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 focus on a graph application in the domain of discrete time quantum walks. The question that this research want to answer is how this model of computation works and if a speedup w.r.t. classical is possible.

Contents

1 Introduction

1.1 Acronyms

Acronym	Extended
QW	Quantum Walk
DQWL	Discrete QW on a Line
DTQW	Discrete Time QW

1.2 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 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.

Random walks can be divided in two major classes, discrete-time and continuous-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 $t \in \mathbb{R}^+ \bigcup \{0\}$ 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.

1.3 Quantum Walks

The Quantum version of 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.

Following the previous example of the random walk on a line here we discuss about a discrete quantum walk on a line called coined Discrete Quantum Walk on a Line (Coined DQWL), there are also versions without coin. It's worth introducing the Coined DQWL 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 DQWL: 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 becomes $\mathcal{H}_c = |0\rangle, |1\rangle$ 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

$$\mathcal{H} = \mathcal{H}_p \otimes \mathcal{H}_c \tag{1}$$

A state of the coined DQWL can be defined by the vector:

$$|\phi_{initial}\rangle = |position\rangle_{initial} \otimes |coin\rangle_{initial} \tag{2}$$

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:

$$S = |0\rangle_c\langle 0| \otimes \sum i |i+1\rangle_p\langle i| + |1\rangle_c\langle 1| \otimes \sum i |i-1\rangle_p\langle i| \tag{3}$$

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

$$U = S \times (C \otimes I_p) \tag{4}$$

Applying this operator to a given system is equal to perform a step of a random walk. This explanation derives from [13] and [6]. 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 encoding bit. The Figure 1 below shows how we can bind the qubits to the nodes of the graph.

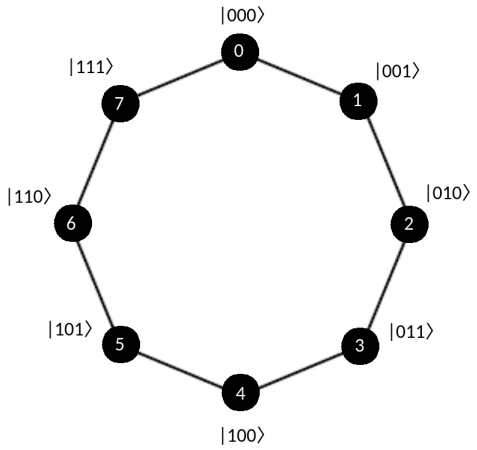


Figure 1: Cyclic graph with 8 nodes, and the respective position state in qubits version

2.1 Components description

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

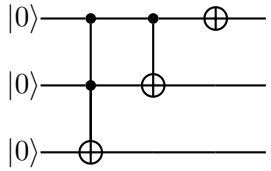
Starting from the **walker operator**: for this version we need to encode the nodes of the graph in 3 qubits, the position of the walker is given by the binary encoding of the node, for instance the node 0 is represented as $|000\rangle$.

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

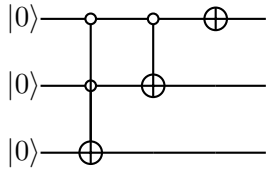
The **shift operator** operator in this case will move the actual position that we call $|i\rangle$ to one of the adjacent nodes, that corresponds to $|i+1\rangle$ or $|i-1\rangle$, which is decided by the outcome of the coin operator.

2.2 Circuit for the CQW

To implement the circuit in qiskit we need to translate the operators defined in quantum circuits, first we need 1 qubit for the coin and 3 for the position, we need log(N) qubits but for encoding larger number we may need more qubits as support for the multi Toffoli gates, we will see why in the next section. 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.

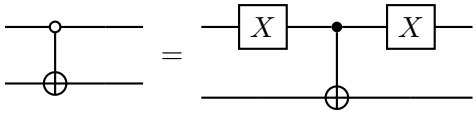


The decrement circuit is similar but uses negative controlled not gates, the circuit below shows the decrement circuit for 3 qubits.

