User's Manual

Version 1.0

http://qubit.lncc.br/qwalk

HiperWalk

High-Performance Quantum Walk Simulator

— ○ —

Chapter 1

Installation

1.1 About HiperWalk

Hiperwalk is a freeware open-source program that allows the user to perform simulations of quantum walks (QWs) on graphs using HPC. The user can employ the parallel resources of the computer, such as accelerator cards, multicore CPU and GPGPU to speedup the overall process without knowing parallel programming. It uses Python, OpenCL, Neblina, and Gnuplot languages. The simulator outputs the main statistics of the probability distribution associated to the quantum walk in data files and automatically generates plots. The current version addresses the discrete-time QW versions: coined QW (DTQW), the staggered (or coinless) QW, and Szegedy's QW. Continuous-time QW will be available in future versions.

To use Hiperwalk, the user needs to install Python 2.7.x (do not install version 3.2.x), numpy, gnuplot, Neblina, and OpenCL. Hiperwalk is being developed in Linux (Ubuntu, Fedora, and CentOS).

1.2 Installing Python 2.7

Python is native to Linux. The user can check the current version with command

```
$ python --version
```

which should return Python 2.7.x where x is a number. If the version is not the correct one, the user must get the correct version at `https://www.python.org/`. Follow the instructions in that web page, download the program, decompress the tar file, go the directory, and type

```
$ ./configure
$ make
$ sudo make install
```

The user can check the installation by issuing command

```
$ python
```

which must open Python's environment:

```
>>> quit()
```

Command quit() quits Python.

1.3 Installing Numpy

Numpy is a library that allows numerical calculations in Python. It is not native. The user must go to site `http://www.numpy.org/`, download Numpy, and must follow the instructions described in that webpage. The procedure depends on which Linux distribution the user is employing. In some cases, the implementation can be tricky.

In general, to install Numpy the user must download Numpy, untar the downloaded file, go to the numpy folder and install. In Ubuntu, the user skips all previous directions and simply issue command

```
$ sudo apt-get install python-numpy
```

The user can check whether the installation was successful by importing numpy inside python (after changing folder)

```
$ python
>>> import numpy
```

No error message can be printed.

1.4 Installing NetworkX

NetworkX is a library that allows graph manipulations in Python (useful for custom and staggered models). It is not native. The user must go to site `https://pypi.python.org/pypi/networkx/`, download last version of NetworkX, and must follow the instructions described in NetworkX's webpages or download file `networkx-1.11.zip`, extract the directory, enter in the new folder, and execute the following command

```
$ sudo python setup.py install
```

The user can check whether the installation was successful by importing networkx inside python

```
$ python
>>> import networkx
```

No error message can be printed.

1.5 Installing Gnuplot

Gnuplot is a graphing utility for Linux and other platforms created to allow visualization of mathematical functions and data. It can be installed by issuing command

```
$ sudo apt-get install gnuplot
```

Hiperwalk also need command *convert* in the Linux package *imagemagick*[1]. It can be installed by issuing command

```
$ sudo apt-get install imagemagick
```

1.6 Neblina

Neblina[2] is a programming language for matrix-matrix and matrix-vector calculations that allows the user to employ parallel heterogeneous architectures with minimum knowledge of parallel programming. Neblina is able to access GP-GPU cards, Tesla K80/K40/K20/K10, many-core CPUs, Xeon Phi cards[3] and other parallel hardwares of the machine. Neblina needs OpenCL.

[1] `http://www.imagemagick.org/`

[2] `http://qubit.lncc.br/neblina` or `http://www.lncc.br/~pcslara/neblina`

[3] Neblina does not have a good speedup on Xeon Phi coprocessors compared with the speedup on Tesla cards, AMD GPU cards, and others.

1.6.1 Installing OpenCL

OpenCL is a framework for writing programs that execute across heterogeneous platforms providing parallel computing using task-based and data-based parallelism.

