J. Comp. Int. Sci. (2015) 6(1):21-29
http://epacis.net/jcis/PDF_JCIS/JCIS-0092.pdf
jcis@epacis.net

Pan-american Associacion of Computational Interdisciplinary Sciences

Simulation of Quantum Walks using HPC

Pedro C. S. Lara[a], Aaron Leão[b], Renato Portugal[b1]

[a]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, Petrópolis, RJ, 25620-003, Brazil.
[b]Laboratório Nacional de Computação Científica, Petrópolis, RJ, 25651-075, Brazil.

Received on March 20, 2015 / Accepted on April 30, 2015

Abstract

We describe program Hiperwalk, which is a new simulator of the main quantum walk models using high-performance computing (HPC). The simulator is able to generate the dynamics of discrete-time quantum walks and staggered quantum walks, and will be able simulate continuous-time quantum walks and Szegedy's quantum walks. It has an user-friendly input and is able to use hybrid HPC architectures, which includes the main ones available nowadays. The simulator outputs the main statistics associated with the probability distribution of the quantum walk in data files and automatically generates plots. The simulator uses open-source non-proprietary codes and employs freeware languages: Python, Neblina, Gnuplot, and OpenCL.

Keywords: *quantum walks, HPC, simulation, parallel programming languages, quantum computing, quantum algorithms.*

1. Introduction

The discrete-time quantum walk (DTQW) model on the line was introduced by Aharonov *et al.* [4] and it was generalized to regular graphs in Ref. [3]. In the DTQW, the particle hops from site to site depending on the value of an internal degree of freedom, which plays the role of the coin. Quantum walks on N-dimensional lattices were studied by many authors [20, 32, 24] and display the key feature of spreading quadratically faster in terms of probability distribution, compared to the classical random walk model on the same underlying structure [5]. The DTQW was successfully applied to develop quantum algorithms, specially for searching a marked node in graphs [30, 8, 26]. There are other models of quantum walks and some of them do not use an auxiliary Hilbert space and have no coin. The continuous-time quantum walk model introduced by Farhi and Gutman [16] and the staggered (or coinless) quantum walk model formally defined in [27] are examples of such models. The staggered model can be used to search a marked node on two-dimensional finite lattices with the same number of steps (asymptotically in terms of the system size) compared to the coined model, with the advantage of using a smaller Hilbert space [9]. Going to the opposite direction, Szegedy's quantum walk model [31] use the largest Hilbert space the dimension of which is the square of the dimension of the staggered model.

In this work, we describe a new high-performance quantum-walk simulator, called Hiperwalk[2], that can be used to obtain the dynamical evolution of the quantum walk models and can be used to calculate the main statistical distributions associated with the probability distribution. The simulator has three main parts: 1) an user interface built in Python that generate matrices and vectors based on the input to be used

[1]E-mail of Corresponding Author: portugal@lncc.br
[2]Site `http://qubit.lncc.br/qwalk` has downloading and installation instructions.

in the core of the program, 2) a core program written in the Neblina language (developed by one of us) which is able to run matrix calculations in heterogeneous HPC architectures, 3) a module that calculates the main statistical distribution and generates data files, output files and plots. In the present version, only the DTQW and the coinless (or staggered) models were implemented and described.

To the best of our knowledge, Hiperwalk is the only simulator of its kind. Other kind of QW simulators are pyCTQW for continuous-time quantum walk [18], qwViz for visualisation of quantum walks on graphs [10], and QWalk for simulation of DTQW on one- and two-dimensional lattices [22]. A version of QWalk using HPC was described in Ref. [29].

2. Quantum-Walk Models

The goal of the new simulator is to generate the dynamics of the main quantum-walk (QW) models known in Literature in a generic graph. In this Section we briefly describe each one pointing out the main references. There are four models: 1) Discrete-Time Quantum Walk (DTQW), 2) Continuous-Time Quantum Walk (CTQW), 3) Szegedy's Quantum Walk, and 4) Staggered Quantum-Walk.

