search we get the results Table 12.6 and Table 12.8.

Number of gates \ Number of lines	1	2	3	4	5	6	7	8
0	1	1	1	1	1	1	1	1
1		1	1	1	1	1	1	1
2			4	5	5	5	5	5
3			1	18	22	23	23	23
4				19	114	141	145	146
5				6	293	883	1,046	1,073
6				1	346	4,123	8,311	9,430
7					97	12,314	57,516	89,874
8					6	17,967	311,380	829,495
9						8,049	1,161,639	6,721,597
10						694	2,494,728	44,179,051
11						4	2,275,942	217,421,135
12						1	524,718	710,850,603
13							8,098	1,284,845,188
14							2	947,875,714
15								159,305,838
16								1,384,123
17								5

Table 12.7: Number of linear reversible functions reachable for different problem and circuit sizes.

Number of gates \ Number of lines	1	2	3	4	5	6	7	8
0	1	1	1	1	1	1	1	1
1		1	1	1	1	1	1	1
2			3	4	4	4	4	4
3			1	12	15	16	16	16
4				12	66	84	87	88
5				4	157	471	564	582
6				1	184	2,121	4274	4,875
7					53	6,245	29,031	45,410
8					5	9,069	146,258	416,152
9						4,076	581,664	3,364,233
10						364	1,248,453	22,097,401
11						3	1,138,735	108,724,675
12						1	262,742	355,447,216
13							4,100	642,445,628
14							2	473,954,811
15								79,658,969
16								692,630
17								4

Table 12.8: Number of linear reversible functions reachable for different problem and circuit sizes (with inverse).

Chapter 13

Conclusion and perspectives on CNOT circuits synthesis

Contents

13.1 Size optimization with full qubit connectivity

In Part C of this manuscript, we studied three different kinds of algorithms for the synthesis of CNOT circuits with full qubit connectivity and with the size of the circuit as objective:

- a Gaussian elimination based algorithm called GreedyGE, where we greedily choose the row operation that maximizes the number of zero left-most elements of the operator to synthesize,
- purely greedy methods, where we choose the row operation that minimizes a cost function,
- an algorithm based on the syndrome decoding problem, where we reformulate the problem as a series of syndrome decoding problems for which various strategies are applied.

Overall, each algorithm gives the best results depending on the number of qubits or the size of the optimal circuit implementing the desired operator:

- For large n ($n > 150$) in a worst-case scenario, GreedyGE is the best algorithm. The algorithm is fast and it produces circuits with around 25% less CNOTs than the PMH algorithm.
- For medium n ($30 < n < 150$) and in a worst-case scenario, the syndrome decoding based algorithm gives the best results. Especially, the better the solver of the syndrome decoding problem is, the shorter the produced circuits are. But optimal or near-optimal methods do not scale well. We report up to 55% of gain compared to the PMH algorithm.
- For small n ($n < 30$) or when the optimal circuit is expected to be small, purely greedy methods provide almost optimal results.

We now present a few open questions/tracks for improvement.

- Even though we improved practically and theoretically the Patel-Markov-Hayes algorithm, the asymptotic complexity of GreedyGE remains a factor of 2 away from the theoretical lower bound. Can we find an algorithm that reaches this lower bound? If yes, we think it will not be with an algorithm like PMH or GreedyGE that blindly zeroes the entries of the matrix. Indeed, with such algorithms, the maximum number of entries we are ensured to zero with a row operation is $\log_2(n)$. However, in order to reach the theoretical lower bound of $\frac{n^2}{2\log_2(n)}$, each row operation needs to zero in average $2\log_2(n)$ entries. This cannot be achieved with simple Gaussian elimination based algorithms.

- The syndrome decoding based algorithm may be a good candidate to reach the theoretical lower bound, but we need tools from the cryptographic realm to establish tight worst-case complexities. This should be the subject of a future work. The practical complexity also depends on the algorithm used for solving the syndrome decoding problem. Can we find a better solver? The integer programming solver provides optimal results but is unusable for problems larger than 50 qubits. The greedy heuristics scale better and we manage to address larger problems with them but at the cost of a suboptimal solution. This sub-optimality may explain why the method does not provide good results for large n compared to PMH or GreedyGE. We know that the syndrome decoding problem, in its general form, is NP-Hard. Thus it is unlikely that we can fundamentally do better than a brute-force solver or a suboptimal greedy algorithm. Yet there is one feature of the problem that we did not exploit in our work and that may simplify the problem. This relies on a graph-oriented reformulation of our problem and we let its solving as a future work. When we scan a circuit to compute the available parities, each new parity is obtained because we apply a CNOT gate that sums two rows of the current linear reversible operator. So, when we gather the different parities in a matrix H, if the gathering is done chronologically, each new column of H can be written as a sum of two previous columns of H. We can create a parity graph where each node is a parity and, given three nodes v_1, v_2, v_3 corresponding to three parities p_1, p_2, p_3, we add two edges $v_1 \to v_3, v_2 \to v_3$ if $p_3 = p_1 \oplus p_2$. We call such a three-node structure a triangle. Each node except the first n ones have two such in-edges. There is no restriction on the number of out-edges though. A solution to the syndrome decoding problem is given by a subset of nodes: those nodes are considered active. Given such a subset, we can switch to another solution by considering one triangle and taking the complementary in terms of active/inactive nodes. We can then navigate in the space of all the solutions in search of the one with the fewest active nodes: this is the optimal solution of the syndrome decoding problem. One can show that we can always reach the optimal solution with only the elementary operation "take the complementary of a triangle" because whatever the set of active nodes, we can always reach a canonical form where the active nodes are solely among the first n nodes. The problem is illustrated on 4 qubits in Fig 13.1. Is there an efficient solution to this problem?

- Some quantum algorithms have been proposed for solving the syndrome decoding problem via the Information Set Decoding algorithm [20, 95, 100]. This gives the possibility of designing a hybrid quantum/classical compiler for this particular synthesis problem.

- Do we need the best solver possible? From the experiments we saw that the better the solver was, the better were the results. Nevertheless, this result is not obvious and raises a question: how far are we from the optimal circuit with an optimal solver of the syndrome decoding problem? When we optimally solve one instance of the syndrome decoding problem we solely compute an optimum of a subpart of the global synthesis problem. We can imagine for

instance "wasting" a CNOT to create a parity that will be reused for other syndrome decoding problems. Solving non optimally one instance of the syndrome decoding problem would, therefore, help to have smaller solutions of other larger instances of the syndrome decoding problem and eventually synthesize a shorter circuit. It would be interesting to develop a global optimization algorithm which would calculate the next CNOT not according to the cost of the current syndrome decoding problem but of all the syndrome decoding problems to follow. Such a method would certainly provide shorter circuits but the question is to know how good its scalability is.

- Another avenue for improvement would be to mix GreedyGE and the syndrome decoding based algorithm. When you apply GreedyGE on an operator A, first you triangularize A and then you synthesize an upper triangular operator U. During this process, once the first column is treated, there will be no more CNOT gates with control the first qubit during the triangularization but also during the synthesis of U. In other words, the first row of U can be ultimately synthesized with a specific instance of the syndrome decoding problem where you can apply any parity that appeared in $A[2:end, 2:end]$ after the first column has been treated during the first step of GreedyGE. Similarly the second row of U can be synthesized by solving a syndrome decoding problem with all the parities that appeared in $A[3:end, 3:end]$ after the treatment of the second column in GreedyGE's part. Compared to a standard syndrome decoding based algorithm, the number of parities collected is much larger because it includes everything that was done during the first part of the synthesis (the triangularization part). This increases the chances to have small solutions on the syndrome decoding part. On the other hand GreedyGE produces less optimal circuits for medium-sized registers, this introduces an overhead that may not be compensated by the gain in the syndrome decoding parts.

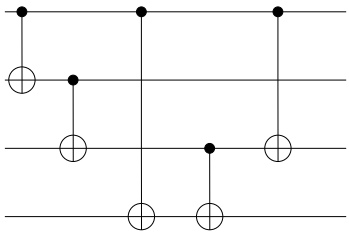
- Finally, purely greedy methods have shown to be quite effective especially to find short circuits. But ultimately the scalability of those methods are limited. Can we find better cost functions? A main problem for a good scalability is the presence of local minima. In our implementation we simply choose a random operation with minimum cost in hope of falling into another minimum. How can we deal with those minima more efficiently?

