

Quantum Research Project

Pierriccardo Olivieri

March 2020

Abstract

This project aims to analyze in a computer science perspective the quantum version of random walks: the quantum walks. This model of computation can be seen as a building block to construct new algorithms, here we are interested in a graph application in the domain of discrete time quantum walks. The question that this research want to answer is if a speedup w.r.t. classical is possible and discuss eventual limitations.

Contents

1	Introduction	1
1.1	Random Walks	1
1.2	Quantum Walks	2
2	Coined Quantum Walk	2
2.1	Components description	3
2.2	Circuit for the CQW	3
3	Results	4
3.1	Results	4
4	Performance evaluation	5
4.1	Generalizations	5
4.2	Performance	5
4.3	The search problem	5
4.4	Efficiency of a QW circuit . . .	6
4.5	Scalability and limitations of coined DTQW	6
5	Szegedy quantum walk	6
5.1	Markow chains	6
5.2	application to graphs	6
5.3	The szegedy QW operator . . .	7
6	Applications	7
7	Conclusions	7

A Coined Quantum walk

9

1	1
1	1
2	2
2	2
3	3
3	3
4	4
4	4
5	5
5	5
5	5
6	6
6	6
6	6
6	6
7	7
7	7
7	7

1 Introduction

1.1 Random Walks

A random walk, also known as a stochastic or random process, is a mathematical object which describe a path constituted by random step over a mathematical space. A common example is a random walk in a line, consider the integer set \mathbb{Z} , starting from a certain point, for instance $x = 0$, the path is defined by randomly choose to move left or right, the movement is achieved increasing or decreasing the value of x by 1. Tossing a coin will help in choosing randomly, with equal probability, the next step to take. This example could be generalized by increasing the dimension of the mathematical space considered, in a cartesian plane the starting point will have two coordinates and the possible moves becomes 4, and there are many other possible generalizations.

The random walk can be divided in two major classes, discrete-time and continous-time random walk. Intuitively as the names suggest the difference between this two class is in the time function that could be integer or real, for

a more formal definition consider the random walk as a system composed by a family of random variables X_t the variables X_t will measure the system at time t , now if we consider $t \in \mathbb{N}$ is a discrete-time stochastic process, otherwise if $t \in \mathbb{R}^+ \cup 0$ is a continuous-time stochastic process.

Here we focus on the discrete-time, in particular we bind this to graphs considering the random walk on a cyclic graph of order N .

1.2 Quantum Walks

The Quantum version of the random walks is called Quantum Walks, also here there is a distinction between the two model of discrete quantum walks and continuous quantum walks, the focus of this research remain in the discrete-time domain also for the quantum side.

The discrete quantum walk described here is called Coined Discrete Quantum Walk on a Line (Coined DQWL), there are also versions without coin. It's worth introducing the Coined QDWL in a formal and generic way, then apply to a more concrete example. A formal description starts considering the three main components of a Coined QDWL: A walker operator, a coin, an evolution operator usually called shift operator.

The Walker represents the position of our system and is defined in a Hilbert space infinite but countable \mathcal{H}_p , a vector in that space represents the position of the walker, $|position\rangle \in \mathcal{H}_p$

The coin operator is defined in a two-dimension Hilbert space, if we consider as basis state $|0\rangle$ and $|1\rangle$ then the coin space will become $\mathcal{H}_c = |0\rangle, |1\rangle$ and with $|coin\rangle \in \mathcal{H}_c$.

The complete system finally will be in a Hilbert space composed by the Kronecker product of the two spaces above defined $\mathcal{H} = \mathcal{H}_p \otimes \mathcal{H}_c$. A state of the coined DQWL can be defined by the vector:

$$|\phi_{initial}\rangle = |position\rangle_{initial} \otimes |coin\rangle_{initial} \quad (1)$$

The evolution operator, also called shift operator will actually perform a step starting from the initial position, apply this operator to the system is equal to toss a coin and depending on the outcome move the walker to the left or to the right, we can do this by increment or decrement the actual position by 1. A possible form of the shift operator could be described by this formula:

$$S = |0\rangle_c \langle 0| \otimes \sum_i |i+1\rangle_p \langle i| + |1\rangle_c \langle 1| \otimes \sum_i |i-1\rangle_p \langle i| \quad (2)$$

Finally we can see a walker as operator itself, called U and given by:

$$U = S \times (C \otimes I_p) \quad (3)$$

Applying this operator to a given system is equal to perform a step of a random walk. A more formal explanation can be found in [Ven08] and [Kem03]. The example on the following section apply this general concept to a cyclic graph.