The Discrete-Time Quantum Walks were the first to be proposed [4] and it was successfully used to solve the problem of quantum spatial search on lattices and hypercubes. Shenvi et al. [30] developed a quantum search algorithm for the hypercube with time complexity $O(\sqrt{N})$, where N is the number of vertices of the graph. This result has a quadratic gain over the corresponding classical algorithm. Ambainis et al. (AKR) [8] used a similar method to develop a quantum search algorithm in two-dimensional lattices in time $O(\sqrt{N}\log N)$ almost quadratically better then the time $O(N\log N)$ of the classical algorithm. Tulsi [33] introduced an extra qubit in the system and improved the complexity of the AKR algorithm. Ambainis et al. (ABNOR) [6] also showed how to eliminate amplitude amplification method used in the AKR algorithm by doing post-processing calculations. Moreover, Refs. [2, 17, 1] describe similar methods to many other graphs generating new efficient quantum algorithms.

The concept of Continuous-Time Quantum Walks were proposed by [16]. One of the first applications was the NAND-tree evaluation in time $O(\sqrt{N})$ developed in Ref [15]. Discrete versions of the latter were presented in [11, 7, 12] requiring time $N^{\frac{1}{2}+O(1)}$. The evaluation of minimax trees using $N^{\frac{1}{2}+O(1)}$ oracle queries was discussed in [13].

Szegedy's quantum walk model presents general results for the spatial search problem in broad classes of graphs. Szegedy [31] showed that the quantum hitting time has a quadratic gain compared to the classical hitting time for the problem of detecting whether a set of vertices is marked, in connected, regular and non-bipartite graphs. Santos and Portugal [28] analyzed the search problem on the complete graph and on the cycle. With different proposals, Magniez [21] and Krovi et al. [19] developed quadratically faster quantum algorithms to find a marked vertex on ergodic and reversible Markov chains.

The Staggered Quantum-Walk model was discussed in some early references, such as [23, 25, 14] and the application for searching on lattices was performed numerically. Ambainis, Portugal and Nahimov [9] analytically proved that the staggered quantum walk in those lattices, using the same tessellation proposed by Falk [14] has complexity $O(\sqrt{N}\log N)$. This result has the same complexity of the algorithm that uses quantum walks with coin. However, it is important to note that the staggered quantum walk needs no additional space and therefore this algorithm uses minimal memory. Here, we use Falk's method to implement this model. The evolution operator is defined by a graph tessellation. The simplest line or even-cycle tessellation is based on two sets of orthonormal vectors

$$\left|u_x^0\right\rangle = \cos\frac{\alpha}{2}\left|2x\right\rangle + \mathrm{e}^{i\phi_1}\sin\frac{\alpha}{2}\left|2x+1\right\rangle, \tag{1}$$

$$\left|u_x^1\right\rangle = \cos\frac{\beta}{2}\left|2x+1\right\rangle + \mathrm{e}^{i\phi_2}\sin\frac{\beta}{2}\left|2x+2\right\rangle, \tag{2}$$

which are used to define the reflection operators

$$U_0 = 2\sum_{x=-\infty}^{\infty} |u_x^0\rangle\langle u_x^0| - I, \quad (3)$$

$$U_1 = 2\sum_{x=-\infty}^{\infty} |u_x^1\rangle\langle u_x^1| - I. \quad (4)$$

For a N-cycle (N must be even), index x runs from 0 to $N-1$. One step of the quantum walk is driven by the unitary operator $U = U_1U_0$.

3. Simulation using HPC

The notable progress of GP-GPU architectures in recent years and the emergence of many other accelerator architectures like FPGA, IBM's Cell and Intel's MIC (Xeon Phi) indicate a strong tendency towards the heterogeneity in HPC. In this context, we use the Neblina[3] language to simulate, using parallel resources, QW models.

Neblina is a language focused on establishing a parallel computing layer requiring minimal knowledge of the user about parallel programming. For programming in Neblina, the operations are sequential independently of the underlying architecture. The Neblina interpreter sends the data to the parallel processing unit (either CPU or GPU or other) increasing productivity. This is transparently performed by using OpenCL paralell API, which accesses heterogeneous architectures. The simulation of QWs uses highly scalable matrix-matrix and matrix-vector operations that allow speedups when we use Neblina as the core program. We are interested also to obtain the statistical data of the dynamics (standard deviation, limiting distribution, mixing time and so on) of QW models. The simulation and the calculation of those metrics can generate large overheads in the overall processing time. Besides, some theoretical results can only be confirmed by performing simulations with large number of elements (asymptotic behavior).