13.2 Depth optimization with full qubit connectivity

We looked for a practical implementation of the divide-and-conquer framework proposed in [90]. We have proposed a generalization of their method, by simply reformulating the synthesis of the operator as a succession of binary matrix zeroing problems with a precise set of elementary operations, namely row and column operations, and the bit flip of one entry. Several strategies can be employed to solve the zeroing problem. With a block strategy, we improved the worst case upper bound for practical values of n ($150 < n < 480000$). With a greedy approach, we were able to synthesize an n-qubit operator with depth between $0.85n$ and n. This is an improvement by a factor of 2 compared to the state-of-the-art method.

We now present a few open questions/tracks for improvement.

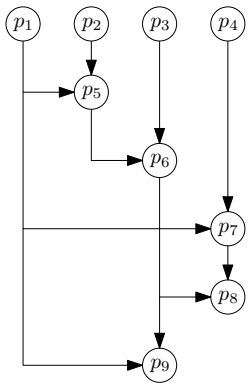
- It seems clear when looking at the benchmarks that the upper bound of our framework is to be improved. It is easy to have some results when we only consider the maximum number of ones in each row of $B = A_3 A_1^{-1} \in F_2^{n \times n}$, but combining this with a restriction on the columns makes the proofs harder. For instance it is easy to prove that one can flip no more



(a) CNOT circuit

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

(b) Parity table



(c) Parity graph

Figure 13.1: An example of a CNOT circuit, the parities that appear in it and the associated parity graph.

than $n/2$ entries on each row of B to get a matrix $B \oplus C$ with only two different row vectors. Such matrix can be zeroed in a logarithmic number of steps and this would give an upper bound of approximately $n+$logarithmic terms which is closer to our benchmarks. Yet we are unable to get any property on the maximum number of 1 in the columns of C to conclude.

- In Section 10.3 we showed that the implementation of an operator $A \in F_2^{p \times n}$ is barely more costly than the synthesis of a square operator of size n. We want to highlight that this important result can lead to new strategies for our initial divide-and-conquer framework and improve the theoretical upper bounds on the depth. Consider the matrix $B = A_3 A_1^{-1} \in F_2^{n \times n}$ to zero during step 2 of the framework. If B is of rank $k < n$ then we write

$$B = DF$$

where $D \in F_2^{n \times k}, F \in F_2^{k \times n}$ are both of rank k. By using the block extension algorithm we know that there is a sequence $(E_{i_D j_D})_{i_D j_D}$, resp. $(E_{i_F j_F})_{i_F j_F}$), of row, resp. column, operations of depth $\mathcal{O}(k)$ such that

$$\prod_{i_D, j_D} E_{i_D, j_D} D = \begin{pmatrix} I_r \\ 0 \end{pmatrix},$$

$$F \prod_{i_F, j_F} E_{i_F, j_F} = \begin{pmatrix} I_r & 0 \end{pmatrix}.$$

Equivalently

$$\prod_{i_D, j_D} E_{i_D, j_D} B \prod_{i_F, j_F} E_{i_F, j_F} = \begin{pmatrix} I_r & 0 \\ 0 & 0 \end{pmatrix}$$

and B can be zeroed with a sequence of operations of depth $\mathcal{O}(k)$. So instead of trying to minimize the number of ones in B, as we do, one might be interested in diminishing the rank. The problem can be formulated as: given an integer $r < n$ and $B \in F_2^{n\times n}$ of rank $k > r$, what is the sequence of operations (row operations, column operations, entry flips) of minimum depth that transform B into a matrix of rank r?

This problem is related to other problems in the literature. The matrix rigidity of a matrix A is defined as the minimum Hamming distance between A and a matrix of rank r. In other words the matrix rigidity of A is the minimum number of entries of A that must be modified in order for the rank to drop to r. In the literature the concept of matrix rigidity was used to prove lower bounds on the complexity of classical linear circuits [124, 191]. Most of the work we found on the subject was thus dedicated to finding explicit rigid matrices, which is quite the opposite of our approach. Moreover, the distance for the matrix rigidity problem is defined as the number of flips in A whereas we are concerned in the depth of a sequence of operations.

The problem of matrix rigidity can be extended with the more general problem of low rank approximations where we try to find, for given target rank r, the solution to

$$\min_{\text{rank}(R)=r} \|A - R\|$$

where $\|\cdot\|$ is an appropriate norm. Using the L_1 norm, we recover the problem of finding the matrix rigidity of A. But again none of the norms usually considered take into account the depth required to implement R. Lastly, such problems only consider one of the three operations that are available to us, namely the entry flips. Although row and column operations alone cannot reduce the rank of a matrix, they can help in creating a new matrix that needs less entries to flip to have a reduced rank.

- We did not treat the case where we have clean ancillae. With $n + p$ ancillae, standard techniques [90, 141] compute the two operations

$$|x\rangle\,|0\rangle\,|0\rangle^p \to |x\rangle\,|0 \oplus Ax\rangle\,|0\rangle^p = |x\rangle\,|Ax\rangle\,|0\rangle^p$$

$$|x\rangle\,|Ax\rangle\,|0\rangle^p \to |x \oplus A^{-1}Ax\rangle\,|Ax\rangle\,|0\rangle^p = |0\rangle\,|Ax\rangle\,|0\rangle^p$$

and swap the first two registers. It is easy to show that with kn additional clean ancillae we can synthesize independently blocks of $(n_i)_{i=1..k}$ columns, with $\sum_{i=1}^{k} n_i = n$ and add them together with a logarithmic overhead to create A. The total depth would be

$$d \approx 4(\max_{i=1..k}(d(n_i)) + 1 + \lceil \log_2(k) \rceil) \approx 4n/k$$

but this does not represent a clear improvement over [90]. They show that a circuit with $(3s + 1)n$ ancillae can be parallelized to $\mathcal{O}(\frac{n}{s \log_2(n)})$ and the overhead is not as problematic as in the non-ancillae case.

- Finally, can we extend this framework to take into account connectivity constraints? We believe it is not an easy task because it is not natural in a restricted connectivity to split the qubits into two sets, especially if there is no particular symmetry between the two sets. We think it is preferable to improve the results from [117] for the LNN architecture and extend it to an arbitrary topology.

13.3 Size optimization on constrained architectures

We extended the algorithm using the syndrome decoding problem to take into account the hardware constraints. The idea is to order the qubits according to a Hamiltonian path in the connectivity graph then to solve weighted instances of the syndrome decoding problem. Each parity is assigned a weight corresponding to the number of CNOTs needed to add the parity to the target qubit. The farther the qubit carrying the parity to add is in the hardware, the greater the weight of the parity will be. The experiments have shown that, except for the LNN architecture whose connectivity is too sparse, we consistently outperform the state-of-the-art algorithm based on Steiner trees by a percentage that increases with the number of qubits.

We now present a few open questions/tracks for improvement.

- We have shown that the choice of a "good" Hamiltonian path is given by the solution of an instance of the Minimum Linear Arrangement problem. We did not implement an algorithm for solving it yet. Instead we manually chose a suitable Hamiltonian path for each architecture we considered. Solving the MinLA problem may give better results, this will be the subject of future work.
- Can we extend efficiently our framework to architectures that do not have a Hamiltonian path? A straightforward extension would consist in considering paths with holes between the qubits. This would require to use more expensive CNOT as one of the target in the path has to be removed. Otherwise our algorithm can be used identically. Is there a more efficient way to do the extension?
- We expect our algorithm to give better and better results as the density of the connectivity graph increases. A detailed comparison against the Steiner tree based method with more and more connected architectures will be a subject of future work.

Part D

Conclusion and future research work

Summary

During this thesis we have broached the subject of the synthesis of quantum circuits, in a rather broad sense, by focusing on two precise abstract representations of quantum operators:

- via a unitary matrix in $\mathcal{SU}(2^n)$. Any operator has such a representation and moreover this representation is unique. The compilation of a unitary matrix into a circuit is historically what is meant by "synthesis of quantum circuits".
- via an invertible Boolean matrix in $\mathbb{F}_2$. This compact form is used to represent so-called reversible linear operators. Reversible linear operators represent a subclass of operators having multiple applications in quantum and in particular are at the center of current compilers.