We recommend to install AMD OpenCL, which can be found at the AMD site[4]. The user must select the version that corresponds to the user's machine. After downloading, the user must unpack and install by issuing commands

```
$ tar -xvf <file_name>.tar.bz2
$ sudo ./<file_name>.sh
```

where `<file_name>` is the name of downloaded file. In this intallation we use the 64-bit Linux version, which is AMD-APP-SDK-linux-v2.9-1.599.381-GA-x64. The next command works only for this version. For other versions use the installation notes provided in the AMD web page.

The user needs to link the AMD library with Linux by issuing command

```
$ sudo ln -s /opt/AMDAPPSDK-2.9-1/lib/x86_64/libOpenCL.so
   /usr/lib/x86_64-linux-gnu/libOpenCL.so
```

The user must re-open the terminal. Many things can go wrong because the installation of OpenCL is tricky. The user may get help from us by email[5].

1.6.2 Installing Neblina

To install Neblina, go to `http://www.lncc.br/~pcslara/neblina` and download the current version. OpenCL must be working correctly. The user must issue commands

```
$ tar -xvf neblina-<VERSION>.tar.gz
$ cd neblina-<VERSION>
$ ./configure
$ make
$ sudo make install
```

To list the available OpenCL devices issue command

```
$ neblina -l
```

The id numbers returned by this command can be used as an input parameter inside Hiperwalk to force the use of a specific OpenCL device. The device with id: 0 is the default.

If the machine has GPU graphic cards and they were not listed after above command, the user must install the vendor OpenCL library, the card driver, and verify whether the card was installed correctly.

1.7 Installing Git

Git is a program to manage versions and backups of software. To install Git, issue command

```
$ sudo apt-get install git
```

It is going to be used to install Hiperwalk.

[4] `http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/`
[5] Pedro (pedro.lara@cefet-rj.br)

1.8 Installing HiperWalk

Hiperwalk will be installed in folder `hpwalk` and will erase any file or folder inside it. To install Hiperwalk from the user's home folder, issue commands

```
$ git clone https://github.com/hiperwalk/hpwalk.git
$ cd hpwalk
$ sudo ./install.sh
```

Close and reopen the terminal. Command `git` creates folder `hpwalk` and downloads the source code. The installation moves the source code to folder `/usr/local/hiperwalk` and adds this new folder to the path variable. Folder `hpwalk` will have only examples, which are files with extension `.in` or `.dat`. Folder `hpwalk` can be used as a workplace as a starting point. The safest procedure is to move the examples to a new folder (called `hiperwalk`, for instance). As a quick test, go to folder `hpwalk` and issue command

```
$ hiperwalk coined1D.in
```

No error is expected. A new folder called `DIR_DTQW1D` will be created with the output data files. The user can also test the installation by issuing command

```
$ hiperwalk -test
```

which must return only OKs.

Non-sudoer users must use the longer local form

```
$ python hiperwalk.py ./examples/coined1D.in
```

which runs Python loading Hiperwalk and saves the results in folder `./examples/DIR_DTQW1D`.

Chapter 2

Using HiperWalk

To use Hiperwalk after completing installation, the user must issue command

```
$ hiperwalk <input_file>.in
```

The results will be stored in a folder specified in `<input_file>.in`. There are some prepared input files, which can be easily found by the extension (`.in`) in folder `hpwalk`. In the current version of Hiperwalk, the input file is the main interface between the user and the simulator. The user can select a prepared input file and modify it following the correct syntax described in the next section.

2.1 Input File Syntax

The input commands used by all models are:

`WALK <MODEL>` selects a quantum walk model. Currently it can be `DTQW`, `STAGGERED`, `SZEGEDY`, `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 that 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 ($N \geq 1$).

`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.