Our simulation of QWs goes through three distinct major steps:

- Conversion of the input parameters into unitary operator U, which describes the dynamical evolution. These parameters can be the underlying structure (line, lattices, cycles and others), coin operator, initial states, and so on.
- Application of unitary operator U in a quantum state $|\psi\rangle$ generating vector $U^i|\psi\rangle$ for $i = \{1, \ldots, T\}$ for large values of T.
- Statistical interpretation of $U^i|\psi\rangle$ for $i = \{1, \ldots, T\}$. This includes the probability distribution, standard deviation, limiting distribution, and so on.

For the first and last steps we use the Python programming language that allows a high abstraction and complex data structures facilitating the process of converting input parameters into matrices and vectors. Second step is certainly the most costly one. In this step we use the OpenCL language to parallelize the application of matrix U on the quantum state $|\psi\rangle$ (a vector). So, this part can be performed in a GP-GPU or in a CPU multicore using the same program code.

In addition, our simulator generates understandable output data by creating automatically gnuplot sripts.

4. Hiperwalk: High-Performance Quantum-Walk Simulator

Hiperwalk[4] is a freeware open-source program, available in GitHub[5], that allows the user to perform simulations of quantum walks on graphs using HPC. The user can use the parallel resources of the computer, such as accelerator cards, multicore CPU and GPGPU to speedup the overall process without knowing

[3] http://www.lncc.br/∼pcslara/neblina

[4] Site `http://qubit.lncc.br/qwalk` has downloading and installation instructions.

[5] `https://github.com/`

parallel programming. It is under development and employs Python, OpenCL, Neblina[6], and Gnuplot languages.

In the current version of Hiperwalk, the input text file is the only interface between the user and the simulator. The main input commands are:

`WALK <MODEL>` selects a quantum walk model. Currently it can be `DTQW`, `STAGGERED`, `CUSTOM`. This is a required command.

`DIRECTORY <NAME>` defines a directory path which is used by the simulator to save the output files. This is a required command. `<NAME>` must not have SPACE of TABULAR character.

`GRAPH <TYPE> <SIZE>` defines the graph for the walk. This is a required command.
`<TYPE>` can be `LINE`, `CYCLE`, `LATTICE`, `TORUS`.
`<SIZE>` is the number of vertices. It must be a positive integer or nonexistent (for infinite graphs).

`STEPS <T>` defines the number of steps until which the system will evolve. `<T>` must be a positive integer. This is a required command.

`ALLSTATES <N>` forces the simulator to save the states for all time steps are multiple of `N`. If `N`=1, all intermediate states will be saved. This keyword changes the number of points used in the statistics files. This parameter is optional. The default is to save only the last step. The value of `N` must be a positive integer ($\texttt{N} \geq 1$).

`PLOTS TRUE` forces the simulator to generate the graphics of the mean and the standard deviation using Gnuplot. This command can be suppressed. The default is `FALSE`, which avoids to create the graphics.

`PLOTZEROS TRUE` forces the simulator to save and print probabilities that are exactly zero. This command can be suppressed. The default is `FALSE`, which avoids to save and plot zeros.

`ANIMATION TRUE` forces the simulator to save data and make a plot of the probability distribution at each time step. At the end, the simulator generates an animation file called *evolution.gif*. This command can be suppressed. The default is `FALSE`, which avoids to create the animation.

`VARIANCE TRUE` forces the simulater to plot the variance. The default is `FALSE`.

`HARDWAREID <N>` forces the simulator to use the Nth processor unit (parallel device), which is described by command `neblina -l`. The default value of `N` is 0. The value of `N` must be a non-negative integer.

Comments can be introduced by putting the character `#` in the beginning of the each line.