During the first half of this manuscript (Part B) we have emphasized the interest of working on a joint optimization of classical and quantum resources in the synthesis process. If quantum resources are currently extremely limited, classical resources are also limiting the synthesis to a few qubits. We have considered two types of optimizations:

- First we have proposed a method of synthesis using a QR factorization algorithm based on Householder transformations. Our method is almost as fast as a matrix/matrix product which is one of the most optimized routines in scientific software libraries, and allows us to synthesize operators up to 15 qubits while the state-of-the-art algorithm, the Quantum Shannon Decomposition, is limited to 12 qubits.
- Then we have explored the use of numerical optimizers for the synthesis. For the majority of operators, representing the worst case, we obtained excellent results by succeeding in synthesizing operators and states with a number of entangling gates given by a theoretical lower bound. In other words, it is impossible to do better for the majority of operators and that gives theoretical indications on the validity of the theoretical lower bounds. It is also possible to increase the number of gates in the circuit to drastically reduce the number of iterations necessary for the convergence of the optimizer. Thus we can increase the problem size that can be treated by this method while producing smaller circuits than those synthesized by state of the art methods even though the circuits are not optimal anymore.

These two new synthesis methods can also be combined with methods from the literature. For example, the use of a numerical optimizer for small operators makes it possible to improve the size of the circuits produced by the QSD. We can thus propose a continuous landscape of methods according to the size of the problem. For each problem size, a synthesis method can be proposed which minimizes the number of gates in the circuit, knowing the quantity of classical resources available. Typically for $n > 12$ only the Householder method is viable but for $n = 10$ it makes more sense to combine the QSD with our optimizer by stopping the recursion of the QSD at $n = 4$ or $n = 5$. We hope that these methods will be useful for the execution of future quantum algorithms

based on the generation of circuits without particular structure, as one might encounter in the QRAM generation.

During the second half of the manuscript we studied the synthesis of linear reversible circuits by implementing them only with the CNOT gate. We were interested in the optimization of quantum resources only — the number of CNOTs or the depth of the CNOT circuit — and in two cases of architectures: an architecture where the qubits are fully connected and the current architectures where the qubits are linked only to their neighbors in the hardware. We have proposed new algorithms in the following three cases:

1. size optimization for a fully connected architecture,
2. depth optimization for a fully connected architecture,
3. size optimization for architectures with partial connectivities.

In each of the three cases, we have designed algorithms outperforming the state of the art, with various gains:

1. on average 25% with a maximum of 55% in case 1,
2. 50% gain on average on the depth in case 2,
3. finally between 10% and 20% gain, depending on the architecture, in case 3.

Linear reversible operators are used in the compilation of Clifford+T circuits, in particular in the synthesis of so-called CNOT+T circuits. So we replaced the methods of synthesis of the purely CNOT parts by our own to see the gains that this could bring on libraries of reversible functions. We have shown that our methods allow to considerably reduce the number of CNOTs while preserving another important metric: the T-depth. Some state-of-the-art algorithms manage to generate circuits with fewer CNOTs but at the cost of a much greater T-depth. We offer a possible tradeoff between these two metrics. The depth is also significantly reduced. These optimized circuits come closer and closer to circuits that can be reasonably performed on quantum computers to arise in the coming years.

Global perspectives

The current trend in the literature is to focus on subclasses of operators that can be expressed compactly, i.e., polynomially in the number of qubits. CNOT circuits, stabilizer circuits, CNOT+T circuits, oracles, Hamiltonians in quantum chemistry, all these operators can be described efficiently and the implementation of a circuit implementing them is the subject of continuous progress in the literature. It might be interesting to look at other forms of operators that can be easily simulated. For example, match gates are gates acting on 2 qubits of the form

$$G(A,B) = \begin{pmatrix} p & 0 & 0 & q \\ 0 & w & x & 0 \\ 0 & y & z & 0 \\ r & 0 & 0 & s \end{pmatrix} \quad A = \begin{pmatrix} p & q \\ r & s \end{pmatrix}, B = \begin{pmatrix} w & x \\ y & z \end{pmatrix}$$

where A, B are matrices in $\mathcal{SU}(2)$ with same determinant. A circuit made of $G(A,B)$ gates acting only on nearest neighbor lines can be efficiently simulated by a classical computer [93]. Thus such circuits are more likely to be efficiently optimized.

For generic circuits synthesis we solely focused on the unitary matrix as representation of a quantum operator. What about using a Hermitian matrix H such that $U = e^{iH}$? We enter the realm of Hamiltonian simulation, i.e., the simulation of a quantum system specified by its Hamiltonian, and we somehow go back to the born of the quantum computer with Richard Feynman and Yuri Manin evoking the idea of simulating a quantum system with a quantum machine itself. Can we transpose some or our methods to the Hamiltonian simulation? Some compilation algorithms have already been successfully transposed into the Hamiltonian simulation, take for instance the GraySynth algorithm [42]. There is a simple link between a unitary given as a product of Householder matrices and its associated Hamiltonian matrix. We plan to transpose the Householder framework to the case of the Hamiltonian simulation.

From a theoretical point of view, it would be very interesting to continue the study of Lie algebras and Lie groups. We now know that the QSD fits perfectly in a Lie algebraic framework. Can we find more efficient decompositions? The work in [144] provides a fairly general framework for finding new decompositions. These decompositions — also called KAK decompositions — make it possible to decompose a unitary on n qubits into 4 unitaries on n-1 qubits and 3 operators described by specific Hamiltonians: the link between unitary synthesis and Hamiltonian simulation is straightforward. The complexity of a KAK decomposition is given by the cost of synthesizing those three Hamiltonians. A good class of Hamiltonians would be a class easier to implement than the multiplexors that also fits into a Cartan decomposition. Does such a class exist?

Finally, some works during the second half of the 2000's have established a link between finding a quantum circuit and finding a path in a certain curved geometry [52, 147, 149]. Finding the optimal circuit implementing an operator is equivalent to solving a doubly exponentially hard problem [147]. More precisely, one has to solve an exponentially large instance of the closest vector in a lattice problem. Solving exactly this problem is hard, but what about approximating it?

To our knowledge, no practical use of these results has been undertaken. Can good heuristics be designed?

Bibliography

[1] Scott Aaronson. BQP and the polynomial hierarchy. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 141–150. ACM, 2010.

[2] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5):052328, 2004.

[3] Alfred V Aho and Krysta M Svore. Compiling quantum circuits using the palindrome transform. *arXiv preprint quant-ph/0311008*, 2003.

[4] Zeyad Abd Algfoor, Mohd Shahrizal Sunar, and Hoshang Kolivand. A comprehensive study on pathfinding techniques for robotics and video games. *Int. J. Comput. Games Technol.*, 2015:736138:1–736138:11, 2015.

[5] C. Allouche, M. Baboulin, T. Goubault de Brugière, and B. Valiron. Reuse method for quantum circuit synthesis. *Recent Advances in Mathematical and Statistical Methods*, 259:3–12, 2018.

[6] Noga Alon, Mauricio Karchmer, and Avi Wigderson. Linear circuits over $\mathrm{GF}(2)$. *SIAM J. Comput.*, 19(6):1064–1067, 1990.

[7] Daniel Alsina and José Ignacio Latorre. Experimental test of Mermin inequalities on a five-qubit quantum computer. *Phys. Rev. A*, 94:012314, Jul 2016.

[8] Matthew Amy. Matthew Amy's Github. https://github.com/meamy.

[9] Matthew Amy, Parsiad Azimzadeh, and Michele Mosca. On the controlled-NOT complexity of controlled-NOT-phase circuits. *Quantum Science and Technology*, 4(1):015002, 2018.

[10] Matthew Amy, Dmitri Maslov, and Michele Mosca. Polynomial-time T-depth optimization of Clifford+T circuits via matroid partitioning. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 33(10):1476–1489, 2014.

[11] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin 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(6):818–830, 2013.

[12] Matthew Amy and Michele Mosca. T-count optimization and Reed-Muller codes. *IEEE Trans. Inf. Theory*, 65(8):4771–4784, 2019.

[13] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, 3 edition, 1999.

[14] Sanjeev Arora, László Babai, Jacques Stern, and Z Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. *Journal of Computer and System Sciences*, 54(2):317–331, 1997.

[15] Juan Miguel Arrazola, Thomas R Bromley, Josh Izaac, Casey R Myers, Kamil Bradler, and Nathan Killoran. Machine learning method for state preparation and gate synthesis on photonic quantum computers. *Quantum Science and Technology*, 2018.

[16] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[17] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Phys. Rev. A*, 52:3457–3467, Nov 1995.

[18] Ville Bergholm, Juha J. Vartiainen, Mikko Möttönen, and Martti M. Salomaa. Quantum circuits with uniformly controlled one-qubit gates. *Phys. Rev. A*, 71:052330, May 2005.