2 Coined Quantum Walk

The following example shows how to implement a Coined discrete quantum walk on a cyclic graph with $N = 8$ nodes. This can be achieved using the coined DQW on a line where the line is the cyclic graph. First we need to encode the graph nodes in a binary notation to fit with qubits. In general a graph with 2^n nodes needs n encoding bit, in our case since we have 2^3 nodes we can use 3 bit to encode. The Figure 1 below shows how we can bind the qubits to the nodes of the graph.

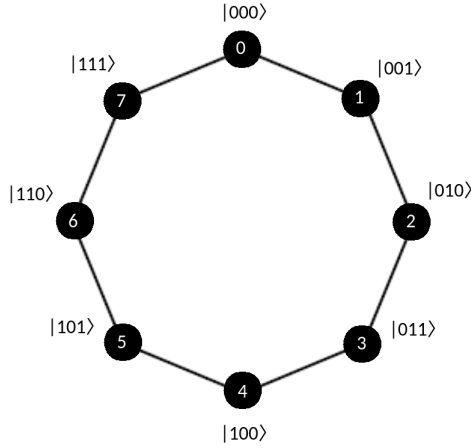


Figure 1: Cyclic graph with 8 nodes, and the respective position state in qubits version

2.1 Components description

We can fit now the 3 main components mentioned in the previous section for this specific example.

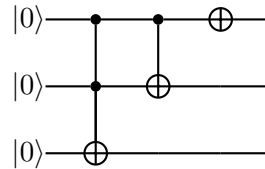
Starting from the **walker operator**: for this version we need to encode the nodes of the graph in 3 qubits, usually we need $\log(N)$ qubits, in general this is not always true, we will see why in the next section. Using 3 qubits the position of the walker is given by the binary encoding of the node, for instance the node 0 is represented as $|000\rangle$.

For the **coin operator** operator, we will use an Hadamard coin, that consists in apply the Hadamard operator to the system.

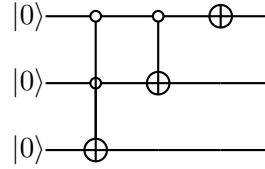
The **shift operator** operator in this case will move the actual position that we call $|i\rangle$ to one of the adjacent nodes, that corresponds to $|i+1\rangle$ or $|i-1\rangle$, which is decided by the outcome of the coin operator.

2.2 Circuit for the CQW

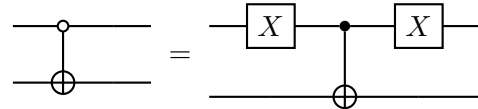
To implement the circuit in qiskit we need to translate the operator defined in quantum circuits, first we need 1 qubit for the coin and 3 for the position. The Hadamard can be easily implemented using an Hadamard gate on the first qubit. Then we need a circuit to perform an increment and a decrement on the initial state, this is less trivial and to achieve that we need two sub circuit that uses multi controlled toffoli gates. The circuit below represents an incrementer circuit for 3 qubits.



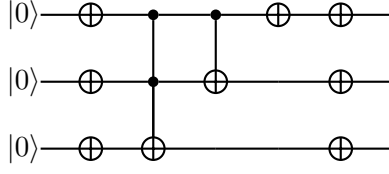
The decrement circuit is similar but uses negative controlled not gates, the circuit below shows the decrement circuit for 3 qubits.



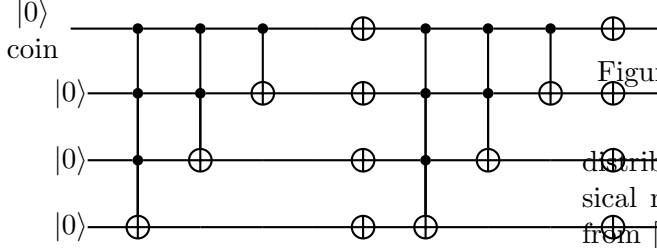
The negative controlled not can be represented by negate before and after the controlled not, the equivalence circuit below clarify this explanation, for a detailed explanation look at [NC10].



Finally the decrement circuit defined above, using this equivalence, is showed below.



A similar implementation in detail is covered in [DW07]. The final circuit by combining all the components defined is showed below.