4.1 Commands only for DTQW

`BEGINSTATE ··· ENDSTATE`
This block defines the initial state of the quantum walk. For the coined case, the initial state has the form $|\psi\rangle = \sum_i \alpha_i |c_i\rangle|p_i\rangle$, where $\alpha_i \in \mathbb{C}$, $|c_i\rangle$ is a coin state, and $|p_i\rangle$ is a position state. Each term in the sum must be entered in a line as follow:

$\Re(\alpha_i)$ $\Im(\alpha_i)$ c_i p_i

For example, state $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)|0\rangle$ corresponds to

```
BEGINSTATE
0.70710678  0  0  0
0  0.70710678  1  0
ENDSTATE
```

[6]Site `http://qubit.lncc.br/neblina` can be used to download the Neblina programming language.

`BEGINCOIN ··· ENDCOIN`
This block defines the coin operator. The options are `HADAMARD <N>`, `FOURIER <N>`, `GROVER <N>`, which produces the Hadamard, Fourier, or Grover operators (dimension `N`×`N`). A customized coin matrix can be defined by inputing each entry $a_{ij} + i\,b_{ij}$ as follows:

`BEGINCOIN`
a_{11} b_{11} $\dots$ a_{1N} b_{1N}
$\vdots$ $\ddots$ $\vdots$
a_{N1} b_{N1} $\dots$ a_{NN} b_{NN}
`ENDCOIN`

4.2 Commands only for STAGGERED

One-dimensional staggered walks:

`BEGINSTATE ··· ENDSTATE`
This block defines the initial state of the quantum walk. For the staggered case, the initial state has the form $|\psi\rangle = \sum_x \alpha_x |x\rangle$. Each term in the sum produces a line with the syntax

$\Re(\alpha_x)$ $\Im(\alpha_x)$ x

where $\alpha_x \in \mathbb{C}$ are the amplitudes and $|x\rangle$ is a position state in the computational basis. For example, state $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$ corresponds to

`BEGINSTATE`
0.70710678 0 0
0 -0.70710678 1
`ENDSTATE`

`POLYGONS <N>` defines the number of vertices inside the polygons of both tessellations. `<N>` must be a positive integer.

`DISPLACEMENT <N>` defines the displacement of the second tessellation toward positive x. `<N>` must be a non-negative integer.

`BEGINTESSELLATION`
$\Re(a_1)$ $\Im(a_1)$ $\dots$ $\Re(a_N)$ $\Im(a_N)$
$\Re(b_1)$ $\Im(b_1)$ $\dots$ $\Re(b_N)$ $\Im(b_N)$
`ENDTESSELLATION`
For `CYCLE` or `LINE`, amplitudes a_1 to a_N, where `N` is the number of vertices in a polygon, define the unitary operator for the first tessellation of the one-dimensional `STAGGERED` model. Amplitudes b_1 to b_N define the unitary operator for the second tessellation. See examples for more details.

Two-dimensional staggered walks:

`BEGINSTATE ··· ENDSTATE`
The initial state has the form $|\psi\rangle = \sum_{xy} \alpha_{xy} |x, y\rangle$. Each term in the sum produces a line with the syntax

$\mathrm{re}(\alpha_{xy})$ $\mathrm{im}(\alpha_{xy})$ x y

where $\alpha_{xy} \in \mathbb{C}$ are the amplitudes and $|x, y\rangle$ is a position state.

`POLYGONS <Nx> <Ny>` defines the dimensions of the polygons of both tessellations. `<Nx>` and `<Ny>` must be a positive integers.

`DISPLACEMENT <Nx> <Ny>` defines the displacement of the second tessellation. `<Nx>` moves toward increasing x and `<Ny>` moves toward increasing y. `<Nx>` and `<Ny>` must be a non-negative integers.

BEGINTESSELLATION
$\Re(a_{11})$ $\Im(a_{11})$ ... $\Re(a_{1\,\mathtt{Ny}})$ $\Im(a_{1\,\mathtt{Ny}})$ ... $\Re(a_{\mathtt{Nx}\,\mathtt{Ny}})$ $\Im(a_{\mathtt{Nx}\,\mathtt{Ny}})$
$\Re(b_{11})$ $\Im(b_{11})$ ... $\Re(b_{1\,\mathtt{Ny}})$ $\Im(b_{1\,\mathtt{Ny}})$ ... $\Re(b_{\mathtt{Nx}\,\mathtt{Ny}})$ $\Im(b_{\mathtt{Nx}\,\mathtt{Ny}})$
ENDTESSELLATION

For TORUS or LATTICE, parameters a_{11} to $a_{\mathtt{Nx}\,\mathtt{Ny}}$ define the unitary operator for the first tessellation of the two-dimensional STAGGERED model. Parameters b_{11} to $b_{\mathtt{Nx}\,\mathtt{Ny}}$ define the unitary operator for the second tessellation.