[19] Elwyn Berlekamp, Robert McEliece, and Henk Van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[20] Daniel J. Bernstein. Grover vs. McEliece. In *International Workshop on Post-Quantum Cryptography*, pages 73–80. Springer, 2010.

[21] Ethan Bernstein and Umesh V. Vazirani. Quantum complexity theory. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 11–20. ACM, 1993.

[22] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[23] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

[24] Susan Blackford. Benchmark LAPACK. http://www.netlib.org/lapack/lug/node71.html.

[25] Alex Bocharov, Martin Roetteler, and Krysta M Svore. Efficient synthesis of probabilistic quantum circuits with fallback. *Physical Review A*, 91(5):052317, 2015.

[26] Alex Bocharov, Martin Roetteler, and Krysta M Svore. Efficient synthesis of universal repeat-until-success quantum circuits. *Physical review letters*, 114(8):080502, 2015.

[27] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artif. Intell.*, 129(1-2):5–33, 2001.

[28] Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to cryptology. *J. Cryptology*, 26(2):280–312, 2013.

[29] Sergey Bravyi and Jeongwan Haah. Magic-state distillation with low overhead. *Physical Review A*, 86(5):052329, 2012.

[30] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Physical Review A*, 71(2):022316, 2005.

[31] Sergey Bravyi and Dmitri Maslov. Hadamard-free circuits expose the structure of the Clifford group. *arXiv preprint arXiv:2003.09412*, 2020.

[32] Michael J. Bremner, Christopher M. Dawson, Jennifer L. Dodd, Alexei Gilchrist, Aram W. Harrow, Duncan Mortimer, Michael A. Nielsen, and Tobias J. Osborne. Practical scheme for quantum computation with any two-qubit entangling gate. *Phys. Rev. Lett.*, 89:247902, Nov 2002.

[33] Hans J Briegel, David E Browne, Wolfgang Dür, Robert Raussendorf, and Maarten Van den Nest. Measurement-based quantum computation. *Nature Physics*, 5(1):19–26, 2009.

[34] Stephen S. Bullock. Note on the Khaneja Glaser decomposition. *Quantum Information & Computation*, 4(5):396–400, 2004.

[35] Stephen S. Bullock and Igor L. Markov. An arbitrary two-qubit computation in 23 elementary gates or less. In *Proceedings of the 40th Annual Design Automation Conference*, DAC '03, pages 324–329, New York, NY, USA, 2003. ACM.

[36] Stephen S. Bullock and Igor L. Markov. Asymptotically optimal circuits for arbitrary n-qubit diagonal computations. *Quantum Info. Comput.*, 4(1):27–47, January 2004.

[37] Renan Cabrera, Traci Strohecker, and Herschel Rabitz. The canonical coset decomposition of unitary matrices through Householder transformations. *Journal of Mathematical Physics*, 51(8):082101, 2010.

[38] A Robert Calderbank, Eric M Rains, Peter W Shor, and Neil JA Sloane. Quantum error correction and orthogonal geometry. *Physical Review Letters*, 78(3):405, 1997.

[39] Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):339–354, 1998.

[40] Don Coppersmith. An approximate Fourier transform useful in quantum factoring. *arXiv preprint quant-ph/0201067*, 2002.

[41] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. On the qubit routing problem. In Wim van Dam and Laura Mancinska, editors, *14th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2019, June 3-5, 2019, University of Maryland, College Park, Maryland, USA*, volume 135 of *LIPIcs*, pages 5:1–5:32. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[42] Alexander Cowtan, Will Simmons, and Ross Duncan. A generic compilation strategy for the unitary coupled cluster ansatz. *arXiv preprint arXiv:2007.10515*, 2020.

[43] George Cybenko. Reducing quantum computations to elementary unitary operations. *Computing in Science & Engineering*, 3(2):27–32, 2001.

[44] Mehmet Dağlı, Domenico D'Alessandro, and Jonathan DH Smith. A general framework for recursive decompositions of unitary quantum evolutions. *Journal of Physics A: Mathematical and Theoretical*, 41(15):155302, 2008.

[45] Christopher M. Dawson and Michael A. Nielsen. The Solovay-Kitaev algorithm. *Quantum Info. Comput.*, 6(1):81–95, January 2006.

[46] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, and Cyril Allouche. Synthesizing quantum circuits via numerical optimization. In João M. F. Rodrigues, Pedro J. S. Cardoso, Jânio M. Monteiro, Roberto Lam, Valeria V. Krzhizhanovskaya, Michael Harold Lees, Jack J. Dongarra, and Peter M. A. Sloot, editors, *Computational Science - ICCS 2019 - 19th International Conference, Faro, Portugal, June 12-14, 2019, Proceedings, Part II*, volume 11537 of *Lecture Notes in Computer Science*, pages 3–16. Springer, 2019.

[47] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, and Cyril Allouche. Quantum circuits synthesis using Householder transformations. *Comput. Phys. Commun.*, 248:107001, 2020.

[48] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. Quantum CNOT circuits synthesis for NISQ architectures using the syndrome decoding problem. In Ivan Lanese and Mariusz Rawski, editors, *Reversible Computation - 12th International Conference, RC 2020, Oslo, Norway, July 9-10, 2020, Proceedings*, volume 12227 of *Lecture Notes in Computer Science*, pages 189–205. Springer, 2020.

[49] Arianne Meijer-van de Griend and Ross Duncan. Architecture-aware synthesis of phase polynomials for NISQ devices. *arXiv preprint arXiv:2004.06052*, 2020.

[50] A. De Vos and S. De Baerdemacker. Block-ZXZ synthesis of an arbitrary quantum circuit. *Phys. Rev. A*, 94:052317, Nov 2016.

[51] J. Dongarra. Basic Linear Algebra Subprograms Technical Forum Standard. *Int. J. of High Performance Computing Applications*, 16(1), 2002.

[52] Mark R Dowling and Michael A Nielsen. The geometry of quantum computation. *arXiv preprint quant-ph/0701004*, 2006.

[53] Byron Drury and Peter Love. Constructive quantum Shannon decomposition from Cartan involutions. *Journal of Physics A: Mathematical and Theoretical*, 41(39):395305, 2008.

[54] Henrique N Sá Earp and Jiannis K Pachos. A constructive algorithm for the Cartan decomposition of $\mathrm{SU}(2^N)$. *Journal of mathematical physics*, 46(8):082108, 2005.

[55] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[56] Heng Fan, Vwani Roychowdhury, and Thomas Szkopek. Optimal two-qubit quantum circuits using exchange interactions. *Physical Review A*, 72(5):052323, 2005.

[57] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[58] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106*, 2000.

[59] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6-7):467–488, 1982.

[60] Claus Fieker, William Hart, Tommy Hofmann, and Fredrik Johansson. Nemo/hecke: Computer algebra and number theory packages for the Julia programming language. In *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '17, pages 157–164, New York, NY, USA, 2017. ACM.

[61] Austin G. Fowler. Constructing arbitrary Steane code single logical qubit fault-tolerant gates. *Quantum Information & Computation*, 11(9&10):867–873, 2011.

[62] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[63] Finley Freibert. The classification of complementary information set codes of lengths 14 and 16. *Adv. in Math. of Comm.*, 7(3):267–278, 2013.

[64] Hartmut Führ and Ziemowit Rzeszotnik. On biunimodular vectors for unitary matrices. *Linear Algebra and its Applications*, 484:86–129, 2015.

[65] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co, 1979.

[66] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1996. Third edition.

[67] Oleg Golubitsky and Dmitri Maslov. A study of optimal 4-bit reversible Toffoli circuits and their synthesis. *IEEE Trans. Computers*, 61(9):1341–1353, 2012.

[68] Daniel Gottesman. *Stabilizer codes and quantum error correction*. PhD thesis, Caltech, 1997.

[69] Daniel Gottesman. The Heisenberg representation of quantum computers. 1998.

[70] Lov Grover and Terry Rudolph. Creating superpositions that correspond to efficiently integrable probability distributions. *arXiv preprint quant-ph/0208112*, 2002.

[71] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. ACM.

[72] Victor Guillemin and Alan Pollack. *Differential topology*, volume 370. American Mathematical Soc., 2010.

[73] Alfred Haar. Der massbegriff in der theorie der kontinuierlichen gruppen. *Annals of mathematics*, pages 147–169, 1933.

[74] Hartmut Häffner, Christian F Roos, and Rainer Blatt. Quantum computing with trapped ions. *Physics reports*, 469(4):155–203, 2008.

