Quantum Walks and Monte Carlo

Tayyab Ali

alitayyab.pu@gmail.com

Aug 2025

Abstract

The main aim of this project is to highlight how quantum computers can be beneficial for simulating statistical processes. We plan to use quantum hardware to run Galton board simulations and see how much faster they can be compared to classical methods. We'll start with a simple quantum circuit that models just one quantum peg, and then we'll gradually expand it to more levels to perform detailed simulations on quantum devices.

1. Overview

In classical probability, a random walk is a key model that illustrates how a particle moves in steps, choosing random directions at each turn—usually with an equal chance of going left or right. This concept is commonly applied to things like diffusion, stock market trends[1], and various random processes. The Galton board, or quincunx, is a physical representation of this idea. It has a series of pegs that a ball, dropped from the top, bounces off of, moving either left or right with equal odds. When the ball lands in bins at the bottom, it ends up forming a binomial distribution. This setup nicely showcases the central limit theorem, especially as you add more levels, which leads toward a normal distribution. Even though classical random walks and Galton boards offer clear and practical ways to visualize randomness, running simulations for them can be quite demanding on resources, especially when you increase the number of steps or pegs, making it harder to scale for more complex systems.
Quantum random walks take the basic idea of classical random walks and give it a twist using principles from quantum mechanics, like superposition and interference. Instead of strictly following a set path or relying on classical probabilities, in a quantum walk, the 'walker' can be in many places at once, thanks to a unitary operation. The movement is controlled by a quantum coin operator, which decides the likelihood of moving left or right. Because of quantum interference, the probabilities of these movements can either amplify or cancel each other out, leading to probability distributions that are quite different from the usual classical Gaussian ones. Notably, these distributions often spread out more quickly, with the standard deviation increasing in a linear way as the number of steps goes up, rather than the classical model where it grows with the square root of the number of steps. The quantum Galton board is a way to visualize this quantum walk, set up as a quantum circuit. Each peg on the board represents a quantum coin operation—usually a controlled rotation or a Hadamard gate—and the quantum walker's journey is depicted through the states of qubits. By adjusting the rotation angles at each peg, these boards allow for the study of biased quantum walks and can simulate a variety of distributions that classical methods can't touch. This arrangement gives a significant speed boost when simulating multiple paths at once, as the quantum circuit inherently captures all the potential routes in superposition, highlighting how quantum computing can enhance simulations of complicated statistical processes[2, 3].

2. Mapping the Problem to Quantum Circuit

To create a quantum Galton board with a quantum circuit, we treat the different routes the 'ball' can take as quantum states of qubits. Each qubit holds information about its position or the choices it can make at each peg. The aim here is to generate superpositions of all the possible paths and then manipulate those to mimic the behavior of a quantum random walk. The quantum gates we use act like 'quantum pegs' that divide and direct the amplitudes, much like how a ball bounces left or right, but with the added twist of quantum interference.

```python
from qiskit import QuantumCircuit

qc = QuantumCircuit(4, 2)
qc.h(0)                   # Quantum coin flip
qc.x(2)                   # Initialize path qubit
qc.cswap(0, 1, 2)         # First conditional branch
qc.cx(2, 0)               # Entangle coin with path
qc.cswap(0, 2, 3)         # Second conditional branch
qc.measure([1, 3], [0, 1]) # Measurement
qc.draw('mpl')
```

Code 1. Qiskit implementation of 1-level QGB

The given quantum circuit implements a one-level quantum Galton board (QGB) using four qubits and two classical bits. Qubit q_0 serves as the quantum coin, while q_1, q_2, and q_3 represent possible path positions of the walker. The Hadamard gate H on q_0 creates the superposition $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$, simulating an unbiased coin toss. The X gate on q_2 initializes the walker at the central position. The first controlled-swap (Fredkin) gate swaps q_1 and q_2 conditioned on the coin state, representing the walker's first decision point. The CX gate between q_2 and q_0 introduces entanglement and enables interference effects characteristic of quantum walks. A second controlled-swap between q_2 and q_3 allows movement toward the other branch. Only q_1 and q_3 are measured because they encode the walker's final position, while intermediate qubits preserve superposition until the end. This configuration represents a single branching level of the QGB. For higher levels, ***each additional level would require introducing further controlled-swap gates between the next adjacent position qubits***, effectively extending the branching structure to simulate deeper quantum walks[4].

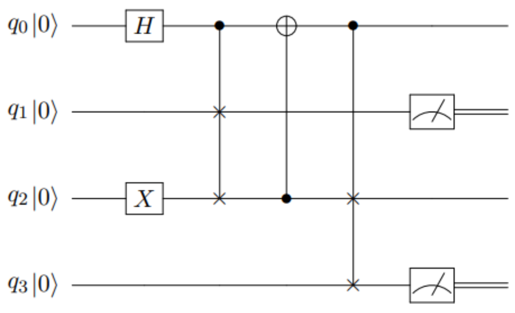

Figure 1. Circuit corresponding 1[4]

2.1. N-level QGB

The single-level Quantum Galton Board (QGB) circuit is essentially the building block for simulating quantum random walks. We can take this unit and use it repeatedly to create an n-level QGB, which lets us simulate more complex branching structures that show full quantum interference effects. At every level, we reset the quantum coin qubit and put it back into superposition. This way, the branching decisions are independent at each stage. We use controlled-swap and controlled-X operations to adjust the amplitude to the left or right, all while creating interference patterns that are quite different from what you'd see in classical walks. In the end, after n iterations, we only measure the qubits at the odd-indexed positions, keeping those quantum correlations intact right up until the last step.

