Discrete-event simulation of quantum walks: Appendix

M. Willsch, D. Willsch, and K. Michielsen
Institute for Advanced Simulation, Jülich Supercomputing Centre,
Forschungszentrum Jülich, D-52425 Jülich, Germany and
RWTH Aachen University, D-52056 Aachen, Germany

H. De Raedt
Institute for Advanced Simulation, Jülich Supercomputing Centre,
Forschungszentrum Jülich, D-52425 Jülich, Germany and
Zernike Institute for Advanced Materials,
University of Groningen, Nijenborgh 4,
NL-9747 AG Groningen, The Netherlands
(Dated: April 27, 2020)

Appendix A: Quantum theoretical description of the QW

In this appendix, we outline the quantum theoretical description of the QW that can be used to obtain the theoretical results given in Table I. Using the same notation as in Eqs. (1) and (3), we denote the basis states as $|x,s\rangle$, where $x \in \{-L,\ldots,L\}$ and $s \in \{\uparrow,\downarrow\}$. In the context of Fig. 1, x increases from the top-right corner to the bottom-left corner as indicated by the indices of the detectors. The label $s=\uparrow$ ($s=\downarrow$) corresponds to the horizontal (vertical) arrows in Fig. 1.

With this notation, the optical elements at position x in Fig. 1 are described in terms of the operators

$$B^{(x)} = |x\rangle\langle x| \otimes M_{\rm BS} = \frac{1}{\sqrt{2}}(|x,\uparrow\rangle + i|x,\downarrow\rangle)\langle x,\uparrow| + \frac{1}{\sqrt{2}}(i|x,\uparrow\rangle + |x,\downarrow\rangle)\langle x,\downarrow|, \tag{A1}$$

$$P_1^{(x)} = |x\rangle\langle x| \otimes M_{\varphi_1} = e^{i\varphi_1}|x,\uparrow\rangle\langle x,\uparrow| + |x,\downarrow\rangle\langle x,\downarrow|, \tag{A2}$$

$$P_2^{(x)} = |x\rangle\langle x| \otimes M_{\varphi_2} = |x,\uparrow\rangle\langle x,\uparrow| + e^{i\varphi_2}|x,\downarrow\rangle\langle x,\downarrow|, \tag{A3}$$

where the beam-splitter matrix $M_{\rm BS}$ is given in Eq. (6) and the phase-shifter matrices M_{φ_1} and M_{φ_2} are given in Eq. (7). Combining the three operators, we obtain the description of a dashed box at position x in Fig. 1 as

$$T^{(x)} = P_2^{(x)} B^{(x)} P_1^{(x)} = \frac{1}{\sqrt{2}}|x\rangle\langle x| \otimes \begin{pmatrix} e^{i\varphi_1} & i \\ ie^{i(\varphi_1+\varphi_2)} & e^{i\varphi_2} \end{pmatrix}. \tag{A4}$$

The particle jump between the dashed boxes in Fig. 1 is described by the shift operator S given by Eq. (3) which shifts the $\uparrow$ state from x to $x-1$ and the $\downarrow$ state from x to $x+1$. For the initial state $|\Phi^{(0)}\rangle = |0,\uparrow\rangle$, we thus obtain the state after $l = 1,2,\ldots$ steps as

$$|\Phi^{(l)}\rangle = \begin{cases} SB^0|0,\uparrow\rangle & (l=1) \\ S(T^{(-l+1)} + T^{(-l+3)} + \cdots + T^{(l-1)})|\Phi^{(l-1)}\rangle & (l>1) \end{cases}. \tag{A5}$$

The probabilities in Table I at position x after l steps (corresponding to the detector D_x in Fig. 1) can then be computed as

$$p(x,l) = |\langle x,\uparrow|\Phi^{(l)}\rangle|^2 + |\langle x,\downarrow|\Phi^{(l)}\rangle|^2. \tag{A6}$$

Appendix B: PYTHON implementation of the subquantum model

In Listings 1–3, we give a simple PYTHON implementation of the DES used for the QWs studied in this paper (the three source files are also available online [1]). The code shows that DES provides a local realist model in which the trajectory of each individual particle can be followed. Nonetheless, the collective statistics of the individual particles perfectly agree with the experimental results and the quantum-theoretical predictions (see Figs. 2 and 4).

Listing 1. `qw_general.py`: General QW implementing the network in Fig. 1 and used to obtain the results in Fig. 2

```python
#!/usr/bin/env python3
import numpy as np
from des_elements import particle,beamsplitter

L = 4            # number of steps/levels in the quantum walk
N = 100000       # number of individual particles simulated
phi1 = np.pi/2   # phase shifter angle 1
phi2 = -np.pi/2  # phase shifter angle 2

# create local beamsplitters at each level l and position x
beamsplitters = {}
for l in range(1,L+1):
    beamsplitters[l] = {}
    for x in range(-l+1,l,2):
        beamsplitters[l][x] = beamsplitter(polarizing=False)

# create local detectors (particle counters) at level L
detectors = {}
for x in range(-L,L+1,2):
    detectors[x] = 0

# simulate each individual particle's trajectory through the network in Fig. 1
for n in range(N):
    p = particle()  # create new particle
    x = 0           # current position of the particle on its trajectory
    port = 0        # current port at the beam splitters (0: left/right, 1: top/down)
    x,port = beamsplitters[1][x].passthru(p,x,port)  # first beam splitter at level 1
    for l in range(2,L+1):                           # each dashed box in Fig. 1
        if port == 0: p.apply_phaseshift(phi1)           # phase shifter 1
        x,port = beamsplitters[l][x].passthru(p,x,port)  # beam splitter
        if port == 1: p.apply_phaseshift(phi2)           # phase shifter 2
    detectors[x] += 1  # finally detect the particle at position x

# print normalized detector counts for Fig. 2
print('# x Nx/N (for l='+str(L)+')')
for x,Nx in detectors.items():
    print(x, Nx/N)
```