[75] David Hanneke, Jonathan P Home, John D Jost, Jason M Amini, Dietrich Leibfried, and David J Wineland. Realization of a programmable two-qubit quantum processor. *Nature Physics*, 6(1):13–16, 2010.

[76] Eric A. Hansen and Rong Zhou. Anytime heuristic search. *J. Artif. Intell. Res.*, 28:267–297, 2007.

[77] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103:150502, 2009.

[78] Yong He, Ming-Xing Luo, E Zhang, Hong-Ke Wang, and Xiao-Feng Wang. Decompositions of n-qubit Toffoli gates with linear circuit complexity. *International Journal of Theoretical Physics*, 56(7):2350–2361, 2017.

[79] Loic Henriet, Lucas Beguin, Adrien Signoles, Thierry Lahaye, Antoine Browaeys, Georges-Olivier Reymond, and Christophe Jurczak. Quantum computing with neutral atoms. *arXiv preprint arXiv:2006.12326*, 2020.

[80] Luke E Heyfron and Earl T Campbell. An efficient quantum compiler that reduces T count. *Quantum Science and Technology*, 4(1):015004, 2019.

[81] Peter Høyer and Robert Spalek. Quantum fan-out is powerful. *Theory Comput.*, 1(1):81–103, 2005.

[82] Martin Idel and Michael M Wolf. Sinkhorn normal form for unitary matrices. *Linear Algebra and its Applications*, 471:76–84, 2015.

[83] Takahiro Ikeda and Hiroshi Imai. Enhanced A* algorithms for multiple alignments: Optimal alignments for several sequences and k-opt approximate alignments for large cases. *Theor. Comput. Sci.*, 210(2):341–374, 1999.

[84] Intel. Math Kernel Library (MKL). http://www.intel.com/software/products/mkl/.

[85] Raban Iten, Roger Colbeck, and Matthias Christandl. Quantum circuits for quantum channels. *Physical Review A*, 95(5):052316, 2017.

[86] Raban Iten, Roger Colbeck, Ivan Kukuljan, Jonathan Home, and Matthias Christandl. Quantum circuits for isometries. *Phys. Rev. A*, 93:032318, Mar 2016.

[87] Raban Iten, Oliver Reardon-Smith, Luca Mondada, Ethan Redmond, Ravjot Singh Kohli, and Roger Colbeck. Introduction to UniversalQCompiler. *CoRR*, abs/1904.01072, 2019.

[88] P. A. Ivanov, E. S. Kyoseva, and N. V. Vitanov. Engineering of arbitrary $\mathrm{U}(N)$ transformations by quantum Householder reflections. *Phys. Rev. A*, 74:022323, Aug 2006.

[89] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. ScaffCC: Scalable compilation and analysis of quantum programs. *Parallel Comput.*, 45:2–17, 2015.

[90] Jiaqing Jiang, Xiaoming Sun, Shang-Hua Teng, Bujiao Wu, Kewen Wu, and Jialin Zhang. Optimal space-depth trade-off of CNOT circuits in quantum logic synthesis. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 213–229. SIAM, 2020.

[91] Cody Jones. Low-overhead constructions for the fault-tolerant Toffoli gate. *Physical Review A*, 87(2):022328, 2013.

[92] Richard Jozsa and Noah Linden. On the role of entanglement in quantum-computational speed-up. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 459(2036):2011–2032, 2003.

[93] Richard Jozsa and Akimasa Miyake. Matchgates and classical simulation of quantum circuits. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 464(2100):3089–3106, 2008.

[94] Stasys Jukna and Igor Sergeev. Complexity of linear Boolean operators. *Foundations and Trends in Theoretical Computer Science*, 9(1):1–123, 2013.

[95] Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. In *International Workshop on Post-Quantum Cryptography*, pages 69–89. Springer, 2017.

[96] Ajai Kapoor and Romeo Rizzi. Edge-coloring bipartite graphs. *Journal of Algorithms*, 34(2):390–396, 2000.

[97] Phillip Kaye and Michele Mosca. Quantum networks for generating arbitrary quantum states. *arXiv preprint quant-ph/0407102*, 2004.

[98] Navin Khaneja and Steffen J Glaser. Cartan decomposition of $\mathrm{S}U(2^n)$ and control of spin systems. *Chemical Physics*, 267(1):11–23, 2001.

[99] Sumeet Khatri, Ryan LaRose, Alexander Poremba, Lukasz Cincio, Andrew T Sornborger, and Patrick J Coles. Quantum-assisted quantum compiling. *Quantum*, 3:140, 2019.

[100] Elena Kirshanova. Improved quantum information set decoding. In *International Conference on Post-Quantum Cryptography*, pages 507–527. Springer, 2018.

[101] Aleks Kissinger. PyZX. https://github.com/Quantomatic/pyzx.

[102] Aleks Kissinger and Arianne Meijer-van de Griend. CNOT circuit extraction for topologically-constrained quantum memories. *arXiv preprint arXiv:1904.00633*, 2019.

[103] Aleksei Yur'evich Kitaev. Quantum computations: algorithms and error correction. *Uspekhi Matematicheskikh Nauk*, 52(6):53–112, 1997.

[104] Andreas Klappenecker and Martin Rötteler. Quantum software reusability. *International Journal of Foundations of Computer Science*, 14(05):777–796, 2003.

[105] Vadym Kliuchnikov, Alex Bocharov, and Krysta M Svore. Asymptotically optimal topological quantum compiling. *Physical review letters*, 112(14):140504, 2014.

[106] Vadym Kliuchnikov and Dmitri Maslov. Optimization of Clifford circuits. *Physical Review A*, 88(5):052307, 2013.

[107] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Asymptotically optimal approximation of single qubit unitaries by Clifford and T circuits using a constant number of ancillary qubits. *CoRR*, abs/1212.0822, 2012.

[108] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Fast and efficient exact synthesis of single-qubit unitaries generated by Clifford and T gates. *Quantum Information & Computation*, 13(7-8):607–630, 2013.

[109] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Practical approximation of single-qubit unitaries by single-qubit quantum Clifford and T circuits. *IEEE Trans. Computers*, 65(1):161–172, 2016.

[110] Emanuel Knill. Approximation by quantum circuits. *arXiv preprint quant-ph/9508006*, 1995.

[111] Emanuel Knill and Raymond Laflamme. Theory of quantum error-correcting codes. *Physical Review A*, 55(2):900, 1997.

[112] Pieter Kok, W. J. Munro, Kae Nemoto, T. C. Ralph, Jonathan P. Dowling, and G. J. Milburn. Linear optical quantum computing with photonic qubits. *Reviews of Modern Physics*, 79:135–174, Jan 2007.

[113] Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[114] Richard E Korf. Real-time heuristic search. *Artificial intelligence*, 42(2-3):189–211, 1990.

[115] Richard E Korf. *Artificial intelligence search algorithms*. 1996.

[116] B. Kraus and J. I. Cirac. Optimal creation of entanglement using a two-qubit gate. *Phys. Rev. A*, 63:062309, May 2001.

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

[118] R. B. Lehoucq. The computation of elementary unitary matrices. *ACM Trans. Math. Softw.*, 22(4):393–400, December 1996.

[119] Harry Levine, Alexander Keesling, Giulia Semeghini, Ahmed Omran, Tout T Wang, Sepehr Ebadi, Hannes Bernien, Markus Greiner, Vladan Vuletić, Hannes Pichler, et al. Parallel implementation of high-fidelity multiqubit gates with neutral atoms. *Physical review letters*, 123(17):170503, 2019.

[120] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for NISQ-era quantum devices. In Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck, editors, *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, pages 1001–1014. ACM, 2019.

[121] Rui Li, Unai Alvarez-Rodriguez, Lucas Lamata, and Enrique Solano. Approximate quantum adders with genetic algorithms: an IBM quantum experience. *Quantum Measurements and Quantum Metrology*, 4(1):1–7, 2017.

[122] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in neural information processing systems*, pages 767–774, 2004.

[123] Seth Lloyd. Universal quantum simulators. *Science*, pages 1073–1078, 1996.

[124] Satyanarayana V Lokam et al. Complexity lower bounds using linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 4(1–2):1–155, 2009.

[125] Martin Lukac and Georgiy Krylov. Study of GPU acceleration in genetic algorithms for quantum circuit synthesis. In *47th IEEE International Symposium on Multiple-Valued Logic, ISMVL 2017, Novi Sad, Serbia, May 22-24, 2017*, pages 213–218. IEEE Computer Society, 2017.