4.3 Commands only for CUSTOM

INITIALSTATE <filename> expects the file name describing the initial condition. The initial condition must be in the computational basis ($|\psi\rangle = \sum_i \alpha_i |i\rangle$) and the i-th line of <filename> must have the i-th amplitude in the format

$\mathrm{re}(\alpha_i)$ $\mathrm{im}(\alpha_i)$

UNITARY <filename1> <filename2> ... expects the file names describing unitary operators. The operators must be stored in the sparse form. If

$$U = \sum_{i,j} u_{i,j} |i\rangle\langle j|,$$

is a unitary operator, the format of each line of <filename> is

i j $\mathrm{re}(u_{i,j})$ $\mathrm{im}(u_{i,j})$

Entries that are zero must no be stored and it does not matter the order of the lines. CUSTOM is used for generating the state at step t when the inital state and unitaries U_1, U_2, and so on are given. That is, Hiperwalk calculates $(U_2 U_1)^t |\psi_0\rangle$.

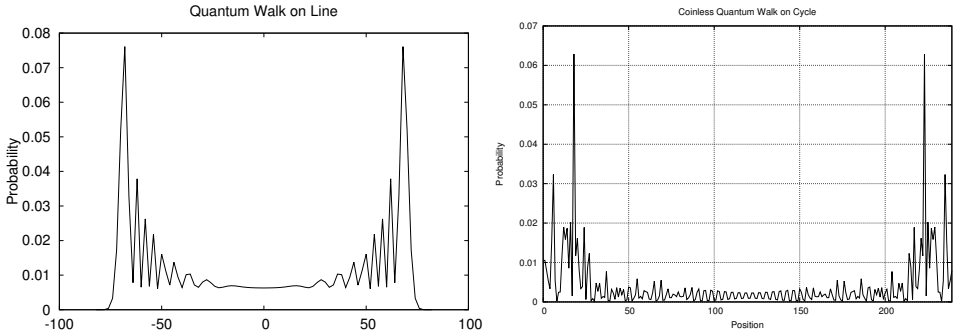
5. EXAMPLES

In this section we show two examples using the coined and staggered quantum walk models. Consider two input samples. The left (right) input is an example for the coined (staggered) model.

```
1    WALK DTQW
2    DIRECTORY DIR1
3    STEPS 100
4    GRAPH LINE
5
6    BEGINSTATE
7    0.70710678 0 0 0
8    0 -0.70710678 1 0
9    ENDSTATE
10
11   BEGINCOIN
12   HADAMARD 2
13   ENDCOIN
14
15   PLOTS TRUE
16
17
18
19
```

```
1    WALK STAGGERED
2    DIRECTORY DIR2
3    STEPS 100
4    GRAPH CYCLE 240
5
6    BEGINSTATE
7    0.70710678 0 120
8    0 0.70710678 121
9    END STATE
10
11   POLYGONS 2
12   DISPLACEMENT 1
13
14   BEGINTESSELLATION
15   0.92387953 0 0.3826834 0
16   0.50000000 0 0.8660254 0
17   ENDTESSELLATION
18
19   PLOTS TRUE
```

The left column shows an input sample using the coined quantum walk on a line. The initial condition is $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)|0\rangle$. The coin is the 2×2 Hadamard matrix. The right column shows an input sample using the staggered quantum walk on a cyle with 240 vertices. The polygons are of 2×1 vertices. The displacement of the second tessellation is of one vertex. The tessellation is set with amplitudes 0.92387953 for the first vertex and 0.3826834 for the second vertex, while, the second tessellation has amplitudes 0.50000000 and 0.8660254 for the first and second vertices, respectively, and the initial condition is $|\psi\rangle = \frac{1}{\sqrt{2}}(|120\rangle + i|121\rangle)$.