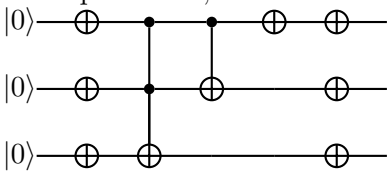


The negative controlled not can be represented by negate before and after the controlled not, the equivalence circuit below clarify this explanation, this circuit equivalence

came from [10] which provide other equivalence to build quantum circuits.



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



To implement the circuit I started with [2] I didn't know how to build the the negative controlled not before finding that equivalence and [4] helped me. The circuit above are displayed thanks to the latex package [5]. The final circuit is obtained by combining all the components defined and is showed below in Fig.2.

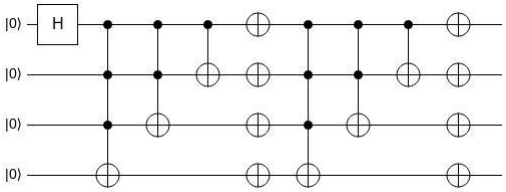


Figure 2: Complete circuit for the coined quantum walk on a cyclic graph with N=8 nodes

Click here for a Quirk Simulation that shows one step of the circuit above, the sub-circuit are upside down since Quirk uses a little-endian convention.

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

The circuit obtained in the previous section can be applied to a system of 4 qubits several times to perform a random walk. Fig. 3 shows the results of a 100 steps random walks, obtained by applying the circuit defined in Fig.2 100 times.

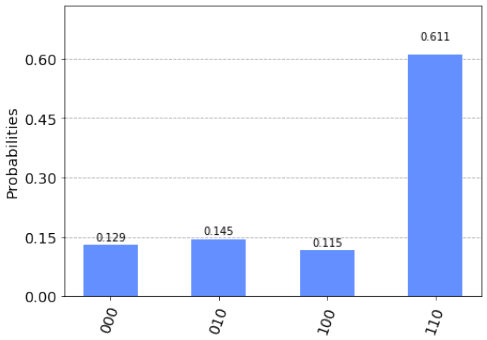


Figure 3: results of a 100 step coined quantum walk

As we can notice in Fig.4 the probabilities of finding 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. Differently from classical, we can see a true randomness behavior different from a Gaussian distribution that we will observe in a classical random walk. The image below taken from [7] shows a comparison between the gaussian distribution of classical systems and the distribution observed for the quantum after 100 step of the walk.

This means that the walker can pass through or be in an odd(even) position 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

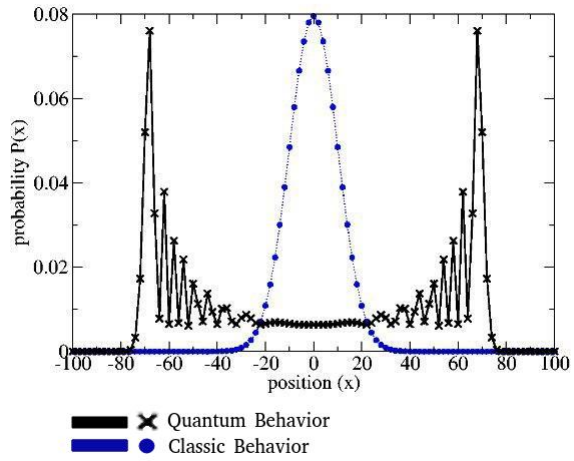


Figure 4: comparison of distribution of a 100 step random walk between classical and quantum

zero. An explanation of this behavior can be found in [6], this strange behavior is due to the Hadamard coin, which 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.

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 [13].

This particular distribution is interesting since shows that quantum can achieve a true randomness, while classical computer simulate with algorithms this random behavior.

4 Generalizations

The example presented is just an application to cyclic graph, also the components presented, the walker, the coin and the shift operator are meant to be 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 asymmetrical behavior of the Hadamard coin. For more details about different coins an introductory summary can be found in [6], but there are also Quantum walks without the coin, as it is mentioned in [13].

