Quantum Resetting of Walks

An exploration of the effect of stochastic and quantum resetting on classical and quantum walks

Dhruva Sambrani, supervised by Manabendra Nath Bera

Dept of Physics, IISER - Mohali

Table of Contents

§0 Introduction

Random walks are a commonly used tool in the arsenal of algorithms on Classical Computers to solve a variety of problems which do not have a known easy solution. The power of such methods can generally be attributed to the fact that while the space of possible solutions is vast, we generally only need to sample few solutions to come to a close solution. This power has been routinely exploited in the past to sample from Markov Chains using the MCMC algorithm which can be found in any introductory textbook [1] of markov processes, and they have recently been used in the fields of Financial engineering [2] , fluid mechanics [3] , fitting blackhole images [4] and most famously in the Los Alamos project [5] .

With the advent of quantum algorithms, quantum walks have arisen as an obvious extension of classical walks in the quantum domain. The applications of quantum walks are just as many: ANN training [6] , Random Number Generation [7] , List coloring (Grover) [8] , collision finding [9] , link prediction [10] . There has even been a recent foray into classifying and using a quantum-classical walk to speed up certain classical algorithms, namely the Google PageRank Algorithm [11] . The increased interest in quantum walks can be attributed to the quadratic speed up which it grants the solution a-la the Grover algorithm, which itself can be considered as a Quantum Walk [12] . While there is also interest in studying the physical implementation of the walks, we do not concern ourselves with these questions, since experimentalists are much better equipped to ask them.

In this report, we will define both the classical (Section 1) and quantum (Section 2) walks along with certain properties that are, while interesting in their own regard, problematic for certain scenarios (Section 3). Then, we look at previous attempts at finding a solution for this (Section 4), and discuss the shortcomings of the solution. Finally, we introduce a new mechanism (Section 5) which may even eliminate the shortcomings, whose study will be continued in the second part of the Thesis Project.

Along with the theory, certain aspects of the implementation of the walks computationally are also added as necessary. This is done inline instead of in an appendix, which is the norm, since the author believes that such a presentation solidifies the readers' understanding of both, the model and the implementation.

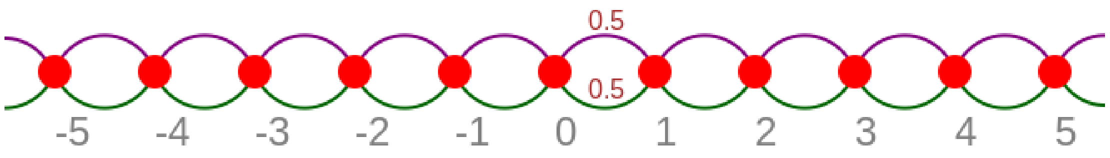
§1 Classical Walks

Classical random walks are defined on a graph, with the walker being on some initial node i_0 with a probability λ_i, and "hopping" from node i to j in each time step with a probability given by p_{ij}. We can thus define a transition matrix $P := P_{ij} = p_{ij}$ which is row stochastic. Thus the probability mass function of the walker at timestep t is given by λP^t. Of course, the exact structure of the graph and the probabilities will decide the properties of the walk and there exists a vast amount of literature devoted to this analysis.

We however will restrict our discussion to the symmetric walk on the 1D chain. This is often called the 1D-Simple Symetric Random Walk, and defined in the following way.

Definition

Consider an infinite 1 D chain, with nodes marked by $\mathbb{Z}$.



Define the probability of hopping from node i to node j

$$p_{ij} = \begin{cases} 1/2 & |i-j| = 1 \\ 0 & \text{otherwise} \end{cases}$$

And the initial state as

$$\lambda_{ij} = \begin{cases} 1 & i = 0 \\ 0 & \text{otherwise} \end{cases}$$

Thus, the hopper can only move to it nearest neighbors, starting from node 0.

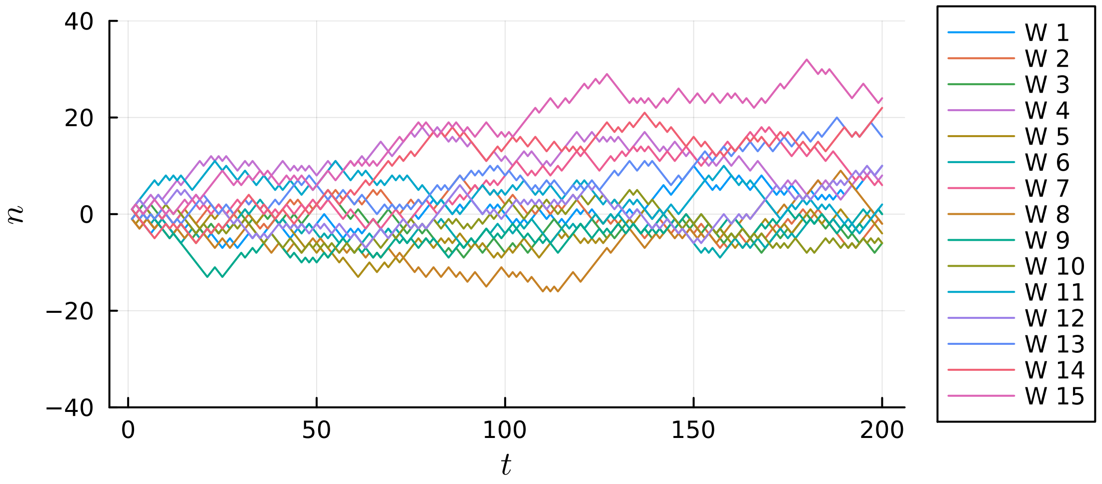
Note that $P(X_t = i| \text{ HISTORY }) = P(X_t = i|X_{t-1})$, which means that the walk is a markov chain.

Multiple Walkers

The SSRW is clearly a stochastic process, and each run of this process will lead to different paths being chosen by the walker, and we are my interested in what the walker does on an average rather than what happens in a particular instance. Thus, we can observe multiple walks, and plot their paths to visualize how they would spread.

In order to simulate a walk, we sample $X_i \in \{-1, 1\}$ with equal probability, and add it to the previous position S_{i-1} to get S_i. Note $S_0 = 0$. Thus, this is equivalent to sampling from `[1, -1]` uniformly `t` times and performing a cumulative sum. For `n` walkers, we can sample `(t, n)` such random numbers, and do a cumulative sum along the first dimension.

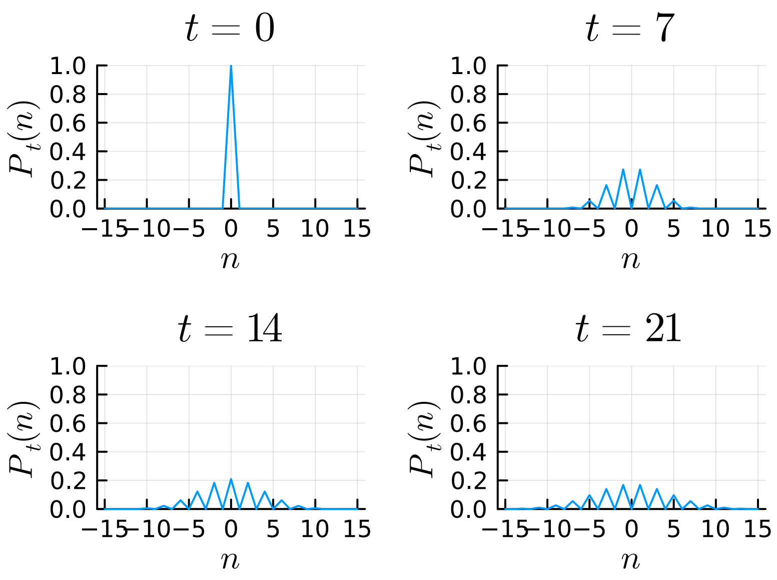
```
bm = cumsum(rand([1, -1], (200, 15)), dims=1);
```