It's worth to make some comments about it, this circuit represents the U operator, in fact the Hadamard coin will randomly choose the direction to take and activate the increment or decrement sub circuit. Therefore this corresponds to a single iteration of the walk, by successively apply this circuit we can perform a random walk on the cyclic graph. The code for this circuit can be found in the Appendix A at the end of this document.

3 Results

3.1 Results

The circuit obtained in the previous section can be applied to a system of 4 qubits several times to perform a random walk. Fig. 2 shows the results of a 100 steps random walks, obtained by apply the circuit 100 times.

As we can notice in the picture the probabilities of find the walker in a given position are quite asymmetrical and also since we started from an even number, odd numbers have zero probability of being measured, in fact, differently from classical, we can see a true randomness behavior different from a Gaussian

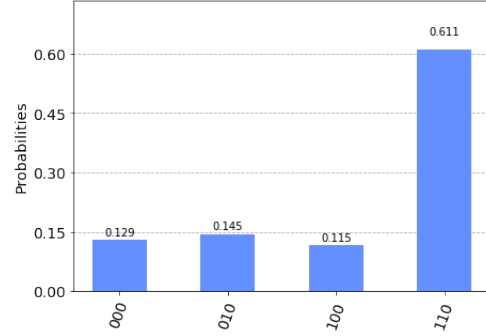


Figure 2: results of a 100 step random walk

distribution that we will observe in a classical random walk. The image belowe taken from [KT04] shows a comparison between the gaussian distribution of classical systems and the distribution observed for the quantum after 100 step of the walk.

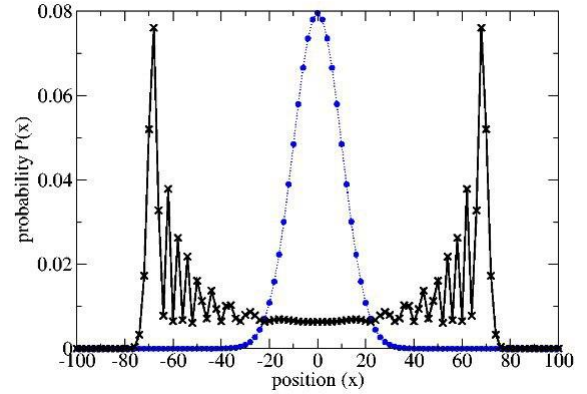


Figure 3: comparison of distribution of a 100 step random walk between classical and quantum

This means that the walker can pass through or be in an odd(even) but the probability to measure the Walker in a odd(even) position starting from an even(odd) position after 100 steps of the walk is very close to zero. This strange behavior is due to the Hadamard coin, this coin treats the two direction (left, right) differently, considering the two states $|\uparrow\rangle$ and

$|\downarrow\rangle$ this coin multiply the phase by -1 only in case of $|\downarrow\rangle$ inducing more cancellations of the right-wards paths. [Kem03]

The asymmetrical behavior can be modified by changing the initial state or changing the coin or balance the Hadamard coin, a more formal description of this behavior is showed in [Ven08].

This particular distribution is interesting since shows that quantum can achieve a true randomness, classical computer can only simulate a random behavior, for sure useful and well done, but still simulated.

4 Performance evaluation

4.1 Generalizations

The example presented is just an application to cyclic graph, also the 3 main component presented, the walker, the coin and the shift operator are quite generic. This method in fact can fit different type of problems and graph. Starting from the coin there are many others operator that we can use, which are symmetrical, other examples are the balanced coins that get rid of this asymmetrical behavior of the Hadamard coin. For more details about this coins an introductory summary can be found in [Kem03], but there are also Quantum walks without the coin, as it is mentioned in [Ven08].

The interesting thing is that the example presented can be adapted to a variety of other graphs, for instance we can build a circuit able to perform a quantum walks also for completed graphs, hypercube, glued trees and others. Obviously, changing the graph, the circuit needs to be adapted but the idea remains the same. A very usefull references that shows circuits for the type of graphs mentioned is [DW07]

An important remark concern the total qubits needed, as mentioned in the previous section if the number of qubits is greather than 2 we need to use multi toffoli gate, this gates

can be obtained by combining several toffoli gates, with the help of some ancillary qubits, those ancillary qubits are needed just to construct the multi controlled toffoli. In Fig. 3 below is showed how to implement a multi controlled toffoli gates with 5 controls qubits, this image is taken from [NC10]. In general following this procedure we need a number of ancillary equals to controls qubits - 1.