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 complete graphs, hypercube, glued trees and others. Obviously, changing the graph, the circuit needs to be adapted but the idea remains the same. A very useful references that shows circuits for the type of graphs mentioned is [2]

An important remark concern the total qubits needed, as mentioned in the previous section if the number of qubits is greater 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. 5 below is showed how to implement a multi controlled toffoli gates with 5 controls qubits, this image is taken from [10]. In general following this procedure we need a number of ancillary equals to controls qubits - 1.

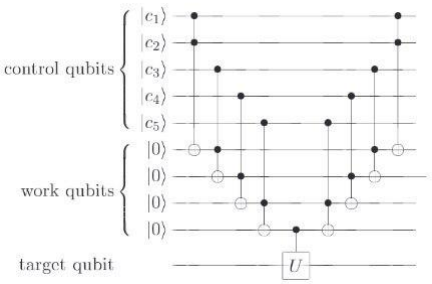


Figure 5: Practical representation of multi toffoli gate

The number of ancillary qubits impacts also in the efficiency of the circuit. In the next sec-

tion will be provided a more formal description of efficiency.

5 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.

5.1 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 steps of the random walk and measure when there is an high probability of measure a marked vertex.

5.2 Efficiency of a QW circuit

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(poly(log(N))) one and two qubits gates where N is the number of nodes in the graph [2]. 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 graphs. The comparison between quantum and classical is made in [3] by considering the relative number of queries to a fixed oracle needed to complete the search. 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 require the circuit to be implementable using O(log(n)) 2-qubit gates. An example of interest is the hypercube, its quantum representation can achieve a quadratic speedup over the number of queries, said N the number of queries to an oracle we have $O(\sqrt{N})$ calls, using a coin biased toward marked nodes. In the paper are presented also results for complete graphs and twisted toroids.

5.3 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 [3] the number of controls qubits is O(log(N)) and the same is for the ancillary, in this case we still respect the notion of efficiency declared, but for sure is a 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.

6 Szegedy quantum walk

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

6.1 Markov chains

A Markov chain is a stochastic process that consists in a sequence of random variables X_n with $n \in \mathbb{Z}^+$ such that:

$$P(X_n|X_{n-1},X_{n-2},...,X_{n-N})=P(X_n|X_{n-1}) \quad (5)$$

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.

6.2 application to graphs

We can use then a Markov Chain to represent the graph. First, considering a graph G(V,E) we construct the adjacency matrix as follows:

$$A_{i,j} = \begin{cases} 1\, if (v_i, v_j) \in E \\ 0\, otherwise \end{cases} \quad (6)$$

Then we can define the transition matrix P as $P_{i,j} = \frac{A_{i,j}}{indeg(j)}$ where indeg(j) represents the number of in-going edges of vertex j. In the cyclic graph C_8 (Fig. 6a) each node has 2 in-going edges therefore the transition matrix is:

$$P = \frac{1}{2}\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Express the graph with this transition matrix give us more flexibility, the previous method used directly labels encoding and was limited to undirected graphs without weights. The Szegedy algorithm works by quantizing the Markov chain, here there are no problem if the graph is directed or has weights.

6.3 The Szegedy QW operator

The Szegedy quantum operator seems quite similar in aspect to coined quantum walk operator but there are lot of differences in the behavior. The space we consider now is an Hilbert space composed by

$$\mathcal{H} = \mathcal{H}_1^N \otimes \mathcal{H}_2^N \quad (7)$$

where N is the number of graph's nodes and $\mathcal{H}$ has dimension N^2. To understand why we can anticipate that this operator works in the bipartite graph generated copying the original node set. We construct 2 sets, X and Y each containing the nodes of the original graph, the bipartite graph will be made by connect nodes between X and Y if they are adjacent, the Fig. 6b below will hopefully clarify this concept, it shows the bipartite graph associated to the cyclic graph C_8 showed in Fig. 6a.