Probability mass function

Another common way to visualize the walk is to plot the probability that the walker is on node i at time t. For this, we simply define the transition matrix and λ appropriately and find λP^t. Note however that the transition matrix for the SSRW is Tridiagonal with the Upper and Lower diagonals as 0.5 and the diagonal as 0, and there exist efficient storage and multiplication routines. Also, since we can only store a finite matrix, we limit the walk to some size. At the boundaries, we simply allow open conditions, because this is the easiest to implement.

```
cps, cps_t = let
    t = 21
    U = SymTridiagonal(fill(0., 31), fill(0.5, 30))
    λ = fill(0., 31)
    λ[16] = 1
    ps = accumulate(1:t, init=λ) do old, _
        U * old
    end[t÷3:t÷3:t], t÷3
end;
```



Properties of the walk

There are certain properties of the walk that are interesting for the problem we pose in the subsequent sections. Primarily, we are interested in the mean, standard deviation and recurrence of the graph.

The probability that a walker is on node n at time t is given by the expression

$$p(n,t) = \binom{t}{\frac{t+n}{2}} \frac{1}{2^t}$$

This equation is valid only if $t + n$ is even and $n \leq t$. If $t + n$ is odd or $n > t$, the probability is zero

It should be obvious from the even symmetry of $p(n,t)$ that the mean $\mu(t) = \sum_n np(n,t) = 0$. This comes from the fact that the walk is symmetric.

The standard deviation of the walker position $\sigma(t) = \sqrt{\langle n^2 \rangle - \langle n \rangle^2} = \sqrt{\sum_n n^2 p(n,t)} = \sqrt{t}$ [13]

§2 Quantum Walks

Introduction

Quantum walks are defined analogous to the classical walks.

First, our walker is quantum mechanical, and the position of the walker can be a superposition of the nodes. Thus, we define the state of the walker to be a superposition of the node states $|i\rangle$.

$$|\psi\rangle = \sum_i c_i |i\rangle$$

Let this Hilbert space be denoted as $\mathcal{H}_W$. The projection of the state on some node i given by $|\langle i|\psi\rangle|^2 = |c_i|^2$ is understood as the probability that the walker will collapse to the state $|i\rangle$ on measurement.

To define the evolution of the node, we look back at the transition matrix P_{ij}. Thus we would define an operation in the following manner -

$$O|i\rangle = \sum_j \sqrt{p_{ij}}|j\rangle$$

and the walk proceeds by repeated application of O. While this expression is enough to define the walk, it is not immediately clear how one would explicitly realize such an operation. There are multiple formalisms to define such walks, but we shall use the coined quantum walk formalism which is very natural for regular graphs, as is the case for the 1D chain. Note that for the symmetric walks on nD lattices, like the 1D chain, the tight-binding model is already a well understood continous time quantum walk. However, we prefer a discrete time formalism.

Coined Quantum Walk for 1D Chain

For the 1D chain, given a specific node, there are only 2 other nodes connected to it. Thus, we can add a 2 level system, which "decides" which node the walker jumps to. More formally, we attach a 2 level qubit system to the walker whose bases are denoted by $|0\rangle$ and $|1\rangle$. Let us denote this hilbert space as $\mathcal{H}_C$

Thus, we can define the shift operation as

$$S|0\rangle|i\rangle = |0\rangle|i-1\rangle$$

$$S|1\rangle|i\rangle = |1\rangle|i+1\rangle$$

How would we define such an operation explicitly?

$$S = |0\rangle\langle 0| \otimes S_L + |1\rangle\langle 1| \otimes S_R$$

Where S_L and S_R are defined as

$$S_L|i\rangle = |i-1\rangle,\ S_R|i\rangle = |i+1\rangle$$

Note that S operates on $\mathcal{H}_C \otimes \mathcal{H}_W$, whereas S_L and S_R operate on $\mathcal{H}_W$ only.

The superposition in the two choices at each step is recovered by putting the coin into a superposition of its basis states. This is achieved via a coin operator, which is commonly defined as $H \otimes I$, where H is the single qubit hadamard operator.

Let us explicitly write down two steps of the walk

$$H \otimes I(|0\rangle|0\rangle) = \frac{|0\rangle|0\rangle + |1\rangle|0\rangle}{\sqrt{2}}$$

$$S\left(\frac{|0\rangle|0\rangle + |1\rangle|0\rangle}{\sqrt{2}}\right) = \frac{|0\rangle|-1\rangle + |1\rangle|1\rangle}{\sqrt{2}}$$

$$H \otimes I \frac{|0\rangle|-1\rangle + |1\rangle|1\rangle}{\sqrt{2}} = \frac{|0\rangle|-1\rangle + |1\rangle|-1\rangle + |0\rangle|1\rangle - |1\rangle|1\rangle}{2}$$

$$S \frac{|0\rangle|-1\rangle + |1\rangle|-1\rangle + |0\rangle|1\rangle - |1\rangle|1\rangle}{2} = \frac{|0\rangle|-2\rangle + (|1\rangle + |0\rangle)|0\rangle - |1\rangle|2\rangle}{2}$$

Repeated application of Coin Operator

Note specifically the repeated application of the coin operator. If the coin operator is not applied in step 3, our state will end up in $\frac{1}{\sqrt{2}}(|0\rangle|-2\rangle + |1\rangle|2\rangle)$ which is not what we wanted. This is because the application of the controlled shift operation entangles the coin and the walker systems, and thus there is no superposition in each term of the system

Computational Implementation

There are 2 ways to implement the Quantum walks, which are both interesting in their own ways. But the foremost thing to tackle would be how to store an infinite vector and an infinite dimensional operator. Since we cannot do either of these simply, we instead limit our walk to a node space of $2n$, and use periodic boundary conditions.

Other Boundary Conditions

One could pick other boundary conditions too, such as the open or the absorbing boundary conditions, but both of these BCs lead to nonunitary operations, which complicate the definition and application of the gate.

Matrix formalism

The first and more immediate method is to simply write down the matrix equivalent of the above operations. For the visualization of the implementations, we will assume that the walk occurs in a 1 D chain of size 4

The coin is prefered to be in the $\frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$ when we start so that the walk proceeds symmetrically [13]

```
init_coin = 1/√2 * (ket(1,2) - 1im * ket(2,2));
```

The coin operator is trivial to write,

```
H =
▶KrausOperators([8×8 SparseMatrixCSC{Float64, Int64} with 16 stored entries:
                 0.707107    ⋅          ⋅          ⋅         0.707107    ⋅          ⋅
                  ⋅         0.707107    ⋅          ⋅          ⋅         0.707107    ⋅
                  ⋅          ⋅         0.707107    ⋅          ⋅          ⋅         0.70710
                  ⋅          ⋅          ⋅         0.707107    ⋅          ⋅          ⋅
                 0.707107    ⋅          ⋅          ⋅        -0.707107    ⋅          ⋅
                  ⋅         0.707107    ⋅          ⋅          ⋅        -0.707107    ⋅
                  ⋅          ⋅         0.707107    ⋅          ⋅          ⋅        -0.70710
                  ⋅          ⋅          ⋅         0.707107    ⋅          ⋅          ⋅
```

```
H = KrausOperators([sparse(hadamard(2)⊗I(4))])
```

