



Report on Quantum Computing exploratory research

Moreno Giussani, Samuele Pino

Project for Software Engineering 2, Politecnico di Milano

June 12, 2019

Contents

1	Introduction	3
1.1	Abstract	3
1.2	Quantum circuits	3
1.2.1	Matrix representation of circuits	3
1.2.2	Some elementary gates	4
1.2.3	Quantum gates in series	4
1.2.4	Quantum gates in parallel	6
1.2.5	Controlled gates	6
2	Quantum Development Kit	9
2.1	Example: Grover Search implementation	9
3	Maximum Flow Analysis	10
3.1	Feasibility analysis	10
4	Insights on Grover Search Algorithm and its implementation	11
4.1	Introduction	11
4.1.1	Premise	11
4.1.2	The problem	12
4.2	The phone book implementation	12
4.3	Feasibility analysis	13

5	Quantum gates in Octave	14
5.1	Elementary (existing) gates	14
5.1.1	Hadamard and X, Y, Z	14
5.1.2	CNOT	15
5.1.3	SWAP	16
5.1.4	CCNOT	16
5.1.5	CSWAP	17
5.2	Operations between gates	18
5.2.1	Kronecker product (or direct product)	18
5.2.2	Gate control (direct sum)	18
5.3	New (derivated) gates	19
5.3.1	DSWAP: double corner swap	19
5.3.2	ICNOT: inverted CNOT	19
5.3.3	CNOT3	20
5.3.4	SWAP3	22
5.3.5	SHIFT3	22
5.3.6	CNOT4	23
6	Other ways explored during the research	25
7	Conclusions	26

Chapter 1

Introduction

1.1 Abstract

1.2 Quantum circuits

We assume the reader already knows the basic concepts about linear algebra and elementary quantum gates. For more information on these basic topics, we recommend [10, 14].

Here we propose a fast walk-through of some possible compositions of quantum gates in the context of a circuit. For further explanations and a more complete reading on the topic we recommend [14, p. 123–129].

1.2.1 Matrix representation of circuits

Operations that make sense in quantum computing are usually performed on more than 2 or 3 qubits and they often give as an output multiple qubits as well. Such computations can be performed by long and complex circuits, therefore we need to be able to decompose them into a sequence of simpler quantum gates. A circuit can be represented by a unitary matrix, which

can mathematically describe the operations performed on an array of input qubits.

Consider a qubit array $|b\rangle = [b_0, b_1]^T$ where b_0 and b_1 are respectively the most and least significative qubits. Therefore a unitary matrix U applied on a $|b\rangle$ will have the following representation:

$$\begin{array}{c} b_0 : \\ b_1 : \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{U} \\ \boxed{U} \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array}$$

$$U |b\rangle = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

where on the right side we have a standard matrix multiplication. Please note that the order of this product is important, as it is *not commutative*.

1.2.2 Some elementary gates

A computation usually requires a combination of elementary gates in 3 main ways: sequentially, in parallel or conditionally. In Table 1.1 we show some gates (and their matrix form) that will be useful for the rest of this document.

1.2.3 Quantum gates in series

The series of quantum gates applied on a qubit line (or on a subset of qubit lines) in a circuit is equivalent to the *dot product* between the matrices of each gate in reverse order.

$$\begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{A} \\ \boxed{A} \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{B} \\ \boxed{B} \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{C} \\ \boxed{C} \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array}$$

$$C \cdot B \cdot A$$

Matrices A, B and C must be of the same size (in this example, being applied on 2 bits, they must be 4×4). The result matrix will be obviously of the same size of A, B and C (4×4).

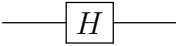
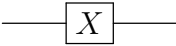
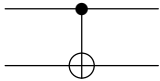
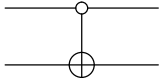
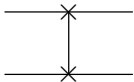
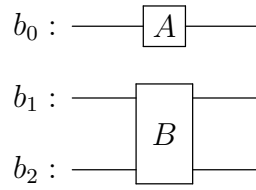
Hadamard gate		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
NOT gate		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
CNOT (on $b_0 = 1$)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
CNOT (on $b_0 = 0$)		$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
SWAP gate		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Table 1.1: Some elementary quantum gates in circuit and matrix representation.