2.1.1 Commands only for DTQW

`BEGINSTATE` $\cdots$ `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
```

`BEGINCOIN` $\cdots$ `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`

$$\begin{matrix} a_{11} & b_{11} & \dots & a_{1N} & b_{1N} \\ \vdots & & \ddots & & \vdots \\ a_{N1} & b_{N1} & \dots & a_{NN} & b_{NN} \end{matrix}$$

`ENDCOIN`

2.1.2 Commands only for STAGGERED

One-dimensional staggered walks:

`BEGINSTATE` $\cdots$ `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`

$\Re(a_1)$ $\Im(a_1)$ ... $\Re(a_N)$ $\Im(a_N)$
$\Re(b_1)$ $\Im(b_1)$ ... $\Re(b_N)$ $\Im(b_N)$

`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. See examples for more details.

2.1.3 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 need not to be stored, it does not matter the order of the lines, and the range of i, j is 1 to N. See examples for more details.

`ALLPROBS <N>` forces the simulator to save the probability distribution for all time steps that are multiple of `N`. If `N`=1, all intermediate probability distributions will be saved. This parameter is optional. The default is to save only the last step. The value of `N` must be a positive integer ($\mathtt{N} \geq 1$).

`ADJMATRIX <filename>` forces the simulator to load the adjacency matrix of a graph from `<filename>`. The adjacency matrix must be stored in the sparse form. The format of each line of `<filename>` is

i j 1

Entries that are zero need not to be stored, it does not matter the order of the lines, and the range of i, j is 1 to N. See examples for more details. The program will compute the distance vector based on the initial condition, mean, variance, 2nd moment, and the standard deviation. Those quantities will be stored in file `statistics.dat`.

2.2 Output Data Files

The list of output files with the presence of command `PLOTS TRUE` and command `ANIMATION TRUE` is

`final_state.dat`	`final_distribution.dat`	`statistics.dat`
`mean.eps`	`final_distribution.eps`	`standard_deviation.eps`
`wavefunction-i.dat`	`probdistribution-i.dat`	`evolution.gif`

In the first line of each data file, there is a short description of the content of the file.

- `final_state.dat` contains final state.
- `final_distribution.dat` contains final probability distribution.
- `statistics.dat` contains the mean, second moment, and the standard deviation for each evolution step.
- `wavefunction-i.dat` contains wave function of i-th step.
- `mean.eps` is the plot of the mean as function of the time steps.
- `final_distribution.eps` is a the plot of the final probability distribution.
- `standard_deviation.eps` is the plot of the standard deviation as function of the time steps.
- `evolution.gif` is the animation of the probability distribution from step 0 until final step.

Chapter 3

Examples

3.1 Discrete-Time Coined QW on a Line

The input file for the DTQW on a line must be similar to the following example:

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

The output data files will be stored in a folder called `DIR1`. The simulation will evolve 100 steps using the 2×2 Hadamard coin and initial state

$$|\psi\rangle = \frac{|0\rangle - i|1\rangle}{\sqrt{2}}|0\rangle,$$

that is, the particle starts at position $x = 0$ and the initial value of the coin is $(|0\rangle - i|1\rangle)/\sqrt{2}$.

Fig. 3.1 shows the final probability distribution ($t = 100$) and the standard deviation obtained by Hiperwalk. The figure also shows the gnuplot fitting.

3.2 Staggered QW on a Cycle

The discrete-time staggered QW on the line based on the simplest tessellation has four complex amplitudes a_1, a_2, b_1, b_2 which describe the block-states of each polygon (see `http://arxiv.org/abs/1408.5166` for

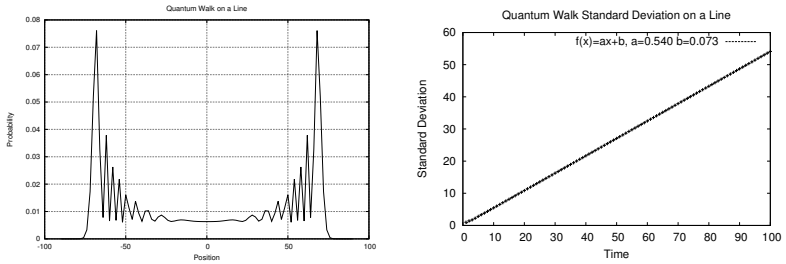

Figure 3.1: Final probability distribution and the standard deviation for the coined model.

details):

$$\left|u_x^0\right\rangle = a_1 \left|2x\right\rangle + a_2 \left|2x+1\right\rangle, \tag{3.1}$$

$$\left|u_x^1\right\rangle = b_1 \left|2x+1\right\rangle + b_2 \left|2x+2\right\rangle. \tag{3.2}$$

The evolution operator for the N-cycle is $U = U_1 U_0$, where

$$U_{0,1} = 2 \sum_{x=0}^{N-1} \left|u_x^{0,1}\right\rangle\left\langle u_x^{0,1}\right| - I. \tag{3.3}$$

The input file for the staggered QW on a cycle with 240 vertices must be similar to the following example:

