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 Random Walks

A random walk, also known as a stochastic or random process, is a mathematical object which describe a path constitued by random step over a mathematical space. A common example is a random walk in the integer set Z, starting from a certain point, for instance x=0, the path is defined by randomly choose to move left or right by increase or decrease the actual value of x by 1. Tossing a coin will help in choosing randomly, with equal probability, the next step to take. The previous 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. The random walk can be divided in two major classes, discrete and continous random walks, here we focus on the discrete-time, in particular we bind this to graphs to perform the search of a marked vertex.

1.2 Quantum Walks

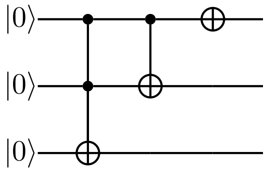
First is worth to give an idea of the random walk above in the quantum world. We can consider a cycle graph like the one in Figure 1. The nodes of the graph can be represented by binary labels, so that we need log(N) quibits to represents a cycle graph with N nodes. Now in order to perform a random walk on this graph we need a way to define a shift operator, i.e. an operator that allows to move, from our current position, in one of the adjiacent nodes and a coin operator that choose randomly the direction to take, left or right in this case.

In the example proposed, considering the current position as $|i\rangle$ shift operator S can move the walker in the left or right position to obtain $|i+1\rangle$ or $|i-1\rangle$ the coin operator C will decide with probability 1/2 wheter to increment or decrement. Defined this two operators the quantum walk operator can be seen as U = SC, applying once this operator U correspond to a random walk step. The states space H in which the walk operator moves is the one defined the two Hilbert spaces, one for the shift operator that we can call $\mathcal{H}_p = |i > wherei = 0...N-1andibelongingtoZ$ and the one for coin operator called $\mathcal{H}_c$ and that in our case can be spanned for instance by $|left\rangle$, $|right\rangle$ finally H is given by $\mathcal{H} = \mathcal{H}_p \otimes \mathcal{H}_c$. This informal explanation is trheated in details by [Kem03].

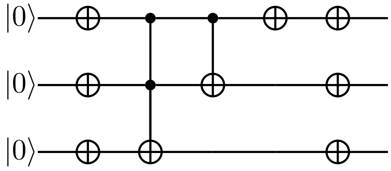
In the example we will use an Hadamard coin, that can simulate a coin toss to decide wheter increment or decrement the actual state. The behavior of the Hadamard coin will be described after the implementation of the example below presented.

2 Coined quantum walk: Qiskit implementation

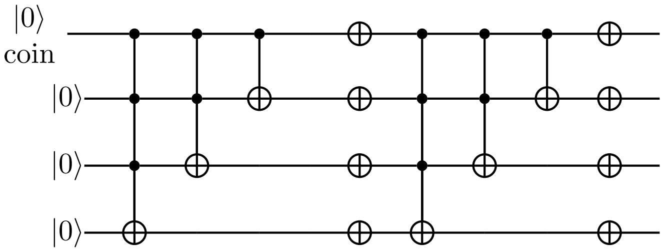
Now we build using Qiskit the circuit for the Quantum Walk described above for the cycle graph when N = 3. Before talking about code it's necessary to provide some representation of the circuit described above. What we need is to define how to represent a label of the graph, with N=8 we need log(N) = 3 bit, so for our quantum implementation we need at least 3 quibit to represent the state plus 1 for the coin. Note that due to the gates needed to implement the circuit the qubit needed can change due to the necessity of ancillary qubits for multi-toffoli gates, this will be explained in the following section when it is explained the general case. To construct the shift operator we need an increment circuit and a decrement circuit. Figure 2 shows how an increment circuit is obtained, basically we need multicontrolled toffoli gate, the circuit below perform an increment so for instance if our actual position is $|000\rangle$ applyng this circuit we will obtain $|001\rangle$ this implementation is based on



the decrement operator use instead the negative controlled not, i.e. we want to perform the decrement operation when the outcome of the coin is zero.



This implementation in detail is covered in [DW07] and i based the implementation in Qiskit using [**garc“IeC –“’“i ˝a2007high**]. Finally the coin that we use for this example will be an Hadamard coin, thus we will apply an Hadamard gate to the first quibit that is the one used for controll. The final circuit is shown below



text.

3 Generalization and results

The example shown above can be generalized in all aspects, let's start with the graph nodes. We have considered for the eample a 8-node graph, but considering the generalized N-node version, are necessary auxiliary qubits for creating multi controlled toffoli gates. Depending on the chosen mode to implement the multi toffoli gate if the control bits are greater than 2 can be required a number of ancillary bits equal to the number of control quibits CQ - 1.

Another generalization concerns the type of graph, by modifying the circuit appropriately it is possible to perform a random walk in different types of graphs including, glued trees, complete graphs etc. a detailed presentation of the various types of graphs with their circuit is present in [DW07]

Shift and Coin operators can also be generalized for various applications, for the Coin operator in particular there are different forms with different properties, the Hadamard coin presented above shows a peculiarity of the Quantum, one would also expect for the Quantum a Gaussian distribution of T-step node visits as happens in the classic version, in reality the distribution obtained is asymmetrical. Figure 2 below shows that the results after T steps in the cyclic graph. More details about this behavior and alternatives symmetrical to the Hadamard coin can be found in [Kem03]

4 Complexity and problem definition

To talk about complexity in Quantum Walks we need to introduce the concept of Hitting Time. A definition can be found in [Ven08]
definition: Hitting Time

Now we can define the problem of searching for a marked vertex. Given the example proposed, using the Hadamard coin, of a cyclic graph with 8 nodes, to search for a marked vertex starting from given vertex we need to perform a walk in the graph and see if the current vertex is the one we are searching. The problem here is how often perform a measure, this choice will impact also on the performances, indeed if we measure the circuit at each step we loose all the advantages of the quantum coerenches and the performances will be equals to classical, we can gain some advantages by set the probability of measuring very small [Kem03].

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.

[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.

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.

```python
from qiskit import *

#increment operator for a 3-bit state register
def increment_op(circuit):
    qr = circuit.qubits
    circuit.mct([qr[0],qr[1],qr[2]],qr[3],None,mode=’noancilla’)
    circuit.ccx(qr[0],qr[1],qr[2])
    circuit.cx(qr[0],qr[1])
    circuit.barrier()
    return circuit

#decrement operator for a 3-bit state register
def decrement_op(circuit):
    qr = circuit.qubits
    circuit.x(qr)
    circuit.barrier()
    circuit.mct([qr[0],qr[1],qr[2]],qr[3],None,mode=’noancilla’)
    circuit.ccx(qr[0],qr[1],qr[2])
    circuit.cx(qr[0],qr[1])
    circuit.barrier()
    circuit.x(qr)
    return circuit

#construct the circuit for one step of the random walk
def random_walk_step(circuit):
    #increment operator circuit
    qr_incr = QuantumRegister(4)
    increment_circ = QuantumCircuit(qr_incr, name=’increment’)
    increment_op(increment_circ)
    increment_inst = increment_circ.to_instruction()

    #decrement operator circuit
    qr_decr = QuantumRegister(4)
    decrement_circ = QuantumCircuit(qr_decr, name=’decrement’)
    decrement_op(decrement_circ)
    decrement_inst = decrement_circ.to_instruction()

    circuit.h(qr[0])
    circuit.append(increment_inst, qr[0:4])
    circuit.append(decrement_inst, qr[0:4])

    return circuit

def random_walk(steps, circuit):
    for i in range(0, steps):
        random_walk_step(circuit)
    return circuit
```