[126] Martin Lukac, Marek A. Perkowski, Hilton Goi, Mikhail Pivtoraiko, Chung Hyo Yu, Kyusik Chung, Hyunkoo Jeech, Byung-Guk Kim, and Yong Duk Kim. Evolutionary approach to quantum and reversible circuits synthesis. *Artif. Intell. Rev.*, 20(3-4):361–417, 2003.

[127] Emanuel Malvetti, Raban Iten, and Roger Colbeck. Quantum circuits for sparse isometries. *arXiv preprint arXiv:2006.00016*, 2020.

[128] Yuri Manin. Computable and uncomputable. *Sovetskoye Radio, Moscow*, 128, 1980.

[129] Esteban A Martinez, Thomas Monz, Daniel Nigg, Philipp Schindler, and Rainer Blatt. Compiling quantum algorithms for architectures with multi-qubit gates. *New Journal of Physics*, 18(6):063029, 2016.

[130] Dmitri Maslov. Linear depth stabilizer and quantum Fourier transformation circuits with no auxiliary qubits in finite-neighbor quantum architectures. *Physical Review A*, 76(5):052310, 2007.

[131] Dmitri Maslov. Advantages of using relative-phase Toffoli gates with an application to multiple control Toffoli optimization. *Physical Review A*, 93(2):022311, 2016.

[132] Dmitri Maslov. Optimal and asymptotically optimal NCT reversible circuits by the gate types. *Quantum Information & Computation*, 16(13&14):1096–1112, 2016.

[133] Dmitri Maslov. Basic circuit compilation techniques for an ion-trap quantum machine. *New Journal of Physics*, 19(2):023035, 2017.

[134] Dmitri Maslov and Martin Roetteler. Shorter stabilizer circuits via Bruhat decomposition and quantum circuit transformations. *IEEE Trans. Inf. Theory*, 64(7):4729–4738, 2018.

[135] Olivia Di Matteo and Michele Mosca. Parallelizing quantum circuit synthesis. *Quantum Science and Technology*, 1(1):015003, 2016.

[136] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, {II}. *Journal of Symbolic Computation*, 60(0):94 – 112, 2014.

[137] Giulia Meuli, Mathias Soeken, Earl Campbell, Martin Roetteler, and Giovanni De Micheli. The role of multiplicative complexity in compiling low t-count oracle circuits. In David Z. Pan, editor, *Proceedings of the International Conference on Computer-Aided Design, ICCAD 2019, Westminster, CO, USA, November 4-7, 2019*, pages 1–8. ACM, 2019.

[138] Giulia Meuli, Mathias Soeken, and Giovanni De Micheli. SAT-based {CNOT, T} quantum circuit synthesis. In *International Conference on Reversible Computation*, pages 175–188. Springer, 2018.

[139] Giulia Meuli, Mathias Soeken, Martin Roetteler, and Giovanni De Micheli. ROS: resource-constrained oracle synthesis for quantum computers. *CoRR*, abs/2005.00211, 2020.

[140] Nikolaj Moll, Panagiotis Barkoutsos, Lev S Bishop, Jerry M Chow, Andrew Cross, Daniel J Egger, Stefan Filipp, Andreas Fuhrer, Jay M Gambetta, Marc Ganzhorn, et al. Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology*, 3(3):030503, 2018.

[141] Cristopher Moore and Martin Nilsson. Parallel quantum computation and quantum codes. *SIAM J. Comput.*, 31(3):799–815, 2001.

[142] M Mottonen and Juha J Vartiainen. Decompositions of general quantum gates. *arXiv preprint quant-ph/0504100*, 2005.

[143] Mikko Möttönen, Juha J. Vartiainen, Ville Bergholm, and Martti M. Salomaa. Quantum circuits for general multiqubit gates. *Phys. Rev. Lett.*, 93:130502, Sep 2004.

[144] Yumi Nakajima, Yasuhito Kawano, and Hiroshi Sekigawa. A new algorithm for producing quantum circuits using KAK decompositions. *Quantum Information & Computation*, 6(1):67–80, 2006.

[145] Yunseong Nam, Neil J Ross, Yuan Su, Andrew M Childs, and Dmitri Maslov. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information*, 4(1):23, 2018.

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

[147] Michael A Nielsen. A geometric approach to quantum circuit lower bounds. *arXiv preprint quant-ph/0502070*, 2005.

[148] Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2011.

[149] Michael A Nielsen, Mark R Dowling, Mile Gu, and Andrew C Doherty. Quantum computation as geometry. *Science*, 311(5764):1133–1135, 2006.

[150] Joe O'Gorman and Earl T Campbell. Quantum computation with realistic magic-state factories. *Physical Review A*, 95(3):032338, 2017.

[151] Peter JJ O'Malley, Ryan Babbush, Ian D Kivlichan, Jonathan Romero, Jarrod R McClean, Rami Barends, Julian Kelly, Pedram Roushan, Andrew Tranter, Nan Ding, et al. Scalable quantum simulation of molecular energies. *Physical Review X*, 6(3):031007, 2016.

[152] Ketan N Patel, Igor L Markov, and John P Hayes. Optimal synthesis of linear reversible circuits. *Quantum Information & Computation*, 8(3):282–294, 2008.

[153] Anirban Pathak. *Elements of quantum computation and quantum communication*. Taylor & Francis, 2013.

[154] Massoud Pedram and Alireza Shafaei. Layout optimization for quantum circuits with linear nearest neighbor architectures. *IEEE Circuits and Systems Magazine*, 16(2):62–74, 2016.

[155] Martin Plesch and Časlav Brukner. Quantum-state preparation with universal gate decompositions. *Phys. Rev. A*, 83:032302, Mar 2011.

[156] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[157] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018.

[158] Python. Scientific Computing Tools for Python (SciPy). https://www.scipy.org.

[159] Xiaogang Qiang, Xiaoqi Zhou, Jianwei Wang, Callum M Wilkes, Thomas Loke, Sean O'Gara, Laurent Kling, Graham D Marshall, Raffaele Santagati, Timothy C Ralph, et al. Large-scale silicon quantum photonics implementing arbitrary two-qubit processing. *Nature photonics*, 12(9):534–539, 2018.

[160] Robert Raussendorf, Daniel E Browne, and Hans J Briegel. Measurement-based quantum computation on cluster states. *Physical review A*, 68(2):022312, 2003.

[161] Michael Reck, Anton Zeilinger, Herbert J. Bernstein, and Philip Bertani. Experimental realization of any discrete unitary operator. *Phys. Rev. Lett.*, 73:58–61, Jul 1994.

[162] Chad Rigetti, Jay M Gambetta, Stefano Poletto, BLT Plourde, Jerry M Chow, AD Córcoles, John A Smolin, Seth T Merkel, JR Rozen, George A Keefe, et al. Superconducting qubit in a waveguide cavity with a coherence time approaching 0.1 ms. *Physical Review B*, 86(10):100506, 2012.

[163] Neil J. Ross. Optimal ancilla-free Clifford+v approximation of z-rotations. *Quantum Information & Computation*, 15(11&12):932–950, 2015.

[164] Neil J. Ross and Peter Selinger. Optimal ancilla-free Clifford+T approximation of z-rotations. *Quantum Information & Computation*, 16(11&12):901–953, 2016.

[165] Mehdi Saeedi, Mona Arabzadeh, Morteza Saheb Zamani, and Mehdi Sedighi. Block-based quantum-logic synthesis. *Quantum Information & Computation*, 11(3&4):262–277, 2011.

[166] Mehdi Saeedi and Igor L Markov. Synthesis and optimization of reversible circuits—a survey. *ACM Computing Surveys (CSUR)*, 45(2):1–34, 2013.

[167] Mehdi Saeedi and Massoud Pedram. Linear-depth quantum circuits for n-qubit Toffoli gates with no ancilla. *Physical Review A*, 87(6):062318, 2013.

[168] Mark Saffman, Thad G Walker, and Klaus Mølmer. Quantum information with Rydberg atoms. *Reviews of modern physics*, 82(3):2313, 2010.

[169] Ben Schaeffer and Marek Perkowski. A cost minimization approach to synthesis of linear reversible circuits. *arXiv preprint arXiv:1407.0070*, 2014.