```
1     WALK STAGGERED
2     DIRECTORY DIR2
3
4     STEPS 50
5     GRAPH CYCLE 240
6     POLYGONS 2
7     DISPLACEMENT 1
8
9     BEGINSTATE
10    0.707106781  0 120
11    0  0.707106781 121
12    ENDSTATE
13
14    BEGINTESSELLATION
15    0.7071067810 0  0.7071067810 0
16    0.8660254040 0  0.5          0
17    ENDTESSELLATION
18
19    PLOTS TRUE
20
21    HARDWAREID 0
```

The output data files will be stored in a folder called `DIR2`. The simulation will evolve 50 steps starting from initial state $|\psi\rangle = (|120\rangle + |121\rangle)/\sqrt{2}$. The tessellation is described by inputting parameters real and imaginary parts of a_1, a_2 in the first line and the real and imaginary parts of b_1, b_2 in the second line of TESSELLATION. The parameters are separated by a black space. The parameters used in this example are: $a_1 = a_2 = 1/\sqrt{2}$, $b_1 = \sqrt{3}/2$, $b_2 = 1/2$.

Fig. 3.2 shows the final probability distribution ($t = 50$) obtained by Hiperwalk.

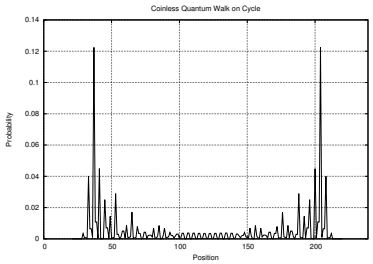

Figure 3.2: Probability distribution for staggered model after 50 steps.

3.3 Customized Evolution Operator

Given a initial state $|\psi\rangle$ and a unitary U, Hiperwalk can calculate the state at time t, that is, Hiperwalk outputs $U^t|\psi\rangle$ using the parallel devices of the user's computer. This kind of application is not restricted to the context of quantum walks, but can be used in any kind of application that requires this kind of calculation, such as a Markov process. The advantage of using Hiperwalk is that the user can compute using parallel resources speeding up the calculations.

An example of input file for a customized evolution operator is:

```
1   WALK CUSTOM
2   DIRECTORY DIR3
3
4   STEPS 100
5
6   INITIALSTATE  <filename1>.dat
7
8   UNITARY  <filename2>.dat  <filename3>.dat ...
9
10  HARDWAREID 0
11
```

The file `<filename1>.dat` must contain the initial state $|\psi\rangle = \sum_i \alpha_i|i\rangle$ in computational basis. The format of each line of `<filename1>.dat` is

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

which corresponds to the ith line (if there are no comments). The entries that are zero **must** be included. The lines **must** be in increasing order.

The unitary operators (one or more) must be stored in `<filename2>.dat`, `<filename3>.dat`, and so on. If

$$U_2 = \sum_{i,j} u_{i,j}|i\rangle\langle j|,$$

the format of each line of `<filename2>.dat` is

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

without including the entries that are zero. It does not matter the order of the lines.

After running Hiperwalk with this input file, the result will be the final state

$$(U_3U_2)^{100}|\psi\rangle,$$

if there are only two files (two unitaries: U_2 corresponds to `<filename2>.dat` and U_3 to `<filename3>.dat`). The final state will be stored in file `final_state.dat` using the same format of the input file `<filename1>.dat`.