The left and right shift operators can be defined in the following manner

```
▶(4×4 Matrix{Float64}:, 4×4 Matrix{Float64}:)
  0.0  0.0  0.0  1.0    0.0  1.0  0.0  0.0
  1.0  0.0  0.0  0.0    0.0  0.0  1.0  0.0
  0.0  1.0  0.0  0.0    0.0  0.0  0.0  1.0
  0.0  0.0  1.0  0.0    1.0  0.0  0.0  0.0
```

```
begin
    R = collect(Tridiagonal(fill(1., 3), zeros(4), zeros(3)))
    R[1, end] = 1
    L = collect(Tridiagonal(zeros(3), zeros(4), fill(1., 3)))
    L[end, 1] = 1
    R, L
end
```

Thus the shift operator is defined as

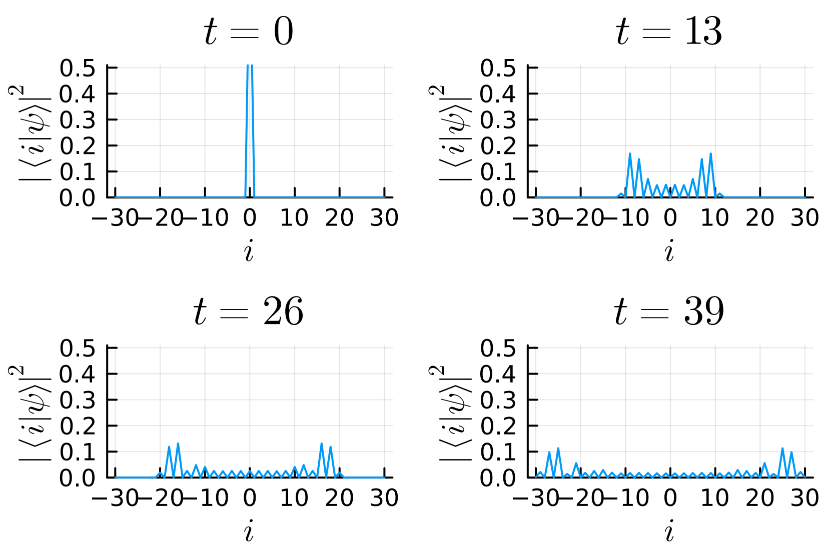
```
S =
▶KrausOperators([8×8 SparseMatrixCSC{ComplexF64, Int64} with 8 stored entries:
                    ⋅        1.0+0.0im      ⋅           ⋅       …      ⋅           ⋅
                    ⋅           ⋅        1.0+0.0im      ⋅              ⋅           ⋅
                    ⋅           ⋅           ⋅        1.0+0.0im         ⋅           ⋅
           1.0+0.0im            ⋅           ⋅           ⋅              ⋅           ⋅
                    ⋅           ⋅           ⋅           ⋅              ⋅           ⋅       1
                    ⋅           ⋅           ⋅           ⋅       …      ⋅           ⋅
                    ⋅           ⋅           ⋅           ⋅          1.0+0.0im       ⋅
                    ⋅           ⋅           ⋅           ⋅              ⋅       1.0+0.0im
```

```
S = KrausOperators([sparse(proj(ket(1, 2)) ⊗ L + proj(ket(2, 2)) ⊗ R)])
```

Thus, we can repeatedly apply $S \circ H$ to the initial state and accumulate the results.

```
begin
    init_state = proj(init_coin ⊗ ket(2,4))
    ψ = [[init_state]; accumulate(1:40, init=init_state) do old, _
        H(S(old))
    end]
end;
```

Now we can plot the readout statistics by partially tracing out the coin, and plotting the diagonal. The following is a walk on 61 nodes.



Quantum Walks are not random

Note, this is a single walker, not an ensemble of walkers as is the case in classical random walks. Quantum walks (without measurement), are completely deterministic.

Circuit Formalism

While the matrix definition of the Quantum Walks are easy to formalize and understand, finally these need to be simulated on some sort of device which may require us to reformulate it. Further, we try to ensure that the reformulation allows for easy additions of other dynamics which we may want to study.

The advantages of using a Quantum Computer over a classical computer for a quantum walk should be obvious. The problem however is that the quantum walk is over a high dimensional space, and we rarely have access to such high dimensional systems which we can control easily. Instead we need to simulate such a system using the accessible 2 level systems available in quantum computers.

Let us define the basic components of our walk.

Nodes

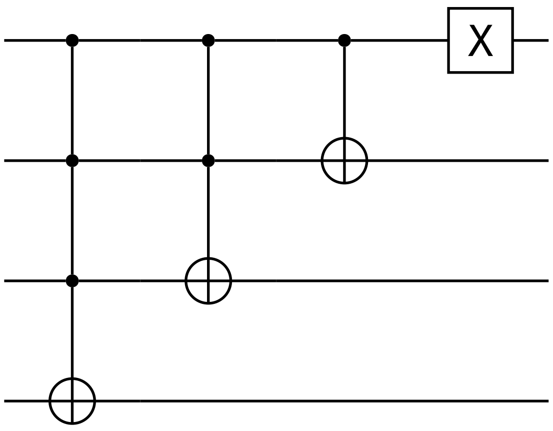
- Each numbered node is then converted into its 2-ary n length representation denoted by (x_i). n is chosen such that 2^n > N
- These bitstrings are encoded into an n qubit computer, where each basis state in the computational basis corresponds to the node with the same bitstring.
- The amplitude of a particular basis corresponds to the amplitude of the walker in the corresponding node

Coin

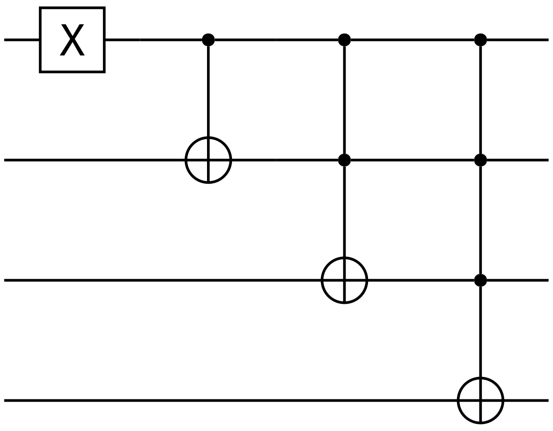
- The Walk Coin is a two level system, as usual

Edges and Shifts

- Since shifts are only to adjacent nodes, the left (right) shift is equivalent of subtracting (adding) 1 from the bitstring of the state.
- From the Quantum adder circuit, we can set one input to be $(0)_{n-1}1$ and reduce the circuit to get the Quantum AddOne circuit. Shown below is the circuit for $n = 4$ $(N = 16)$