Listing 2. `qw_experiment.py`: QW implementing the network in Fig. 3 and used to obtain the results shown in Fig. 4

```python
#!/usr/bin/env python3
import numpy as np
from des_elements import particle,beamsplitter

N = 100000       # number of individual particles simulated
remove = 'none'  # which particles to remove at t2 ('none', 'x=1', 'x=-1')

# local beamsplitters and detectors (x grows from top-right to bottom-left in Fig. 3)
beamsplitters = {
    1: {x: beamsplitter() for x in [-1, 0, 1]},                          # 1. gray region
    2: {x: beamsplitter() for x in [-2, -1, 0, 1, 2]},                   # 2. gray region
    3: {x: beamsplitter() for x in [-3, -2, -1, 0, 1, 2, 3]},            # 3. gray region
    4: {x: beamsplitter() for x in [-4, -3, -2, -1, 0, 1, 2, 3, 4]},     # 4. gray region
}
detectors = {
    'all':  {x: 0 for x in [-2, -1, 0, 1, 2]},  # count all particles
    'x=1':  {x: 0 for x in [-2, -1, 0, 1, 2]},  # count those that went along x=1 at t2
    'x=-1': {x: 0 for x in [-2, -1, 0, 1, 2]},  # count those that went along x=-1 at t2
}

# simulate each individual particle's trajectory through the network in Fig. 3
for n in range(N):
    p = particle()  # create new particle
    x = 0           # current position of the particle on its trajectory
    port = 0        # current port at the beam splitters (0: left/right, 1: top/down)
    for r in [1,2,3,4]:  # pass through each gray region r in Fig. 3
        p.apply_hadamard()                               # the H box in Fig. 3
        x,port = beamsplitters[r][x].passthru(p,x,0)     # always enter at port 0 (left)
        _,port = beamsplitters[r][x].passthru(p,x,port)  # second beam splitter in r
        if port != 0:  # during learning, a few particles may leave at the wrong port
            break
        if r == 1:
            p.x_at_t2 = x  # at t2, tag particle for "ideal non-invasive measurement"
            if remove == 'x=1'  and x == 1:  break
            if remove == 'x=-1' and x == -1: break
        if r == 4:
            detectors['all'][x/2] += 1  # at t3, the particle triggers a detector
            if p.x_at_t2 ==  1: detectors['x=1'][x/2]  += 1
            if p.x_at_t2 == -1: detectors['x=-1'][x/2] += 1

# print normalized detector counts for Fig. 4 (second column depends on 'remove')
print('# x Fig.4(a)-(c) Fig.4(e) Fig.4(f)')
for x in [-2,-1,0,1,2]:
    print(x, detectors['all'][x]/N, detectors['x=-1'][x]/N, detectors['x=1'][x]/N)
```

Listing 3. `des_elements.py`: Definition of the basic DES elements

```
#!/usr/bin/env python3
import numpy as np
from numpy import sin,cos,exp,sqrt,pi
from numpy.random import rand

class particle:
    def __init__(self, xi=0): # xi: polarization (h: xi=0, v: xi=pi/2)
        self.m = np.array([   # every particle carries a message m
            sin(xi),  # vertical component (v)
            cos(xi)   # horizontal component (h)
        ])
    def apply_phaseshift(self, phi):
        self.m = exp(1j*phi) * self.m
    def apply_hadamard(self):
        self.m = 1/sqrt(2.) * np.array([
            [1, 1],
            [1, -1]
        ]) @ self.m

class beamsplitter:
    def __init__(self, polarizing=True, learningrate=0.98):
        if polarizing:  # polarizing beam splitter (transmits h and reflects v)
            self.TBS = np.array([
                [0, 1j, 0,  0],
                [1j, 0, 0,  0],
                [0,  0, 1, 0],
                [0,  0, 0, 1]
            ])
        else:  # normal 50:50 beam splitter (splits both h and v)
            self.TBS = 1/sqrt(2.) * np.array([
                [1,   1j,  0,  0],
                [1j,  1,   0,  0],
                [0,   0,   1, 1j],
                [0,   0,  1j,  1]
            ])
        self.lr = learningrate
        # the state of each local beam splitter is determined by 6 numbers
        self.X = [.5, .5]
        self.Y = [[1, 0], [0, 1]]
    def passthru(self, p, x, i):  # p: particle, x: position, i: input port (0/1)
        self.Y[i] = p.m
        self.X[i]   = self.lr * self.X[i] + (1 - self.lr)
        self.X[1-i] = self.lr * self.X[1-i]
        W = self.TBS @ [
            sqrt(self.X[0])*self.Y[0][0],
            sqrt(self.X[1])*self.Y[1][0],
            sqrt(self.X[0])*self.Y[0][1],
            sqrt(self.X[1])*self.Y[1][1],
        ]
        # determine output port and update particle message
        prob0 = abs(W[0])**2 + abs(W[2])**2
        if rand() < prob0:
            p.m = np.array([W[0], W[2]]) / sqrt(prob0)
            return x-1, 0  # return new position and output port
        else:
            p.m = np.array([W[1], W[3]]) / sqrt(1-prob0)
            return x+1, 1  # return new position and output port
```

[1] `https://jugit.fz-juelich.de/qip/quantum-walk`, last accessed: April 25, 2020.