Algorithm: Quantum Galton Board Circuit Construction

Input: Number of levels n (integer, $n \geq 1$)
Output: Quantum circuit simulating a Quantum Galton Board of n levels

```
1. Initialize Parameters:
     • $total\_qubits = 2n + 2$
     • $middle = n + 1$                          (initial ball position)
     • Create quantum register $q[0 \dots total\_qubits - 1]$
     • Create classical register $c[0 \dots n]$
     • Initialize quantum circuit $qc$ with $q$ and $c$
2. Initial Superposition and Ball Placement:
     • Apply $H(q[0])$                       (coin in superposition)
     • Apply $X(q[middle])$                  (place ball in center)
     • $ball\_positions = [middle]$
3. Simulate Each Level: For each $step$ in $0 \dots n - 1$:
   (a) Apply $RESET(q[0])$
   (b) Apply $H(q[0])$
   (c) $positions\_this\_step = sorted(ball\_positions)$
   (d) $new\_positions = \{\}$
   (e) For each $(i, pos)$ in $positions\_this\_step$:
         • $left = pos - 1, right = pos + 1$
         • If $1 \leq left < total\_qubits$ and $right < total\_qubits$:
              i. $CSWAP(q[0], q[left], q[pos])$
             ii. $CX(q[pos], q[0])$
            iii. $CSWAP(q[0], q[pos], q[right])$
             iv. If $step \geq 1$ and $i$ is not last:
                 $CX(q[right], q[0])$
              v. Add $left$ and $right$ to $new\_positions$
   (f) $ball\_positions = sorted(new\_positions)$
4. Measurement:
     • For $j = 0 \dots n$: $measure\_qubit = 2j + 1$
     • $MEASURE(q[measure\_qubit]) \rightarrow c[j]$
```

3. Local Simulations

- The simulation of the Quantum Galton Board (QGB) circuit is performed on a quantum simulator backend, such as `AerSimulator`, which allows for efficient local execution. Initially, the QGB circuit is transpiled and optimized to match the backend's gate set and improve performance. The circuit is then run with a large number of shots (e.g., 20,000) to collect sufficient measurement statistics.
- Measurement outcomes are obtained as bitstrings, which represent the final positions of the "ball" in the Galton board. Due to qubit-to-bit mapping and endianness conventions, the bitstrings may be reversed to align with the logical ordering of output positions. Each bitstring is mapped to an integer corresponding to a physical position on the board.
- Statistical analysis is then performed on these mapped results, including computing the mean, variance, and frequency of occurrences at each position. ***Additionally, results are grouped into blocks (e.g., blocks of 8 shots), and sums within these blocks are analyzed to study aggregated behavior.***
- Finally, the empirical distribution of block sums is compared against theoretical distributions, such as the normal distribution, via plotting. This comparison validates the quantum simulation results against classical expectations and helps reveal the characteristic distribution patterns generated by the QGB circuit.

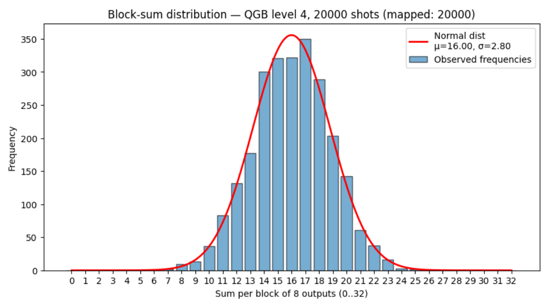

Figure 2. Local Simulation: yields normal distribution

4. Obtaining different Distributions

To obtain different target distributions in the Quantum Galton Board (QGB) simulation, the Hadamard gate (which creates an equal superposition) can be replaced by a rotation around the X-axis, $R_x(\theta)$, with an appropriately chosen angle θ. The parameter θ controls the bias of the quantum coin toss, skewing the probability amplitudes toward specific outcomes and thus shaping the final distribution of the ball positions. For example, $\theta = \pi/2$ reproduces the balanced Hadamard walk distribution, while smaller or larger angles generate biased or exponential-like distributions.

```python
from qiskit import QuantumCircuit

theta = 0.5
qc = QuantumCircuit(4, 2)
qc.rx(theta, 0)
qc.x(2)
qc.cswap(0, 1, 2)
qc.cx(2, 0)
qc.cswap(0, 2, 3)
qc.measure([1,3], [0,1])  # Measurement

qc.draw('mpl')
```

Code 2. Qiskit implementation of 1-level Biased-QGB

References

[1] Eugene F Fama. "Random walks in stock market prices". In: *Financial analysts journal* 51.1 (1995), pp. 75–80.

[2] Julia Kempe. "Quantum random walks: an introductory overview". In: *Contemporary Physics* 44.4 (2003), pp. 307–327.

[3] Ahmed A Abd El-Latif, Abdullah M Iliyasu, and Bassem Abd-El-Atty. "An efficient visually meaningful quantum walks-based encryption scheme for secure data transmission on IoT and smart applications". In: *Mathematics* 9.23 (2021), p. 3131.

[4] Mark Carney and Ben Varcoe. "Universal Statistical Simulator". In: *arXiv preprint arXiv:2202.01735* (2022).