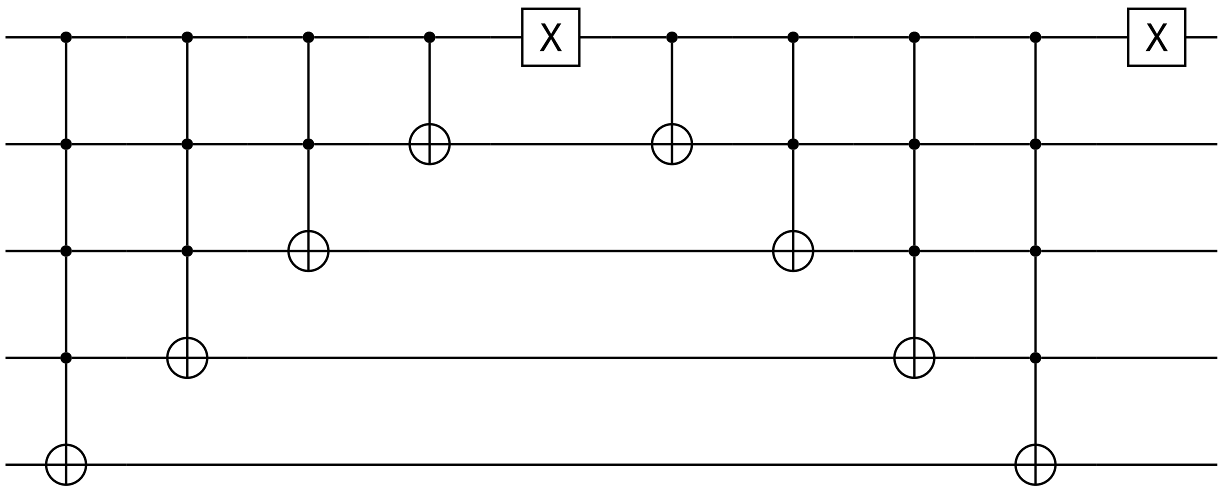
- We can similarly construct the SubOne circuit, but that is simplified by noting that the subone circuit is simply the inverse of the addone circuit, and this corresponds to just inverting the circuit (all gates are unitary).



```
leftshift (generic function with 1 method)
```

```
begin
    rightshift(n) = chain(
        n,
        map(n:-1:2) do i
            control(1:i-1, i=>X) end...,
        put(1=>X)
    )
    leftshift(n) = rightshift(n)'
end
```

The controlled shift operation is encoded as

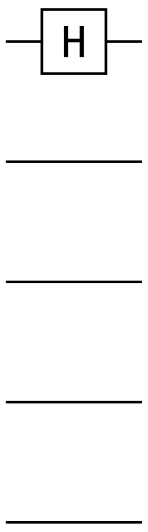


```
shift (generic function with 1 method)
```

```
shift(n) = chain(n+1,
    control(1, 2:n+1 => rightshift(n)),
    put(1 => X),
    control(1, 2:n+1 => leftshift(n)),
    put(1 => X),
)
```

Coin operator

We can add the coin operator as usual



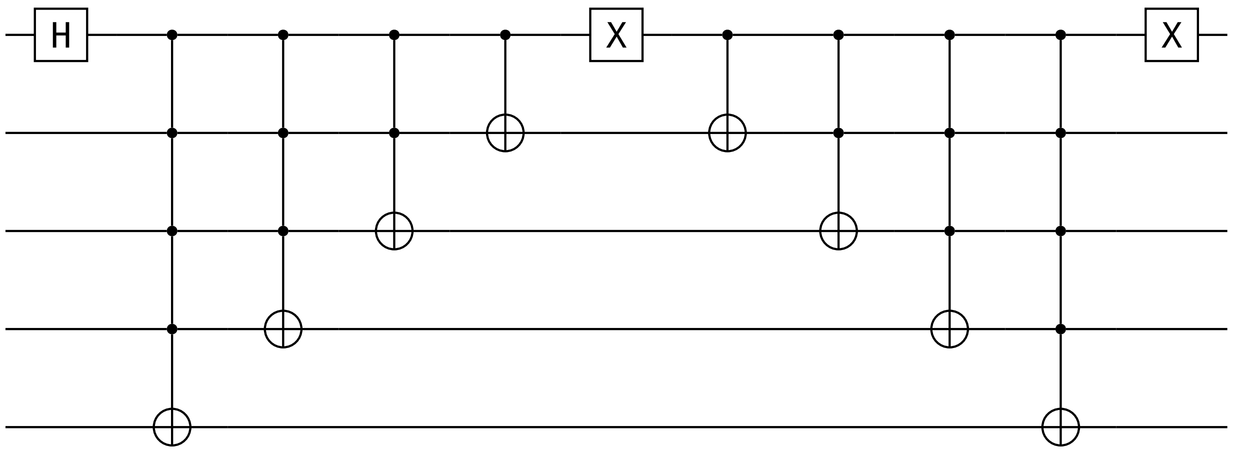
```
coin (generic function with 2 methods)
```

```
coin(n, c=Yao.H) = chain(n+1, put(1 => c))
```

Evolve Circuit

Putting these together, we get the operation for a single step of the evolution as below. Note that the top qubit rail is that of the coin, and the rest are those of the simulation of the system.

This circuit can be repeated to acheive any number of steps.



```
evolve (generic function with 1 method)
```

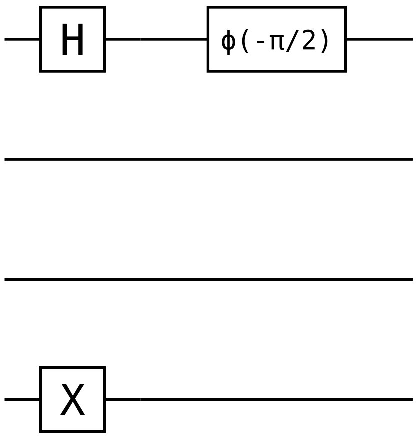
```
evolve(n) = shift(n) * coin(n)
```

Prepare circuit

While we can already simulate the walk, an very useful helper function that we can define is the prepare circuit.

Quantum registers are often initialized to the 0 state, and it is also easy to restart the walk from the 0 state. However, we often like to start the walk in the center of the chain instead of at node 0. Also, the coin is prefered to be in the $\frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$ when we start so that the walk proceeds symmetrically **[renato: pg 30]** .

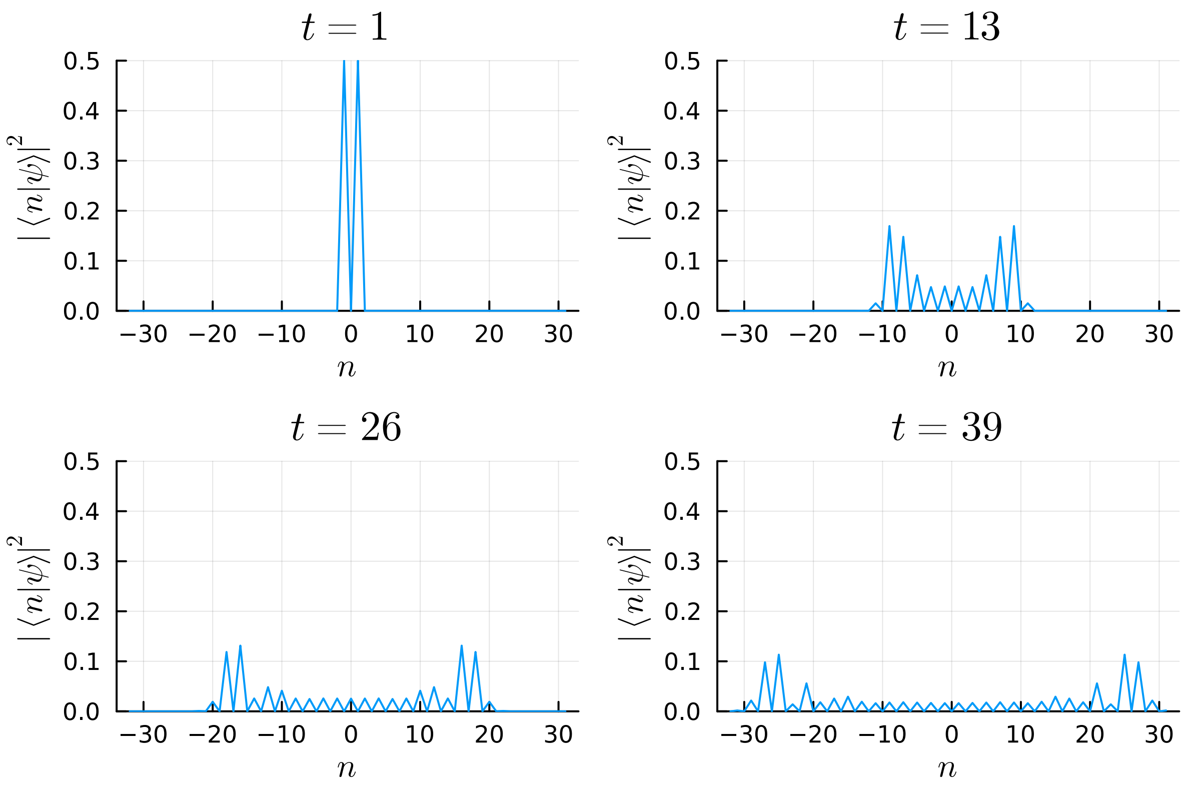
Hence, to perform these steps, we define the following `prepare` subroutine.



```
prepare (generic function with 1 method)
```

```
prepare(n) = chain(
    n+1,
    put(1=>Yao.H),
    put(1=>Yao.shift(-π/2)),
    put(n+1=>X)
)
```

Similarly plotting as before,



Thus we can see that the two formalisms are the same.