1.2.4 Quantum gates in parallel

Applying distinct quantum gates to disjoint subsets of qubits is equivalent to the *direct product* (or *tensor product*, or *kroncker product*) between the matrices of each gate. Here the order is given by the position of the qubits to which gates are applied (most significative first).



$$A^{(2 \times 2)} \otimes B^{(4 \times 4)} = U^{(8 \times 8)}$$

Given $A \in M^{m \times m}$ and $B \in M^{n \times n}$, the result matrix will be $mn \times mn$ dimensional. We can easily notice that the matrix dimension grows fast with consecutive applications of direct product and the resulting dimension is always a power of 2 in quantum circuits.

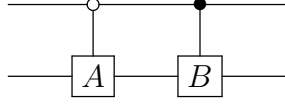
A special case is when some qubits have a gate applied, while others have nothing (that is equivalent to an identity matrix).

The diagram shows two horizontal lines representing qubits. The top line is labeled b_0 and has a small square box labeled X in the middle. The bottom line is labeled b_1 and has no box.

$$X \otimes I_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

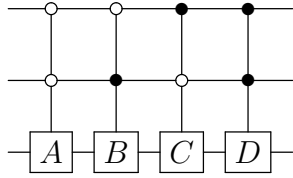
1.2.5 Controlled gates

The effect of a gate on a subset of qubits (“targets”) can be applied conditionally to the value of one or more other qubits (called “controls”). This operation is equivalent to a *direct sum* between matrices.



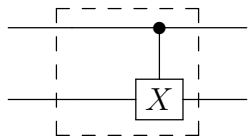
$$A \oplus B = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ 0 & 0 & b_{11} & b_{12} \\ 0 & 0 & b_{21} & b_{22} \end{bmatrix}$$

in this example $A \oplus B$ means that gate A is applied to the bottom qubit if the top one is in state $|0\rangle$, while B is applied if the top qubit is in state $|1\rangle$. It can be easily generalized to the case of 2 control qubits:

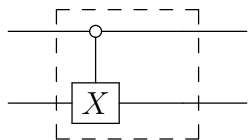


$$A \oplus B \oplus C \oplus D = \begin{bmatrix} A & 0 & 0 & 0 \\ 0 & B & 0 & 0 \\ 0 & 0 & C & 0 \\ 0 & 0 & 0 & D \end{bmatrix}$$

in general if we have n_c control lines and n_t target lines, we can obtain a direct sum of up to 2^{n_c} gates, each gate of dimension 2^{n_t} . If we have less than 2^{n_c} gates to control, the missing spots in the direct sum are filled by appropriate sized identity matrices:



$$I \oplus X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



$$X \oplus I = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Chapter 2

Quantum Development Kit

2.1 Example: Grover Search implementation

Chapter 3

Maximum Flow Analysis

3.1 Feasibility analysis

Chapter 4

Insights on Grover Search Algorithm and its implementation

4.1 Introduction

4.1.1 Premise

Our Quantum Max Flow Analysis algorithm described in Chapter 3, bases its reason to exist in the fact that a small routine of the algorithm (the search for the next arc to be considered among those coming out of a given node) is done by a quantum computer. It is in fact nothing more than a search in a list (or more generally in a database) of one or more elements that satisfy a certain condition (the arcs that have not been visited yet, i.e. having infinite weight value).

4.1.2 The problem

Let's start noticing that although we have presented in Section 2.1 an implementation of Grover Search in Q#, it actually works on what is known as “virtual database”. Alike real databases, virtual (or implicit) ones are not really databases: given n as the number of bits, they are nothing more than the set of integer numbers $[0, 2^n - 1]$.

Such a “database” can be easily implemented with a quantum register initialized with $H^{\oplus n}$. In this way, whenever the register is measured, it collapses to one of all the possible combinations of its bits (i.e. $[0, 2^n - 1]$), being all these combinations all equally probable.

Actually the implementation described in Section 2.1 complies with most of the available literature [9, 11]. It is evident that currently most of the works somehow related to Grover Search Algorithm are devoted to quantum search on virtual databases. [7]

Apparently some people agree that Grover is limited to implicit databases, therefore not convenient or even not useful at all for real databases [13, 15, 3, 1, 2]. On the other hand, someone had a deeper study on the algorithm, understanding the mechanism and implementing (at least mathematically) the encoding and the search on a real database. [5]

4.2 The phone book implementation

The work [5] actually finds a way to encode some elements into a real database. It is done setting a register to an entangled state, as sum of the states corresponding to the elements that we want to encode into the database. This database-register is created by applying a particular matrix

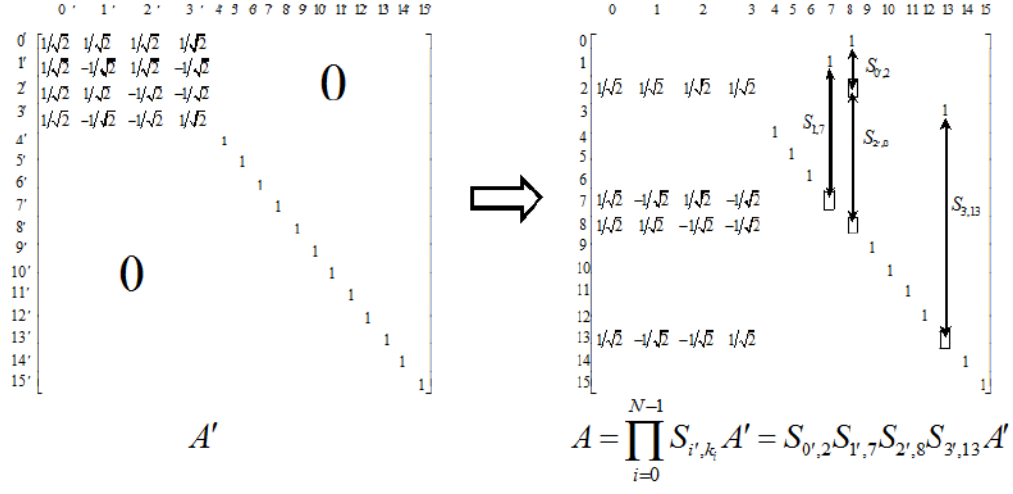


Figure 4.1: Successive row swapping operations to transform A' in the prime frame to A in the unprimed frame for the specific telephone database example. (credits to [5])

to it, in which the database elements are encoded within the rows order. An example is shown in Figure 4.1.

4.3 Feasibility analysis

Chapter 5

Quantum gates in Octave

Octave is a free software and a scientific programming language whose syntax is largely compatible with Matlab.

To fill the gap between some theoretical papers (which perform calculations on matrices) and quantum gates (that are eventually how those matrices are implemented) we modeled some quantum matrices as combination of known gates. In this way it was possible to investigate on how such matrices could be really implemented.

5.1 Elementary (existing) gates

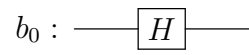
5.1.1 Hadamard and X, Y, Z

We will show only H as an example, but the same applies for X , Y and Z and in general for 2×2 gates.

Matrix

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Circuit



Octave code

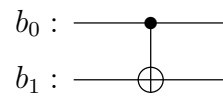
```
| H = [  
1,    1;  
1,   -1  
] ./ sqrt(2);
```

5.1.2 CNOT

Matrix

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

Circuit



Octave code

```
| CNOT = C(X);
```


5.1.3 SWAP

Matrix

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Circuit

$$\begin{array}{l} b_0 : \text{---}\times\text{---} \\ b_1 : \text{---}\times\text{---} \end{array}$$

Octave code

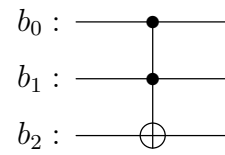
```
| SWAP = [  
1, 0, 0, 0;  
0, 0, 1, 0;  
0, 1, 0, 0;  
0, 0, 0, 1;  
];
```

5.1.4 CCNOT

Matrix

$$CCNOT = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Circuit



Octave code

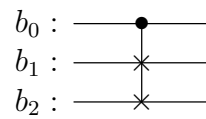
```
| CCNOT = C(CNOT);
```

5.1.5 CSWAP

Matrix

$$CSWAP = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Circuit



Octave code

```
| CSWAP = C(SWAP);
```

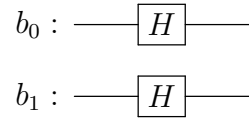
5.2 Operations between gates

5.2.1 Kronecker product (or direct product)

Matrix

$$H \otimes H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Circuit



Code

```
| kron (H, H);
```

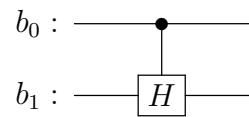
5.2.2 Gate control (direct sum)

Matrix

$$I \oplus H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \oplus \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

Note that CNOT is a “controlled X ”.

Circuit



Code

```
function out = C(gate)
    size = rows(gate);
    out = blkdiag(eye(size), gate);
endfunction;
```

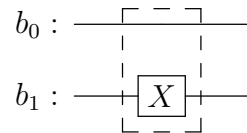
5.3 New (derivated) gates

5.3.1 DSWAP: double corner swap

Matrix

$$SWAP = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Circuit



Octave code

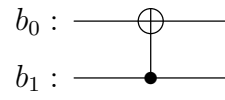
```
| SWAP = kron(eye(2), X);
```

5.3.2 ICNOT: inverted CNOT

Matrix

$$ICNOT = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that **it is not equivalent** to this circuit:



whose matrix is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Circuit



Octave code

```
| ICNOT = DSWAP * CNOT;
```

5.3.3 CNOT3

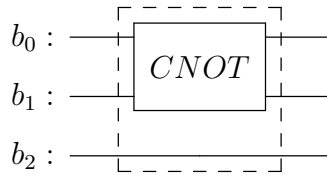
CNOT of grade 3.

Matrix

$$CNOT3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Please note that it is different from CCNOT.

Circuit



Octave code

```
| CNOT3 = kron(CNOT, eye(2));
```

5.3.4 SWAP3

Matrix

$$SWAP3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Octave code

```
| SWAP3 = kron(SWAP, eye(2));
```

5.3.5 SHIFT3

Matrix

$$SHIFT3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Octave code

```
| SHIFT3 = kron(SHIFT, eye(2));
```

5.3.6 CNOT4

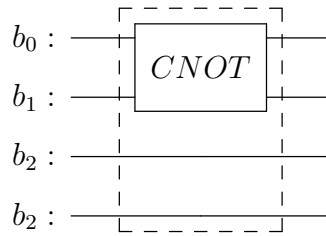
CNOT of grade 3.

Matrix

$$CNOT4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Please note that it is different from CCCNOT.

Circuit



Octave code

```
| CNOT4 = kron(CNOT, eye(4));
```

Chapter 6

Other ways explored during the research

Grover's Algorithm was originally devised to work with functions that are satisfied by a single input. Actually it has also been "generalized to search in the presence of multiple *winners*". [6]

It turns out that Grover's Algorithm can be more useful in "speeding up the solution to NP-complete problems such as 3-SAT" than actual search. [12]

We also considered spatial database search as a possible way of exploiting Grover's Algorithm, in the specific case of graphs with costs on arcs [4, 8].

Although there are some points of contact with Grover, none of them seemed to me of any use for our Max Flow Analysis problem.

Chapter 7

Conclusions

Bibliography

- [1] Grover's algorithm: what to input to oracle? <https://quantumcomputing.stackexchange.com/questions/2149/grovers-algorithm-what-to-input-to-oracle>, 2018.
- [2] Grover's algorithm: where is the list? <https://quantumcomputing.stackexchange.com/questions/2110/grovers-algorithm-where-is-the-list>, 2018.
- [3] How is the oracle in grover's search algorithm implemented? <https://quantumcomputing.stackexchange.com/questions/175/how-is-the-oracle-in-grovers-search-algorithm-implemented>, 2018.
- [4] D. Aharonov, A. Ambainis, J. Kempe, and U. Vazirani. Quantum walks on graphs. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 50–59, New York, NY, USA, 2001. ACM.
- [5] P. Alsing and N. McDonald. Grover's search algorithm with an entangled database state. *Proceedings of SPIE - The International Society for Optical Engineering*, 05 2011.

- [6] M. Boyer, G. Brassard, P. Hyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(45):493–505, 1998.
- [7] B. Broda. Quantum search of a real unstructured database. *The European Physical Journal Plus*, 131(2):38, Feb 2016.
- [8] A. M. Childs and J. Goldstone. Spatial search by quantum walk. *Physical Review A*, 70(2):022314, 2004.
- [9] L. 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.
- [10] A. Helwer. Quantum computing for computer scientists. Available at <https://www.microsoft.com/en-us/research/video/quantum-computing-computer-scientists/>, 2018.
- [11] C. Lavor, L. Manssur, and R. Portugal. Grover’s algorithm: quantum database search. *arXiv preprint quant-ph/0301079*, 2003.
- [12] A. Montanaro. Quantum algorithms: an overview. *npj Quantum Information*, 2:15023, 2016.
- [13] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Is quantum search practical? *Computing in Science Engineering*, 7(3):62–70, May 2005.
- [14] C. P. Williams. *Explorations in Quantum Computing*. Texts in Computer Science. Springer, 2nd edition, 2011.
- [15] C. Zalka. Using grover’s quantum algorithm for searching actual databases. *Physical Review A*, 62:52305, 10 2000.