Before introducing the Szegedy operator we need to define some operators that compose it, like in the coined quantum walk where the walk operator was composed by the coin and position operator here we have two different operators called Reflection Operator (R) and Swap Operator (S). To introduce the R operator we need the state vector that represent the system, is given by $|\psi\rangle \in \mathcal{H}$ and is equal to:

$$|\psi\rangle = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} a_{i,j}\,|i,j\rangle \quad (8)$$

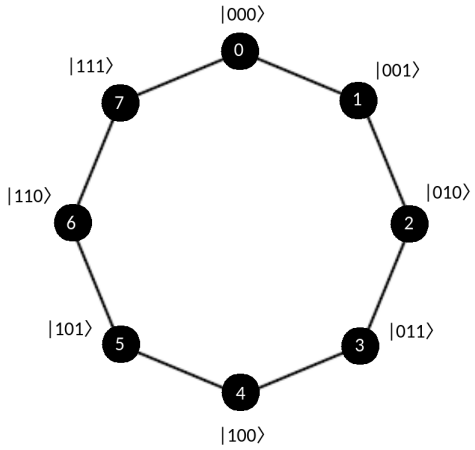
Then we define the projector states of the Markov chain:

$$|\psi_i\rangle = |i\rangle \otimes \sum_{j=0}^{N-1} \sqrt{P_{j+1,i+1}}\,|j\rangle \equiv |i\rangle \otimes |\phi_i\rangle \quad (9)$$

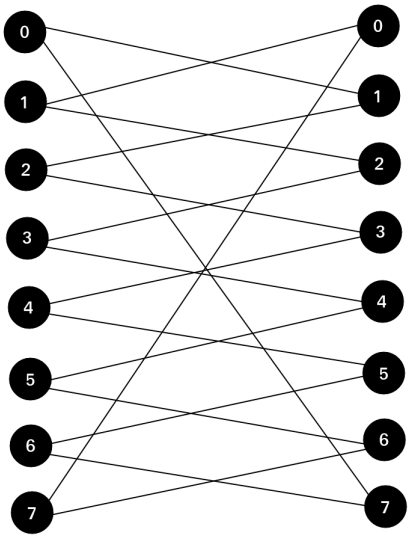
where $|\phi_i\rangle$ is the square-root of the i-th column of the transition matrix P. The projector operator Then is given by:

$$\Pi = \sum_{i=0}^{N-1} |\psi i\rangle\langle\psi i| \quad (10)$$

with the associated Reflection Operator:



(a) Figure A



(b) Figure B

Figure 6: Fig. A shows the cyclic graph, already used for the coined QW, Fig. B shows the correspondent bipartite graph constructed using the sets X and Y which contains the nodes of the cyclic graph edges connect only adjacent nodes of the cyclic graph.

$$\mathcal{R} = 2\Pi - I \tag{11}$$

The second operator needed is the Swap operator S given by:

$$\mathcal{S} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |i,j\rangle \langle j,i| \tag{12}$$

Finally the one-step Szegedy QW operator is given by:

$$U_{walk} = S(I - 2\Pi) = SR \tag{13}$$

This doesn't want to be a formal explanation, the schematized concepts above came from [9] and are just a summary helpful for the implementation. The concept behind the operator are covered with more details in [8, 9, 12], [11] from Pag. 224 and also [14] (which provides an example for a cyclic graph with 6 nodes), since [8, 9, 12] were a difficult reading for me this last two more theoretic references helped me understand some concepts and get the basic idea, especially [14].

7 Szegedy Quantum Walk Implementation

For the Implementation in qiskit I used as references [9] and [8], where I also found the generic circuit to implement the Szegedy QW on a cyclic graph.

As for the coined quantum walk, the circuit is said to be efficient if the one and two qubits gates used are no more than O(Poly(log(N))) with N number of nodes. While for S there are no problems R operator in this form cannot be implemented efficiently, [9] provide a way to implement it efficiently diagonalizing the operator R. Fig. 7 taken from that paper represent the generic circuit for the Szegedy QW on a C_N graph and shows how R is separated in order to realize the circuit efficiently.

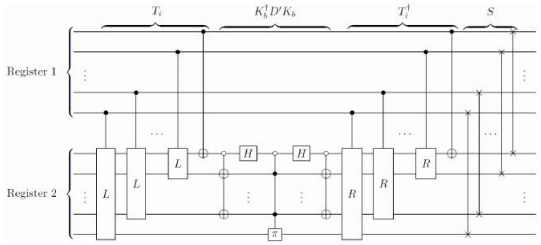


Figure 7: Generic Circuit for the Szegedy Quantum Walk on a Cycle graph with N nodes, this image is taken from

The L and R subcircuit are called respectively one-element left rotation and one-element right rotation, are used to perform a cyclic permutation of the vector given as input and are efficiently implementable requiring at most $log(N)$ qubits (a sequence of multi C-NOT). Swap are trivially efficiently implementable. In Fig. 8 below there is the specific circuit for the one-step Szegedy Quantum Walks on the cyclic graph C_8.

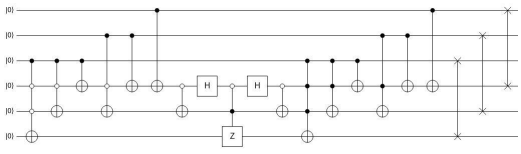


Figure 8: Circuit for the Szegedy QW on a cyclic graph

To make an example we can prepare the initial state of the walker as a superposition $|psi_0\rangle = |0\rangle \otimes |\phi_0\rangle$ where $|\phi 0\rangle = [0, frac1\sqrt{2}, 0, 0, 0, 0, 0, frac1\sqrt{2}]$ and we expect as output in the first register R_1 an equiprobability of the adjacent nodes.

Click this simulation in Quirk to see what mentioned above, again in the simulation multi controlled toffoli are specular since Quirk uses the little-endian convention.

7.1 Comparison with classical

The measure in which we compare those type of algorithm is in Hitting Time, this value in a stochastic process is the first time at which the given process "hits" a given set of sub-state. The famous paper of Szegedy [12] which is the starting point of this algorithm shows that in a certain condition we can achieve a quadratic speed-up. Unfortunately this paper was too hard for me to understand therefore i will not go deeper in this topic.

8 Applications

The methods presented so far are meant to be general and 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 applications that are difficult to solve randomized algorithm are used in context where an exact solution is too hard to be found or too expansive computationally speaking here come these algorithms to help. For instance very famous classical problems, like 3-SAT, or TSP currently are solved in many case using randomized algorithm, obtaining a speed-up over the Hitting Time would be an important achievement [6]. Another interesting application presented in [9] use the Szegedy Quantum Walk for the Page Rank algorithm: Quantum Page Rank algorithm. In [1] an hybrid approach classical-quantum is used for a linear solver algorithm.

9 Conclusions

This research presents the analogue of random walks, the quantum walks in discrete time with a focus on graph applications. We started from an example of Coined quantum walk on a cycle walk to show 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 performed an analysis of this method applied to the problem of search a marked vertex for some family of graphs thanks to [3] we know that quadratic speedup w.r.t. classical, using an efficient circuit. However the method presents some limitations, it works only for undirected and not weighted graph. The Szegedy quantum walk helps to get rid of limitations by quantizing the Markov chain related to the graph. This method can be efficiently implemented for the cycle graph.

References

[1] C.-C. Chen, S.-Y. Shiau, M.-F. Wu, and Y.-R. Wu. Hybrid classical-quantum linear solver using noisy intermediate-scale quantum machines. *Scientific Reports*, 9(1):16251, Nov 2019.

[2] B. L. Douglas and J. B. Wang. Efficient quantum circuit implementation of quantum walks, 2007.

[3] B. L. Douglas and J. B. Wang. Complexity analysis of quantum walk based search algorithms, 2014.

[4] H. García. *High-level quantum programming with quantum walks*. University of Michigan-Dearborn, 2007.

[5] A. Kay. Tutorial on the quantikz package, 2018.

[6] J. Kempe. Quantum random walks: An introductory overview. *Contemporary Physics*, 44(4):307–327, Jul 2003.

[7] V. Kendon and B. Tregenna. *Decoherence in Discrete Quantum Walks*, pages 253–267. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[8] O. Loke. *Quantum circuit design for quantum walks*. PhD thesis, The University of Western Australia, 2017.

[9] T. Loke and J. Wang. Efficient quantum circuits for szegedy quantum walks. *Annals of Physics*, 382:64–84, Jul 2017.

[10] M. A. Nielsen and I. L. Chuang. *Quantum circuits*, page 171–215. Cambridge University Press, 2010.

[11] R. Portugal. *Szegedy's Quantum Walk*, pages 223–246. Springer International Publishing, Cham, 2018.

[12] M. Szegedy. Quantum speed-up of markov chain based algorithms. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–41, 2004.

[13] S. Venegas-Andraca. *Quantum Walks for Computer Scientists*. 2008.

[14] T. G. Wong and R. A. M. Santos. Exceptional quantum walk search on the cycle. *Quantum Information Processing*, 16(6):154, May 2017.

Appendices

A Coined Quantum walk

This code below is referred to coined quantum walk, using Hadamard coin on a cycle graph with number of nodes N=8.

```
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
```

B Szegedy Quantum walk

This code below is referred to Szegedy quantum walk on a cycle graph with number of nodes N=8.

```
from qiskit import *

def szegedy(circuit):
    qr = circuit.qubits

    circuit.x([qr[5],qr[4], qr[3]]) #for the negative controlled gates
    circuit.barrier()

    #L^4
    circuit.mct([qr[4],qr[3], qr[2]], qr[5],None, mode='advanced')
    circuit.ccx(qr[2], qr[3], qr[4])
    circuit.cx(qr[2], qr[3])
    circuit.barrier()

    #L^2
    circuit.ccx(qr[2], qr[3], qr[4])
    circuit.cx(qr[2], qr[3])
    circuit.barrier()

    #L
    circuit.cx(qr[2], qr[3])
    circuit.barrier()
    circuit.x([qr[5],qr[4], qr[3]]) #for the negative controlled gates
    circuit.barrier()

    # multiple target negative controlled not
    circuit.x(qr[3])
    circuit.cx(qr[3], qr[4])
    circuit.x(qr[3])
    circuit.barrier()

    # multi controlled Z rotation
    circuit.h(qr[3])
    circuit.mcrz(180,[qr[4],qr[3]], qr[5])
    circuit.h(qr[3])
    circuit.barrier()

    # multiple target negative controlled not
    circuit.x(qr[3])
    circuit.cx(qr[3], qr[4])
    circuit.x(qr[3])
    circuit.barrier()

    #R^4
    circuit.mct([qr[4],qr[3], qr[2]], qr[5],None, mode='advanced')
    circuit.ccx(qr[2], qr[3], qr[4])
    circuit.cx(qr[2], qr[3])
    circuit.barrier()

    #R^2
    circuit.ccx(qr[2], qr[3], qr[4])
    circuit.cx(qr[2], qr[3])
    circuit.barrier()

    #R
    circuit.cx(qr[2], qr[3])
    circuit.barrier()

    # swap operator
    circuit.swap(qr[5], qr[2])
    circuit.swap(qr[4], qr[1])
    circuit.swap(qr[3], qr[0])

# circuit preparation
qr = QuantumRegister(6)
cr = ClassicalRegister(3)

circuit = QuantumCircuit(qr, cr)

# state preparation for the Cyclic graph
circuit.h(qr[3])
circuit.cx(qr[3], qr[4])
circuit.x(qr[5])
circuit.barrier()
szegedy(circuit)
```