Properties of the walk

Similar to the classical random walk, the mean $\mu(t) = 0$.

However, a more interesting feature is that the standard deviation $\sigma(t) = 0.54t$ [13] . Compare this with the classical random walk, the quantum walk has a quadratic speed up. This the reason for the quadratic speedup commonly seen in the Grover search and other monte carlo problems.

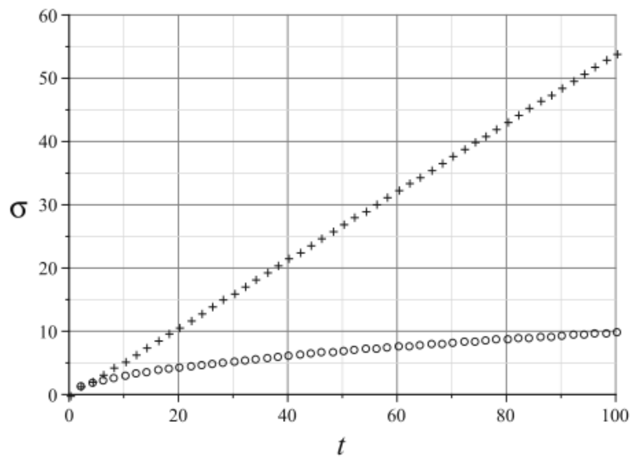


Fig. 3.6 Standard deviation of the quantum walk (*crosses*) and the classical random walk (*circles*) as a function of the number of steps

§3 Markov Processes and Hit Rates

Search Problems

A common application of walks is in search problems. In search problems, we generally have a black box function

$$f(x) = \begin{cases} 0 & x \in G \\ 1 & x \in G^C \end{cases}$$

where $G \cup G^C = S$ which is the search space. We are interested in developing an algorithm to output some $x \in G$ by querying f. This problem however is much more complex, as we require some sort of "learning" by the walker to reach the output nodes. Instead, we look at a simpler problem of hitting a selected target node. Naturally, we not only want high success rates, but we also want low mean hit times.

Formalism

Define the following

- Denote the readout at the nth measurement (at $t = n\tau$) as X_n.
- Select a target node δ
- Probability of first hit in n steps $F_n = P(X_n = \delta | X_i \neq \delta \forall i \in [0, n-1])$
- Mean hit time $\langle t_S \rangle = \sum_{i=0}^{\infty} iF_i$
- Success probability in n steps $S_n = \sum_{i=1}^{n} F_i = P(\exists i \in [0, n] | X_i = \delta)$
- Survival probability $=$ Failure probability $= \mathcal{S}_n = 1 - S_n$
- Asymptomatic versions of these terms are given by taking $n \to \infty$

Readout in Walks

To identify whether a walker has hit the target node, we need to track the location of the walker as it evolves in time.

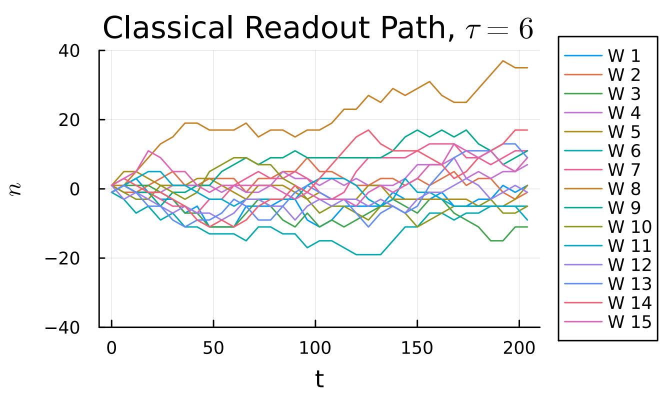
For the classical case, this poses no problem, as measurement does not disturb the system. In the quantum case however, we need to be a bit more careful.

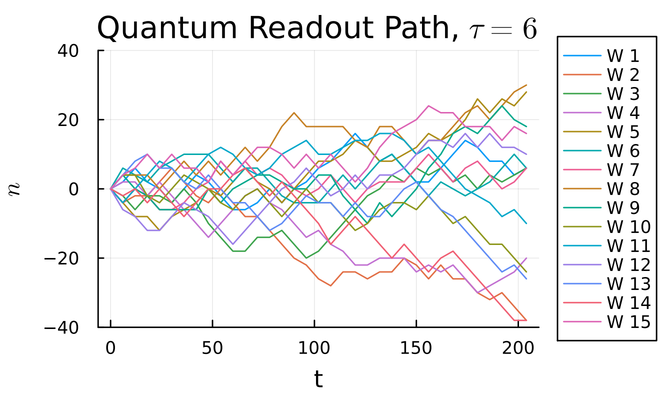
A constantly measured walker will freeze the dynamics of a quantum walker. This is known as the Quantum Zeno effect. The solution for this is to measure after every τ steps.

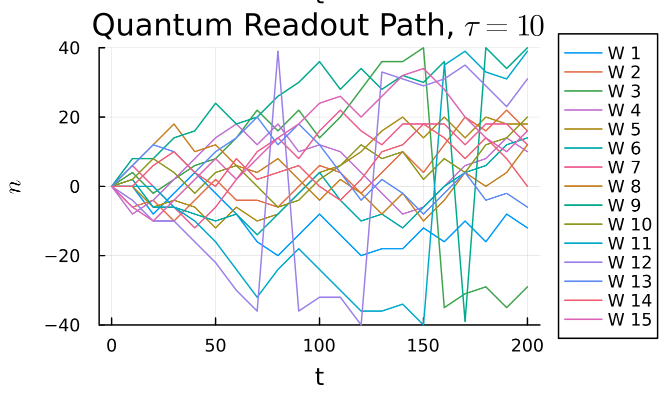
τ=1

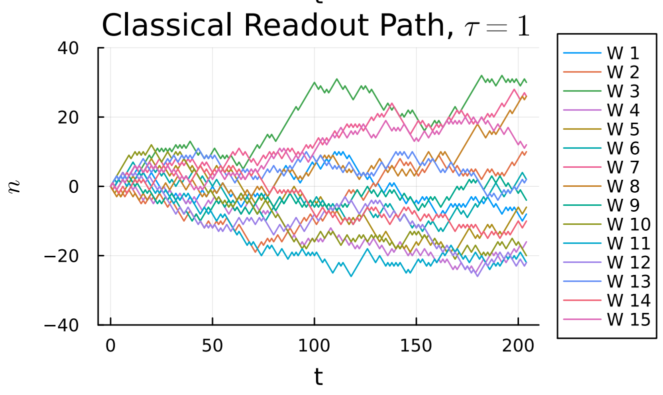
The quantum $\tau = 1$ case reduces the discrete time quantum walk to a classical random walk with $\tau = 1$

Let us plot the readout trajectories of walkers with different paramters









Note the very clear difference in the spread between the classical readout and the quantum readout for same τ. Similarly note the spread between the quantum readouts for different τs.