(a) Coined (b) Coinless

Figure 1: Probability distributions after 100 steps for the coined model (left) and after 50 step for the staggered model (right).

Fig. 1 shows the final probability distribution for each case generated by Hiperwalk. The simulator also produces a graph of the standard deviation. Fig. 2 shows the standard deviation as a function of the number of steps for left-side input (coined model). This figure uses the gnuplot fitting to find the scaling parameters of the standard deviation. From the result, one can easily verify that the standard deviation increases linearly with the number of time steps, as expected.

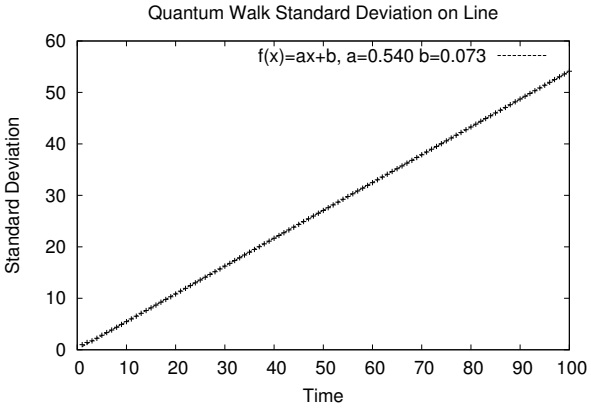


Figure 2: Standard deviation for coined model experiment.

6. CONCLUSIONS

We have briefly described a new simulator called Hiperwalk (high-performance quantum-walk simulator), that can be used to generate the dynamics of the main quantum walk models described in Literature. Up to now, only the discrete-time coined and staggered quantum walk models have been implemented. The continuous-time and Szegedy's quantum walk models are currently being implemented.

This new simulator has two major advantages: 1) it has an user-friendly input, and 2) it is able to use hybrid HPC architectures, which include Xeon processors, Xeon Phi, Tesla cards, graphic cards, intel multi-cores, and others. The load is automatically balanced among the available processors and, to use the simulator, no knowledge of parallel programming is required.

Acknowledgments: We thank CNPq, Capes and Faperj for finantial support. We acknowledge the use of ComCiDis HPC structure.

References

[1] G. Abal, R. Donangelo, M. Forets, and R. Portugal. Spatial quantum search in a triangular network. *Mathematical Structures in Computer Science*, 22:1–11, 2012.

[2] G. Abal, R. Donangelo, F. L. Marquezino, and R. Portugal. Spatial search on a honeycomb network. *Mathematical Structures in Computer Science*, 20:999–1009, 2010.

[3] D. Aharonov, A. Ambainis, J. Kempe, and U. Vazirani. Quantum walks on graphs. In *Proceedings of the 33rd ACM Symposium on Theory of computing*, pages 50–59, 2000. `arXiv:quant-ph/0012090`.

[4] Y. Aharonov, L. Davidovich, and N. Zagury. Quantum random walks. *Physical Review A*, 48(2):1687–1690, 1993.

[5] D. Aldous and J. Fill. *Reversible Markov Chains and Random Walks on Graphs*. Monography in preparation, 1994. `http://www.stat.berkeley.edu/\textasciitildealdous/RWG/book.html`.

[6] A. Ambainis, A. Backurs, N. Nahimov, R. Ozols, and A. Rivosh. Search by quantum walks on two-dimensional grids without amplitude amplification. arvix:quant-ph/1112.3337, 2011.

[7] A. Ambainis, A. Childs, B. Reichardt, R. Spalek, and S. Zhang. Any and-or formula of size n can be evaluated in time $n^{1/2+o(1)}$ on a quantum computer. In *Proceedings of FOCS*, pages 363–372, 2007.

[8] A. Ambainis, J. Kempe, and A. Rivosh. Coins make quantum walks faster. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms*, pages 1099–1108, 2005.

[9] A. Ambainis, R. Portugal, and N. Nahimov. Spatial search on grids with minimum memory. arXiv:quant-ph/1312.0172, 2013.

[10] S. D. Berry, P. Bourke, and J. B. Wang. qwViz: Visualisation of quantum walks on graphs. *Computer Physics Communications*, 182(10):2295 – 2302, 2011.