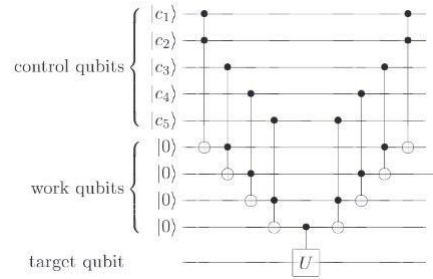


Figure 4: Practical representation of multi toffoli gate

The number of ancillary qubits impacts also in the efficiency of the circuit. In the next section will be provided a more formal description of efficiency.

4.2 Performance

To analyze the performance of quantum walks w.r.t. classical we need to define the quantum walk search problem. A more specific view of possible applications will be discussed at the end of this document.

4.3 The search problem

The search problem consists in find a marked vertex in a given graph by applying random walk. Given a graph G with a set of marked nodes we want to find a marked vertex. The main idea is that starting from a vertex n we perform a step of the random walk and then we check if the current vertex is marked, if not we

continue repeating this procedure. We can also separate this problem in 2 versions, the detecting problem that given the conditions states before, a graph and a set of marked vertex, says if there is a marked vertex, and the actual finding problem that given the same conditions find one marked vertex.

4.4 Efficiency of a QW circuit

To talk about efficiency we need some definitions, the implementation of a quantum walk circuit for a graph walk is said to be Efficient if it can be build with at most $O(\text{poly}(\log(N)))$ one and two qubits where N is the number of nodes in the graph [DW07]. quantum gates. An analysis of quantum walk approach w.r.t. classical is to be done for different family of graphs, since an efficient circuit can not be realized for all the classes of graph. The comparison between quantum and classical is made in [DW14] by considering the relative number of queries to a fixed oracle needed to complete the search. Also the paper provide two definition of efficiency, the first notion require the algorithm to be quadratically faster in search w.r.t. the best possible classical search and the second notion is the definition of efficiency explained above. An example of interest is the hypercube, its quantum representation can achieve a quadratic speedup over the number of query, using a coin biased toward marked nodes. In the paper are presented also results for complete graphs and twisted toroids.

4.5 Scalability and limitations of coined DTQW

An important aspect to consider is the implementation of the controlled NOT gate. As mentioned in the previous section this gate, increasing the number of controls, require some ancillary qubits. In the family of graphs mentioned above in the analysis performed by

[DW14] the number of controls qubits is $O(n)$ and the same is for the ancillary, in this case we still respect the notion of efficiency declared, but for sure is an interesting parameter to consider for the scalability of these circuit.

The method presented, as we just said, can be generalized to other types of graphs but unfortunately is limited to undirected graphs without weights, since various application require weighted and/or directed graphs we need something more generic. In the next chapter is showed a method that can achieve quantum walks also for undirected and weighted graphs. Before presenting this method, we need a brief introduction to Markow Chain, using the Markow Chain representation of the graph we can reduce the limitations and work with directed and weighted graphs.

5 Szegedy quantum walk

The following method proposed by Szegedy helps to reduce some limitations discussed in the previous section. In order to talk about Szegedy algorithm we need to introduct some concepts.

5.1 Markow chains

A Markow chain is a stochastic process that consists in a sequence of random variables X_n with $n \in \mathbb{Z}^+$ such that $P(X_n|X_{n-1}, X_{n-2}, \dots, X_{n-N}) = P(X_n|X_{n-1})$. If is time-independent can be represented by a matrix P called transition matrix. Such that the sum of each row of P is equal to 1.

5.2 application to graphs

We can use then a Markow Chain to perform a random walk on a graph. First, considering a graph $G(V,E)$ we construct the adjacency matrix as follows:

$$A_{i,j} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

Then we can define the transition matrix P as:

$$P_{i,j} = \frac{A_{i,j}}{\text{indeg}(j)} \quad (5)$$

where $\text{indeg}(j)$ represents the number of incoming arc of vertex j . Basically we can express the graph with this transition matrix, this lead to a generalization of the quantum walks presented. The previous method uses the label encoding to represent the graph and was limited to undirected graphs without weights. The Szegedy algorithm instead uses the matrix representation explained above, therefore there are no problem if the graph is directed or even if has weights.

5.3 The szegedy QW operator

The szegedy quantum operator

6 Applications

The methods presented so far are quite general methods that could be applied in various context. For what i understand this is still a fresh research field but has a lot of potential for various application that are difficult to solve. For instance some very famous classical problems, like 3-SAT, could be solved with a speedup w.r.t. classical. Another interesting application presented in [LW17] use the Szegedy Quantum Walk for the Page Rank algorithm: Quantum Page Rank algorithm.