Markov Chains and Walks

In the classical case, it is clear that the 1D SSRW is a markov process. In the appendix, we show that the quantum walk with measurement is also a markov process.

Consequently, we can use certain results from the theory of Markov processes to compare the two walks.

Irreducibilty and Recurrence

Under the usual definition of irreducibility [1] ,

Definition: Irreducibility

If $\forall i, j \in S, \exists n, m | P_{ij}^n > 0, P_{ji}^m > 0$, then the chain is called irreducible.

It is trivial that in the 1D chain, $n = m = |i - j|\tau$ satisfies the condition.

Following [1] ,

Definition: Recurrence

If $i \in S, \sum_{n=1}^{\infty} P_{ii}^n \to \inf$, then the state is called recurrent. Equivalently, $\sum_n F_n \to 1$ for recurrent nodes. If a chain is irreducible and one of its states is recurrent, all its states are recurrent and thus the chain is called recurrent.

It is a well known fact that the 1D SSRW is recurrent. It is just as well known a fact that the 3D SSRW is transient [1] . In the quantum case, it is not as clear whether any of the available definitions of the quantum markov process is more or less better than the others. Thus, the transience of quantum walks depends on the exact definition we are working under. In our definition however, it can be shown using symbolic computation [14] that the walk is indeed transient. This poses an interesting problem.

Survival probability

We can thus measure and plot the $S_n - n$ curve for the quantum and classical walks to compare the efficiency of these walks in the first hit problem.

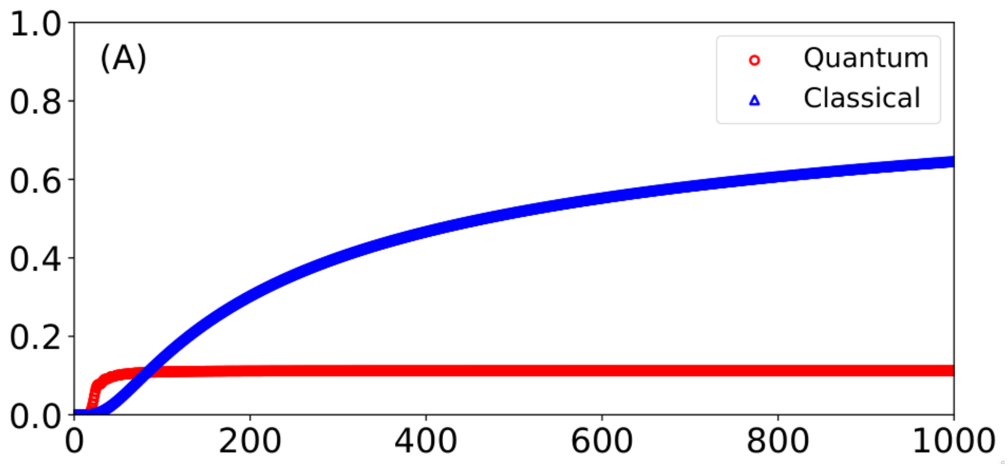
Continuous Walks

Note that the following results are for continuous time walks solved numerically using the analytical solution. Such a result for discrete time walks is harder due to the nature of the walk. However we intend to reproduce the analytical results for the discrete time case too.

Computationally too, due to the finite size of the walk space, the walk automatically becomes recurrent. However one can still observe qualitatively similar plots. Such a plot can be found in the next section.

Below are the plots from [15] for the quantum and classical walks.

Success Probability vs Time



Observations

- The quantum walk has a fast rise in the initial phase but saturates at ~0.1
- The classical walk has a slow rise, but eventually reaches 1

So while the quantum walker is faster, the walk is now transient, which leads to a non zero asymptomatic failure rate.

Clearly this is a problem. What this means physically is that for the first hit problem, if the quantum walk hits the target node, it does so faster than the classical walk, but a majority of the times, it doesn't hit the target node at all.

§4 The Solution: Resetting

The crux of the matter is this. The quantum walk is fast in the initial phase, but eventually saturates asymptotically to a success rate of < 1. This means that some of the walkers hit the target, and others do not. If that is the case, is it possible to restart the walk when it starts saturating? That is, can we restart the walk for the walkers that have not yet hit the target after $r\tau$ time?

For such dynamics, we will need to reformulate the walk slightly.

Formalism

A reset of the walker implies that the walker returns to its initial state, and the walk dynamics continue from there.

However, we can vary when we reset the walker by considering multiple reset processes. A common reset process is the Poissonian reset, where the reset times are modelled as a poissonian process with some parameter r. This is particularly convenient in the case of continuous time walks.

For the discrete walks, the geometric distribution is more natural.

$$t \sim \mathrm{Geom}(\gamma)$$

that is, at each step, there is a γ probability of reset.

This results in different dynamics, which can be seen in subsequent subsections, but it has an equally drastic effect on the recurrence of the walk.

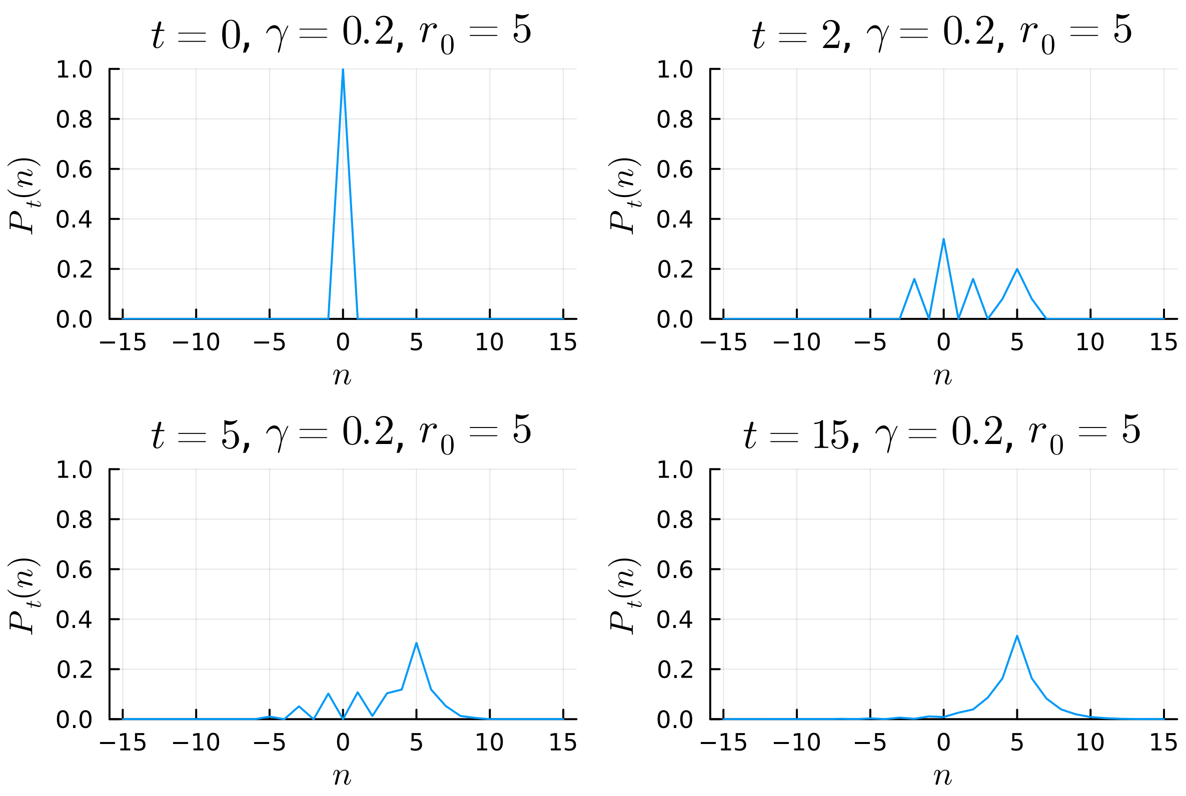
Recurrence and Resetting

In the geometric resetting case, it is clear that $P_{00}^n \geq \gamma$, and hence $\sum_n P_{00}^n \geq \sum_n \gamma$ which diverges as $n \to \infty$. Thus regardless of the initial walk, the final walk will definitely be recurrent. Thus the motivation for resetting the quantum walk which is transient should be immediately clear.