[170] Philipp Schindler, Daniel Nigg, Thomas Monz, Julio T Barreiro, Esteban Martinez, Shannon X Wang, Stephan Quint, Matthias F Brandl, Volckmar Nebendahl, Christian F Roos, Michael Chwalla, Markus Hennrich, and Rainer Blatt. A quantum information processor with trapped ions. *New Journal of Physics*, 15(12):123012, 2013.

[171] Robert Schreiber and Charles Van Loan. A storage-efficient WY representation for products of Householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 10(1):53–57, 1989.

[172] Peter Selinger. Quantum circuits of T-depth one. *Physical Review A*, 87(4):042302, 2013.

[173] Peter Selinger. Efficient Clifford+T approximation of single-qubit operators. *Quantum Information & Computation*, 15(1-2):159–180, 2015.

[174] V. V. Shende, S. S. Bullock, and I. L. Markov. Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1000–1010, 2006.

[175] Vivek V. Shende, Stephen S. Bullock, and Igor L. Markov. Synthesis of quantum logic circuits. In Tingao Tang, editor, *Proceedings of the 2005 Conference on Asia South Pacific Design Automation, ASP-DAC 2005, Shanghai, China, January 18-21, 2005*, pages 272–275. ACM Press, 2005.

[176] Vivek V Shende, Igor L Markov, and Stephen S Bullock. Minimal universal two-qubit controlled-NOT-based circuits. *Physical Review A*, 69(6):062321, 2004.

[177] Vivek V. Shende, Aditya K. Prasad, Igor L. Markov, and John P. Hayes. Synthesis of reversible logic circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(6):710–722, 2003.

[178] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE Computer Society, 1994.

[179] Robert S. Smith, Michael J. Curtis, and William J. Zeng. A practical quantum instruction set architecture. *CoRR*, abs/1608.03355, 2016.

[180] Andrei N Soklakov and Rüdiger Schack. Efficient state preparation for a register of quantum bits. *Physical review A*, 73(1):012307, 2006.

[181] Andrew M Steane. Error correcting codes in quantum theory. *Physical Review Letters*, 77(5):793, 1996.

[182] Andrew M Steane. Simple quantum error-correcting codes. *Physical Review A*, 54(6):4741, 1996.

[183] G. Stewart. The efficient generation of random orthogonal matrices with an application to condition estimators. *SIAM Journal on Numerical Analysis*, 17(3):403–409, 1980.

[184] Xiaobai Sun and Christian Bischof. A basis-kernel representation of orthogonal matrices. *SIAM Journal on Matrix Analysis and Applications*, 16(4):1184–1196, 1995.

[185] Brian D. Sutton. Computing the complete CS decomposition. *Numerical Algorithms*, 50(1):33–65, Jan 2009.

[186] Krysta M. Svore and Matthias Troyer. The quantum future of computation. *IEEE Computer*, 49(9):21–30, 2016.

[187] Stanimire Tomov, Jack J. Dongarra, and Marc Baboulin. Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Comput.*, 36(5-6):232–240, 2010.

[188] Robert R Tucci. A rudimentary quantum compiler (2cnd ed.). *arXiv preprint quant-ph/9902062*, 1999.

[189] Robert R Tucci. An introduction to Cartan's KAK decomposition for QC programmers. *arXiv preprint quant-ph/0507171*, 2005.

[190] Jesús Urías and Diego A. Quiñones. Householder methods for quantum circuit design. *Canadian Journal of Physics*, 94(2):150–157, 2016.

[191] Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In Jozef Gruska, editor, *Mathematical Foundations of Computer Science 1977, 6th Symposium, Tatranska Lomnica, Czechoslovakia, September 5-9, 1977, Proceedings*, volume 53 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 1977.

[192] Benoît Valiron, Neil J. Ross, Peter Selinger, D. Scott Alexander, and Jonathan M. Smith. Programming the quantum future. *Commun. ACM*, 58(8):52–61, 2015.

[193] Alexander Vardy. Algorithmic complexity in coding theory and the minimum distance problem. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 92–109. ACM, 1997.

[194] Juha J. Vartiainen, Mikko Möttönen, and Martti M. Salomaa. Efficient decomposition of quantum gates. *Phys. Rev. Lett.*, 92:177902, Apr 2004.

[195] Farrokh Vatan and Colin Williams. Optimal quantum circuits for general two-qubit gates. *Phys. Rev. A*, 69:032315, Mar 2004.

[196] Farrokh Vatan and Colin P Williams. Realization of a general three-qubit quantum gate. *arXiv preprint quant-ph/0401178*, 2004.

[197] G. Vidal and C. M. Dawson. Universal quantum circuit for two-qubit transformations with three controlled-NOT gates. *Phys. Rev. A*, 69:010301, Jan 2004.

[198] Guifré Vidal. Efficient classical simulation of slightly entangled quantum computations. *Phys. Rev. Lett.*, 91:147902, Oct 2003.

[199] Joseph L Walsh. A closed set of normal orthogonal functions. *American Journal of Mathematics*, 45(1):5–24, 1923.

[200] Jingbo Wang and Kia Manouchehri. *Physical implementation of quantum walks*. Springer, 2013.

[201] JONATHAN WELCH. Efficient approximation of diagonal unitaries over the Clifford+ T basis. *Quantum Information and Computation*, 16(1&2):0087–0104, 2016.

[202] Jonathan Welch, Daniel Greenbaum, Sarah Mostame, and Alan Aspuru-Guzik. Efficient quantum circuits for diagonal unitaries without ancillas. *New Journal of Physics*, 16(3):033040, 2014.

[203] Jonathan M Welch. *On the Synthesis of Quantum Circuits for Diagonal Operators in Quantum Computation*. Doctoral dissertation, Harvard University, Graduate School of Arts & Sciences, 2015.

[204] Robert Wille, Oliver Keszöcze, Marcel Walter, Patrick Rohrs, Anupam Chattopadhyay, and Rolf Drechsler. Look-ahead schemes for nearest neighbor optimization of 1D and 2D quantum circuits. In *21st Asia and South Pacific Design Automation Conference, ASP-DAC 2016, Macao, Macao, January 25-28, 2016*, pages 292–297. IEEE, 2016.

[205] Christopher Wilt and Wheeler Ruml. Effective heuristics for suboptimal best-first search. *Journal of Artificial Intelligence Research*, 57:273–306, 2016.

[206] Christopher Makoto Wilt, Jordan Tyler Thayer, and Wheeler Ruml. A comparison of greedy search algorithms. In *third annual symposium on combinatorial search*, 2010.

[207] William K Wootters and Wojciech H Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982.

[208] Stephen Wright and Jorge Nocedal. Numerical optimization. *Springer Science*, 35(67-68):7, 1999.

[209] Bujiao Wu, Xiaoyu He, Shuai Yang, Lifu Shou, Guojing Tian, Jialin Zhang, and Xiaoming Sun. Optimization of CNOT circuits on topological superconducting processors. *arXiv preprint arXiv:1910.14478*, 2019.

[210] A Chi-Chih Yao. Quantum circuit complexity. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science (FOCS 1993)*, pages 352–361. IEEE, 1993.

[211] Takayuki Yoshizumi, Teruhisa Miura, and Toru Ishida. A* with partial expansion for large branching factor problems. In Henry A. Kautz and Bruce W. Porter, editors, *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA*, pages 923–929. AAAI Press / The MIT Press, 2000.

[212] Jun Zhang, Jiri Vala, Shankar Sastry, and K Birgitta Whaley. Geometric theory of nonlocal two-qubit operations. *Physical Review A*, 67(4):042313, 2003.

[213] Jun Zhang, Jiri Vala, Shankar Sastry, and K Birgitta Whaley. Minimum construction of two-qubit quantum operations. *Physical review letters*, 93(2):020502, 2004.

[214] Rong Zhou and Eric A. Hansen. Multiple sequence alignment using anytime A*. In Rina Dechter, Michael J. Kearns, and Richard S. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*, pages 975–977. AAAI Press / The MIT Press, 2002.