[11] A. Childs, B. Reichardt, R. Spalek, and S. Zhang. Every nand formula on n variables can be evaluated in time $o(n^{1/2+\epsilon})$. Version3, arXiv:quant-ph/0703015, 2007.

[12] A. M. Childs, R. Cleve, S. P. Jordan, and D. Yonge-Mallo. Discrete-query quantum algorithm for nand trees. *Theory of Computing*, 5(1):119–123, 2009.

[13] R. Cleve, D. Gavinsky, and D. Yonge-Mallo. Quantum algorithms for evaluating min-max trees. In Y. Kawano and M. Mosca, editors, *Proceedings of Theory of Quantum Computation, Communication, and Cryptography (TQC 2008)*, volume 5106, pages 11–15. Springer Berlin / Heidelberg, 2008.

[14] M. Falk. Quantum search on the spatial grid. arxiv:quant-ph/1303.4127, 2013.

[15] E. Farhi, J. Goldstone, and S. Gutmann. A quantum algorithm for the hamiltonian nand tree. *Theory of Computing*, 4(1):169–190, 2008.

[16] E. Farhi and S. Gutmann. Quantum computation and decision trees. *Physical Review A*, 58:915–928, 1998.

[17] B. Hein and G. Tanner. Quantum search algorithms on a regular lattice. *Physical Review A*, 82(1)(012326), 2010.

[18] J. A. Izaac and J. B. Wang. pyCTQW: A continuous-time quantum walk simulator on distributed memory computers. *Computer Physics Communications*, 186:81 – 92, 2015.

[19] H. Krovi, F. Magniez, M. Ozols, and J. Roland. Finding is as easy as detecting for quantum walks. In *Proceedings of the 37th International Colloquium Conference on Automata, Languages and Programming*, pages 540–551, 2010.

[20] T. D. Mackay, S. D. Bartlett, L. T. Stephenson, and B. C. Sanders. Quantum walks in higher dimensions. *Journal of Physics A: Mathematical and General*, 35(12):2745, 2002.

[21] F. Magniez, A. Nayak, P. C. Richter, and M. Santha. On the hitting times of quantum versus random walks. In *Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 86–95, 2009.

[22] F. L. Marquezino and R. Portugal. The qwalk simulator of quantum walks. *Computer Physics Communications*, 179:359–369, 2008.

[23] D. Meyer. From quantum cellular automata to quantum lattice gases. *Journal of Statistical Physics*, 85(5):551–574, 12 1996.

[24] A. C. Oliveira, R. Portugal, and R. Donangelo. Decoherence in two-dimensional quantum walks. *Physical Review A*, 74(012312), 2006.

[25] A. Patel, K. Raghunathan, and P. Rungta. Quantum random walks do not need a coin toss. *Phys. Rev. A*, 71:032347, 2005.

[26] R. Portugal. *Quantum walks and search algorithms*. Springer, New York, 2013.

[27] R. Portugal, R. Santos, T. Fernandes, and D. Gonçalves. The staggered quantum walk model. *Quantum Information Processing (accepted), arXiv:1505.04761*, 2015.

[28] R. A. M. Santos and R. Portugal. Quantum hitting time on the complete graph. *International Journal of Quantum Information*, 8(5):881–894, 2010.

[29] M. Sawerwain and R. Gielerak. Gpgpu based simulations for one and two dimensional quantum walks. In A. Kwiecie, P. Gaj, and P. Stera, editors, *Computer Networks*, volume 79 of *Communications in Computer and Information Science*, pages 29–38. Springer Berlin Heidelberg, 2010.

[30] N. Shenvi, J. Kempe, and K. B. Whaley. A quantum random walk search algorithm. *Physical Review A*, 67(052307), 2003.

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

[32] B. Tregenna, W. Flanagan, R. Maile, and V. Kendon. Controlling discrete quantum walks: coins and initial states. *New Journal of Physics*, 5(1):83, 2003, quant-ph/0304204.

[33] A. Tulsi. Faster quantum walk algorithm for the two dimensional spatial search. *Physical Review A*, 78(012310), 2008.