Stochastic Reset Classical Walk

In the classical case, the transition probabilities change to

$$p_{ij} = \begin{cases} (1-\gamma)\cdot 1/2 & |i-j| = 1 \\ \gamma & j = r_0 \\ 0 & \text{otherwise} \end{cases}$$

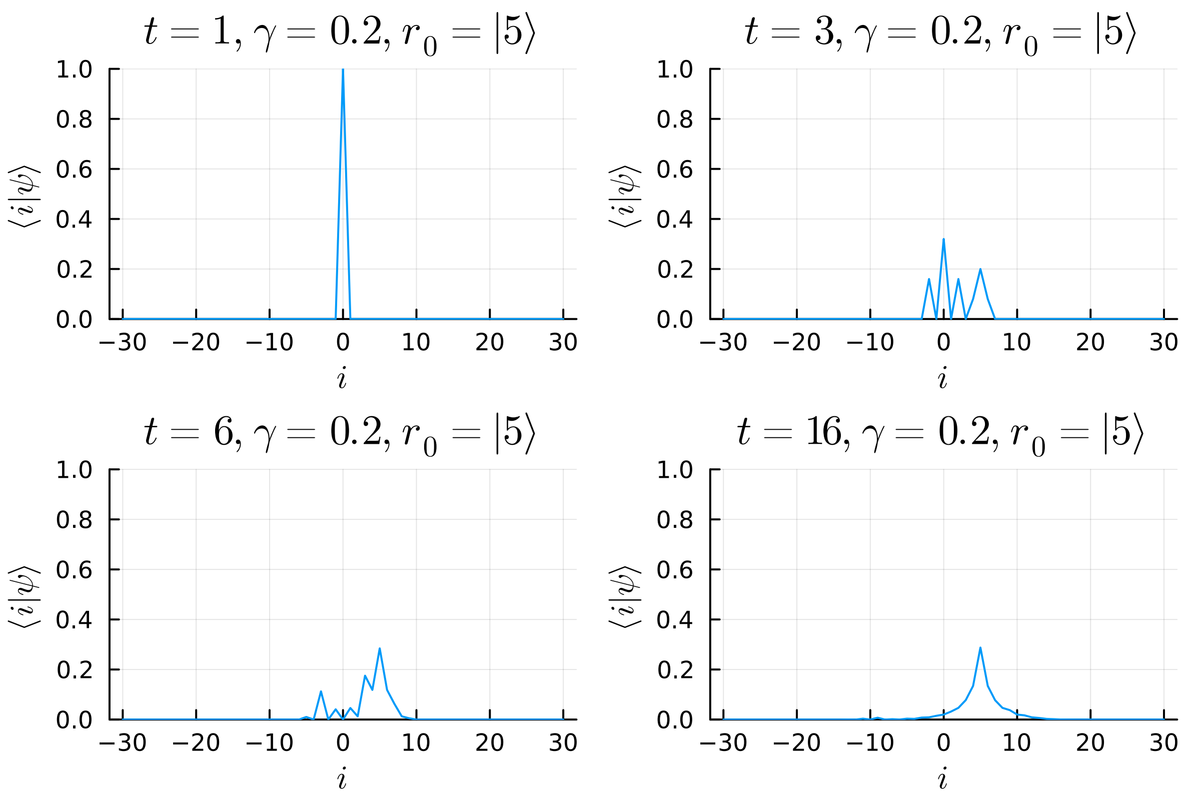


Stochastic Reset Quantum Walk

In the quantum case, the evolution changes from unitary dynamics to a nonunitary CPTP map of the following form.

$$O_{SR}(\rho) = (1-\gamma)\left(U\rho U^{\dagger}\right) + \gamma|r_0\rangle\langle r_0|$$

where U is the walk unitary.



Effect of Resetting on First Hit problem

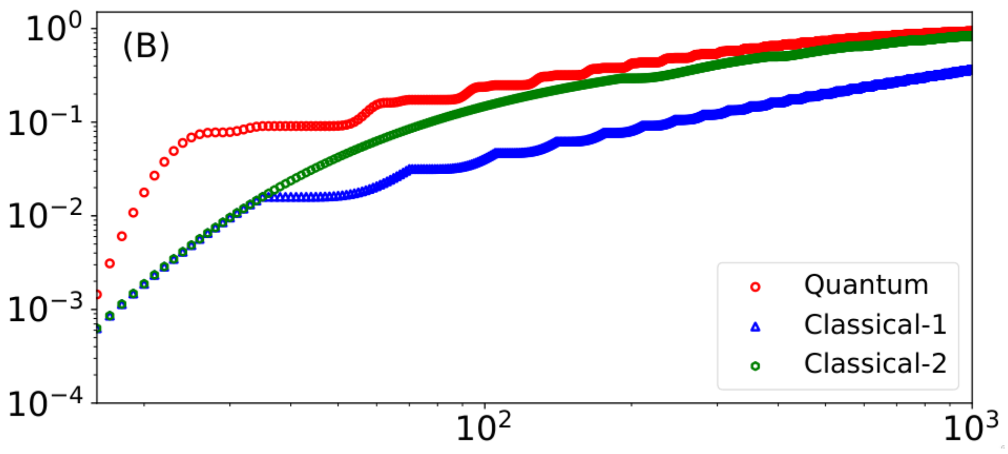
For this analysis, we will consider the simpler "sharp reset" formalism, where the reset time is sampled from a distribution

$$t_r \sim \delta(t - r\tau)$$

Thus we restart after r measurement events (at $t = r\tau$). Once we understand the effect of this kind of reset, gaining intuition for reset times sampled differently is easier.

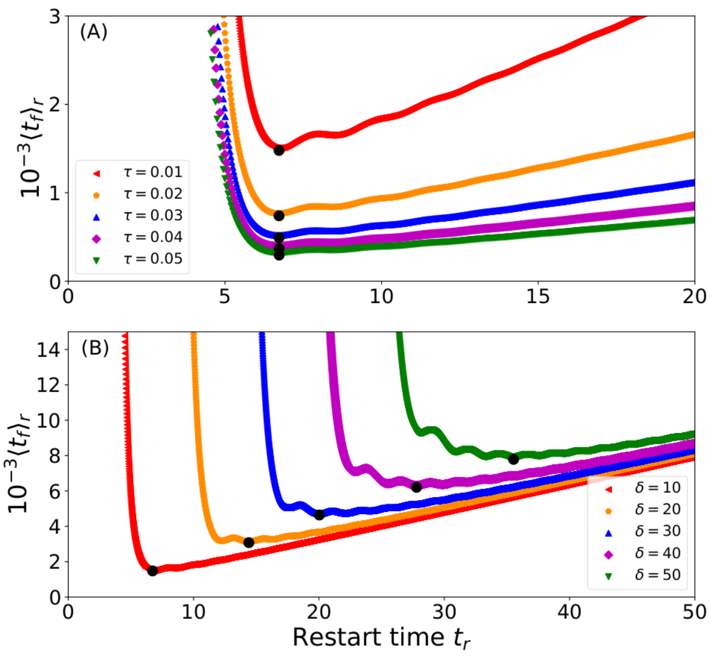
The success probability vs time can now be plotted [15] for the reset case.