Synthèse en français

Nous étudions l'optimisation de ressources classiques et quantiques lors de la synthèse de circuits quantiques. La synthèse de circuits quantiques consiste en la transformation d'une spécification haut niveau d'un algorithme en une séquence d'instructions élémentaires exécutables directement par l'ordinateur quantique : un circuit quantique. Cette étape s'inscrit dans le contexte plus général de la compilation d'algorithmes quantiques et est une problématique essentielle vu que la capacité à générer un circuit optimisé pour un algorithme est la garantie de sa bonne exécution sur une machine quantique dont les ressources sont actuellement très limitées. Le problème de la synthèse de circuits quantiques se décline en plusieurs sous-problèmes, chacun rattaché à une représentation abstraite différente des opérateurs à synthétiser: matrice unitaire, formule booléenne, etc. La première partie de cette thèse s'intéresse à la synthèse d'opérateurs quantiques représentés par une matrice unitaire. Là où la littérature s'est essentiellement intéressée à optimiser la taille du circuit quantique final, nous étudions le compromis entre ressources quantiques — la taille du circuit quantique généré — et ressources classiques — la quantité de calcul classique nécessaire pour générer le dit circuit quantique. Nous proposons deux méthodes qui améliorent l'état de l'art aux deux extrêmes de ce compromis. Dans un premier temps nous développons une méthode nécessitant peu de ressources classiques, efficacement parallélisable sur des architectures multi-coeurs ou GPU, tout en générant des circuits seulement deux fois plus gros que ceux fournis par le meilleur algorithme dans la littérature. En contrepartie notre méthode est capable de synthétiser des opérateurs sur des tailles de problèmes inaccessibles par les autres méthodes existantes. Dans un second temps nous explorons l'utilisation d'algorithmes d'optimisation numériques pour une synthèse optimale. Nous mettons en évidence qu'il est possible de synthétiser des opérateurs sur un petit nombre de qubits avec une taille de circuit optimale donnée par une borne inférieure théorique. Finalement nos deux méthodes se complémentent avec le meilleur algorithme de l'état de l'art, chaque méthode étant la plus efficace pour une plage de tailles d'opérateurs donnée. La seconde partie de cette thèse s'intéresse à la synthèse d'une classe d'opérateurs spécifiques : les opérateurs dits linéaires réversibles. Dans cette partie nous nous intéressons exclusivement à l'optimisation des ressources quantiques et nous considérons deux métriques : la taille du circuit, métrique déjà utilisée durant la première moitié de cette thèse et ici donnée par le nombre de portes CNOT dans le circuit, mais également la profondeur du circuit qui est très étroitement liée au temps d'exécution du circuit quantique. Nous proposons de nouvelles méthodes aux techniques variées (variante de l'élimination Gaussienne, algorithme diviser pour régner, réduction à un problème de cryptographie) optimisant ces deux métriques. Chacune de nos méthodes surpasse l'état de l'art dans une grande majorité des cas et globalement nos méthodes, quand elles optimisent la même métrique, sont complémentaires selon la taille et la complexité de l'opérateur à synthétiser. Nous appliquons également nos méthodes pour la compilation d'une bibliothèque de fonctions réversibles. Nous arrivons à diminuer drastiquement le nombre de portes CNOT et la profondeur totale du circuit tout en gardant d'autres métriques d'intérêt (la profondeur des couches des portes T par exemple) les plus faibles possibles.

Remerciements

Je souhaite remercier dans un premier temps Prof. Mark Asch et Prof. Michele Mosca pour avoir accepté de rapporter ma thèse et pour leurs précieux retours. Je les remercie à nouveau ainsi que tous les membres de mon jury de thèse : Dr. Elham Kashefi, Dr. Cécilia Lancien et enfin Dr. Pascale Senellart. Ce fut un plaisir de soutenir devant eux et une joie de pouvoir échanger avec eux.

Cette thèse ne serait pas ce qu'elle est sans l'aide inestimable de mes encadrants. Un grand merci à Cyril pour m'avoir fait confiance et sans qui cette thèse n'aurait pas vu le jour. Marc et Benoît, mille fois merci pour votre soutien sans faille. Vous avez su me laisser autonome tout en me recadrant quand je partais trop en vrille. J'ai beaucoup appris à vos côtés.

Une thèse, c'est long. Un peu trop long même. Heureusement j'ai pu compter sur deux équipes formidables. Côté Atos je remercie Alessandro, Arnaud, Bertrand, Satia, Simon, Thomas, ainsi que tout le personnel de la cantine. Côté LRI je remercie Aygul, Chuan, Daniel, David, Fabien, Georges, Hai, Houssem, Nicolas, Laércio, Oguz, Pierre, pour tous ces moments studieux autour du billard, du baby, pour toutes les fois où ils ont voulu bosser avec moi qui bavassait à côté. Je remercie tout particulièrement Marie qui m'a appris à imprimer un manuscrit et à distribuer des bouteilles d'eau.

Une thèse c'est long, mais 25 ans encore (un peu) plus. Aussi je remercie tous ceux qui partagent de près ou de loin mon cercle social. Mes amis d'enfance Adrien, Éléonore, Émeric, Florence, Gabriel, Sivane, Edwin, Frédérick, Guillaume, Oiknine, Thibault, puis tous ceux que j'ai rencontrés durant ma vie de jeune adulte : Adrien, JB, JP, Miguel, Victor, Xavier pour ces deux années BJ ; Antoine, Damien, Florian, Francis, François, Georges, Guillaume, Marin, Michel, Nicolas, Pierre, Rémi, Sélim, Thomas, Vérène pour ces années Supélec et au-delà. La vie est quand même plus drôle avec vous, et comme disait ce philosophe moderne : "Ma question préférée : qu'est-ce j'vais faire de tout cet oseille ? Moi et mes kheys on part sur la Lune, amuse-toi bien en Meurthe-et-Moselle". J'ai une pensée particulière pour mes colocataires actuels ou passés : Jean, Adrien, Damien, Francis, Rémi, Pierre, Florian, Clemens, Nicolas[1], Edwin, qui ont compris un peu tard que ma maniaquerie légendaire était effectivement une légende.

25 ans c'est long, mais une vie encore un peu plus (j'espère en tout cas). Je remercie mes parents qui m'ont porté et supporté pendant assez longtemps maintenant. Je remercie également mes soeurs, et Tiana pour soutenir un gars comme moi. Enfin plus généralement je vais remercier toute ma famille et notamment mes ancêtres qui ont eu le bon goût de faire en sorte que je sois là ici.

Je remercie Nobu pour tous les cache-cache ludiques, vols de nourriture et autres déchiquetages de mains ; enfin je remercie Léon pour m'avoir soufflé toutes les contributions de ma thèse.

[1] Et aux jeudi sushi, définitivement le moment phare de la semaine.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Méthodes pour l'optimisation de la synthèse de circuits quantiques

Mots clés: calcul quantique, compilation de circuits quantiques, circuits linéaires réversibles, optimisation de ressources, calcul haute performance

Résumé: Pour exécuter un algorithme abstrait sur un ordinateur quantique il faut compiler l'algorithme en une séquence d'instructions bas niveau exécutables par le processeur. L'étape de compilation est cruciale car elle détermine la quantité de ressources nécessaire pour l'exécution d'un algorithme ; par conséquent elle se doit d'être optimisée. Dans cette thèse nous nous intéressons à une brique de la compilation : la synthèse de circuits quantiques à partir d'une spécification abstraite d'un opérateur. Dans un premier temps nous étudions le cas où la matrice unitaire d'un opérateur quantique nous est donnée et nous explorons la minimisation des ressources quantiques et la minimisation des ressources classiques. Même si l'optimisation simultanée de ces deux types de ressources semble difficile, nous proposons de meilleurs compromis améliorant la littérature. Dans un second temps nous nous intéressons à la classe des opérateurs dits linéaires réversibles. Nous nous intéressons cette fois-ci exclusivement à l'optimisation des ressources quantiques et nous améliorons l'état de l'art dans diverses cas de métriques (taille et profondeur du circuit) et de processeurs quantiques (processeurs NISQ, ou à connectivité complète).

Title: Methods for optimizing the synthesis of quantum circuits

Keywords: quantum computing, compilation of quantum circuits, linear reversible circuits, resources optimization, high-performance computing

Abstract: To run an abstract algorithm on a quantum computer, the algorithm must be compiled into a sequence of low-level instructions that can be executed by the processor. The compilation step is crucial because it determines the quantity of resources necessary for the execution of an algorithm. Therefore, the compilation stage must be optimized. In this thesis, we are interested in a brick of compilation: the synthesis of quantum circuits from an abstract specification of an operator. First, we study the case where the unitary matrix of a quantum operator is given to us and we explore the minimization of both quantum resources and classical resources. Even if the simultaneous optimization of these two types of resources seems difficult, we propose better compromises improving the literature. Secondly, we are interested in the class of so-called reversible linear operators. This time we are exclusively interested in the optimization of quantum resources and we improve the state of the art in various cases of quantum metrics (circuit size, circuit depth) and processors (NISQ, fully-connected processors).

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France