7 Conclusions

In this research we presents the analogue of random walks, the quantum walks in discrete time with a focus on graph applications. We

start from an example of Coined quantum walk on a cycle walk to shows some characteristics and difference w.r.t. classical of this method. This was just an example, in fact this method could be generalized for other types of graphs, or for other version without coins. Then we presents an analysis of this method applied to the problem of search a marked vertex for some family of graphs with quadratic speedup w.r.t. classical and using an efficient circuit. The however method presents some limitations, it works only for undirected and not weighted graph. The Szegedy quantum walk helps to get rid of limitations using a markow chains to repents the graph without express it with encoded label. This method can be efficiently implemented and could be used for interesting applications showed in the prevous section.

References

- [Kem03] J Kempe. “Quantum random walks: An introductory overview”. In: *Contemporary Physics* 44.4 (July 2003), pp. 307–327. ISSN: 1366-5812. DOI: 10 . 1080 / 00107151031000110776. URL: <http://dx.doi.org/10.1080/00107151031000110776>.
- [KT04] Viv Kendon and Ben Tregenna. “Decoherence in Discrete Quantum Walks”. In: *Decoherence and Entropy in Complex Systems: Selected Lectures from DICE 2002*. Ed. by Hans-Thomas Elze. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 253–267. ISBN: 978-3-540-40968-7. DOI: 10.1007/978-3-540-40968-7_18. URL: https://doi.org/10.1007/978-3-540-40968-7_18.

- [DW07] B. L. Douglas and J. B. Wang. *Efficient quantum circuit implementation of quantum walks*. 2007. arXiv: 0706.0304 [quant-ph].
- [Ven08] S. Venegas-Andraca. *Quantum Walks for Computer Scientists*. 2008.
- [NC10] Michael A. Nielsen and Isaac L. Chuang. “Quantum circuits”. In: *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010, pp. 171–215. DOI: 10.1017/CB09780511976667.008.
- [DW14] B. L. Douglas and J. B. Wang. *Complexity analysis of quantum walk based search algorithms*. 2014. arXiv: 1408.1616 [quant-ph].
- [LW17] T. Loke and J.B. Wang. “Efficient quantum circuits for Szegedy quantum walks”. In: *Annals of Physics* 382 (July 2017), pp. 64–84. ISSN: 0003-4916. DOI: 10.1016/j.aop.2017.04.006. URL: <http://dx.doi.org/10.1016/j.aop.2017.04.006>.

A Coined Quantum walk

This code below is referred to coined quantum walk, using Hadamard coin for a cycle graph with number of nodes $N=3$.

```
1 from qiskit import *
2
3 #increment operator for a 3-bit state register
4 def increment_op(circuit):
5     qr = circuit.qubits
6     circuit.mct([qr[0],qr[1],qr[2]],qr[3],None,mode='noancilla')
7     circuit.ccx(qr[0],qr[1],qr[2])
8     circuit.cx(qr[0],qr[1])
9     circuit.barrier()
10    return circuit
11
12 #decrement operator for a 3-bit state register
13 def decrement_op(circuit):
14     qr = circuit.qubits
15     circuit.x(qr)
16     circuit.barrier()
17     circuit.mct([qr[0],qr[1],qr[2]],qr[3],None,mode='noancilla')
18     circuit.ccx(qr[0],qr[1],qr[2])
19     circuit.cx(qr[0],qr[1])
20     circuit.barrier()
21     circuit.x(qr)
22    return circuit
23
24 #construct the circuit for one step of the random walk
25 def random_walk_step(circuit):
26     #increment operator circuit
27     qr_incr = QuantumRegister(4)
28     increment_circ = QuantumCircuit(qr_incr, name='increment')
29     increment_op(increment_circ)
30     increment_inst = increment_circ.to_instruction()
31
32     #decrement operator circuit
33     qr_decr = QuantumRegister(4)
34     decrement_circ = QuantumCircuit(qr_decr, name='decrement')
35     decrement_op(decrement_circ)
36     decrement_inst = decrement_circ.to_instruction()
37
38     circuit.h(qr[0])
39     circuit.append(increment_inst, qr[0:4])
40     circuit.append(decrement_inst, qr[0:4])
41
42    return circuit
43
44 def random_walk(steps, circuit):
45     for i in range(0, steps):
46         random_walk_step(circuit)
47    return circuit
```