Reset Success Probability vs Time



Now we see that the success probability is drastically increased for both cases, but due to the ballistic nature of the quantum walk, we see that the reset quantum walk performs much better than the classical walk.

For a better understanding of the performance of the reset quantum walk with reference to changing reset rates (r) and measurement times (τ), we can plot [15] the mean first hitting time vs these parameters.



Observations

- Deterministic restart leads to zero asymptomatic failure rate
- Eager restarting leads to walker never reaching δ, reducing success rates drastically
- Cautious restart reduces the effect of restart, reducing success rates.
- There exists an optimal r, but this needs to be optimized, which is non trivial for general τ and graph structures.

Can stochastic restarting be better than sharp reset?

No, because even if $\langle r \rangle = r_{\text{optimal}}$, $\langle t_f \rangle > \langle t_f \rangle_{\text{optimal}}$ due to the non-monotonic nature of the curve

The Problem in the Solution

As discussed before, reset rates which are very high or low can end up being detrimental to the success times of the walk. Secondly, there is no clear path as to how to optimize the reset parameter for arbitrary graph structures.

Just as we harnessed the power of quantum superposition to speed up the walk, can we similarly have a superposition between the reset and evolution?

§5 REDACTED

Future Directions and Methods

We intend to use computational simulations, symbolic analysis (possibly aided by a computer), along with a more general model for resetting, drawing inspiration from reference [17] to analyse this new formalism to see whether there are any benefits in the definition of the problem.

REDACTED

We will use, as used until now too, the Julia Language [18] and Julia packages: Yao.jl [19] , QuantumInformation.jl [20] , Pluto.jl [21] , Luxor.jl [22] and PlutoReport.jl [23]

References

1. **[norris_markov_1998]:** [1998] Markov chains; *J. R. Norris*; DOI:
2. **[glasserman_monte_2004]:** [2004] Monte {Carlo} methods in financial engineering; *Paul Glasserman*; DOI:
3. **[griffin_investigation_2019]:** [2019] Investigation of quantum algorithms for direct numerical simulation of the {Navier}-{Stokes} equations; *Kevin P Griffin, Suhas Jain Suresh, Tim J Flint, Wai Hong Ronald Chan*; DOI:10.13140/RG.2.2.22657.81762
4. **[psaltis_markov_2022]:** [2022] Markov {Chains} for {Horizons} {MARCH}. {I}. {Identifying} {Biases} in {Fitting} {Theoretical} {Models} to {Event} {Horizon} {Telescope} {Observations}; *Dimitrios Psaltis, Feryal Özel, Lia Medeiros, Pierre Christian, Junhan Kim, Chi-kwan Chan, Landen J. Conway, Carolyn A. Raithel, Dan Marrone, Tod R. Lauer*; DOI:10.3847/1538-4357/ac2c69
5. **[metropolis_beginning]:** [] The {Beginning} of the {Monte} {Carlo} {Method}; *Nick Metropolis*; DOI:
6. **[de_souza_quantum_2019]:** [2019] Quantum {Walk} to {Train} a {Classical} {Artificial} {Neural} {Network}; *Luciano S. de Souza, Jonathan H.A. de Carvalho, Tiago A.E. Ferreira*; DOI:10.1109/BRACIS.2019.00149
7. **[bae_source_2021]:** [2021] Source {Independent} {Quantum} {Walk} {Random} {Number} {Generation}; *Minwoo Bae, Walter O. Krawec*; DOI:10.48550/ARXIV.2102.02252

8. **[mukherjee_grover_2022]:** [2022] A {Grover} {Search}-{Based} {Algorithm} for the {List} {Coloring} {Problem}; *Sayan Mukherjee*; DOI:10.1109/TQE.2022.3151137
9. **[bonnetain_finding_2022]:** [2022] Finding many {Collisions} via {Reusable} {Quantum} {Walks}; *Xavier Bonnetain, André Chailloux, André Schrottenloher, Yixin Shen*; DOI:
10. **[goldsmith_link_2022]:** [2022] Link prediction with continuous-time classical and quantum walks; *Mark Goldsmith, Guillermo García-Pérez, Joonas Malmi, Matteo A. C. Rossi, Harto Saarinen, Sabrina Maniscalco*; DOI:
11. **[ortega_generalized_2022]:** [2022] Generalized {Quantum} {Google} {PageRank} {Algorithm} with {Arbitrary} {Phase} {Rotations}; *Sergio A. Ortega, Miguel A. Martin-Delgado*; DOI:
12. **[qiskit_quantum]:** [] Quantum {Walk} {Search} {Algorithm}; *Qiskit*; DOI:
13. **[portugal_quantum_2018]:** [2018] Quantum {Walks} and {Search} {Algorithms}; *Renato Portugal*; DOI:10.1007/978-3-319-97813-0
14. **[friedman_quantum_2017]:** [2017] Quantum walks: {The} first detected passage time problem; *H. Friedman, D. A. Kessler, E. Barkai*; DOI:10.1103/PhysRevE.95.032141
15. **[yin_restart_2022]:** [2022] Restart expedites quantum walk hitting times; *Ruoyu Yin, Eli Barkai*; DOI:
16. **[bonomo_first_2021]:** [2021] First passage under restart for discrete space and time: {Application} to one-dimensional confined lattice random walks; *Ofek Lauber Bonomo, Arnab Pal*; DOI:10.1103/PhysRevE.103.052129
17. **[bezanson_julia_2017]:** [2017] Julia: {A} {Fresh} {Approach} to {Numerical} {Computing}; *Jeff Bezanson, Alan Edelman, Stefan Karpinski, Viral B. Shah*; DOI:10.1137/141000671
18. **[luo_yaojl_2020]:** [2020] Yao.jl: {Extensible}, {Efficient} {Framework} for {Quantum} {Algorithm} {Design}; *Xiu-Zhe Luo, Jin-Guo Liu, Pan Zhang, Lei Wang*; DOI:10.22331/q-2020-10-11-341
19. **[gawron_quantuminformationjljulia_2018]:** [2018] {QuantumInformation}.jl—{A} {Julia} package for numerical computation in quantum information theory; *Piotr Gawron, Dariusz Kurzyk, Łukasz Pawela*; DOI:10.1371/journal.pone.0209358
20. **[van_der_plas_fonspplutojl_2022]:** [2022] fonsp/{Pluto}.jl: v0.19.16; *Fons Van Der Plas, Michiel Dral, Paul Berg, Παναγιώτης Γεωργακόπουλος, Rik Huijzer, Nicholas Bochenski, Alberto Mengali, Benjamin Lungwitz, Connor Burns, Hirumal Priyashan, Jerry Ling, Eric Zhang, Felipe S. S. Schneider, Ian Weaver, Rogerluo, Shuhei Kadowaki, Zihua Wu, Jelmar Gerritsen, Rok Novosel, Supanat, Zachary Moon, Luis-Mueller, Michael Abbott, Nicholas Bauer, Patrick Bouffard, Satoshi Terasaki, Shashank Polasa, Jacob S. Zelko*; DOI:10.5281/ZENODO.7340569
21. **[juliagraphics_luxorjl]:** [] Luxor.jl - {Simple} drawings using vector graphics; {Cairo} "for tourists!"; *JuliaGraphics*; DOI:
22. **[sambrani_plutoreportjl]:** [] {PlutoReport}.jl - {Make} awesome reports and talk slides in {Pluto}!; *Dhruva Sambrani*; DOI: