Random Walk and Electric Network in Two Dimensions

Names:

Kexin Liu

Yunzhao Li

Yinqi Qiao

Instructor: Omidali Aghababaei Jazi

STA348H5 F2024

Introduction to Stochastic Processes

Contents

1 Introduction

Random walks and electric networks are fundamental models in mathematics and physics, with applications ranging from circuit analysis to probabilistic simulations. Despite their apparent differences, these two systems share a deep mathematical connection, where harmonic properties govern both voltage distributions in electric networks and transition probabilities in random walks. This relationship provides valuable insights for developing solutions across various fields.

While the theoretical connection between random walks and electric networks is well-established by Doyle and Snell (1984), empirical validation of this equivalence through computational simulations remains limited. Most prior studies focus on theoretical derivations or specific case studies, but the application of modern simulation tools to validate these concepts across diverse network structures has been relatively unexplored. This project addresses this gap by demonstrating the equivalence between voltage and probability distributions using simulation tools like LTspice and Python. Such insights have practical implications for optimizing electrical systems and efficient network design.

This project aims to explore the equivalence between random walks and electric networks through two key objectives:

- To demonstrate that the voltage distribution in a 2D electric network can be approximated using random walk probabilities.
- To extend this framework to approximate current distributions in networks with non-uniform resistances.

The project examines two network models: a 2D grid network and a star-shaped network, each chosen to highlight distinct aspects of their equivalence. The 2D grid network, with its symmetric and structured topology, is well-suited for theoretical validation. In contrast, the star-shaped network introduces asymmetry and irregularity, offering a closer representation of real-world scenarios. This project uses Python to simulate random walk probabilities and LTspice for voltage and current distributions, directly comparing the two systems. The results validate their harmonic properties and show how their equivalence can approximate physical systems using probabilistic tools.

2 Review Topic

2.1 Definition and Examples

2.1.1 Introduction to 1-Dimensional Random Walk

In the context of a random walk on integers, consider the probabilities associated with reaching a specific boundary before another. Let $p(x)$ denote the probability of reaching an upper boundary N before reaching a lower boundary 0, starting from an intermediate point x on the interval $[0, N]$. The function $p(x)$ is governed by three fundamental properties (Doyle and Snell 1984) that provide a complete characterization of this process:

Property (a): $p(0) = 0$

This property states that if the walk starts at the lower boundary 0, the probability of reaching the upper boundary N before returning to 0 is zero. The reason is that 0 is an **absorbing boundary**, where the walk cannot leave this state once it has reached it.

Property (b): $p(N) = 1$

Intuitively, if the walk starts at the upper boundary N, the probability of reaching N is simply just one as it's already there.

Property (c): $p(x) = \frac{1}{2}p(x-1) + \frac{1}{2}p(x+1)$

For any interior point x, where $1 \leq x \leq N-1$, the probability $p(x)$ of reaching N before 0 satisfies this equation.

This means that $p(x)$ is the **average** of the probabilities of reaching N from its two neighboring points, $x-1$ and $x+1$. The reasoning behind this relation comes from the **law of total probability**.

2.1.2 Introduction to 2-Dimensional Random Walk

A **2-dimensional random walk** occurs on a lattice of integer coordinate points in the plane. A walker starts at a given point (a, b) and at each step moves to one of the four neighboring points:

$$(a+1, b), \quad (a-1, b), \quad (a, b+1), \quad (a, b-1),$$

with equal probability of $\frac{1}{4}$ (Doyle and Snell 1984).

For a set of lattice points $S = D \cup B$, where:

- D: Represents **interior points**, which have all four neighbors within S.
- B: Represents **boundary points**, which are connected to at least one neighbor in D.

the probability of reaching a boundary point, starting from any internal point $(a, b) \in D$ is:

$$p(a,b) = \frac{p(a+1,b) + p(a-1,b) + p(a,b+1) + p(a,b-1)}{4}.$$

2.1.3 Harmonic Function

In both 1- and 2-dimensional walks, the probabilistic equations are indicative of harmonicity.

A **harmonic function** is a twice-differentiable function f defined on a domain D that satisfies Laplace's equation: $\nabla^2 f = 0$, where $\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$ in 2D (Kleinberg and Tardos 2005).

One important property of harmonic functions is the **Mean Value Property**, which states that for a harmonic function u defined on a disk D, the value of u at the center of the disk equals the average value of u on the circumference of the disk. Similarly, the probabilistic equations governing random walks in 1- and 2-dimensions satisfy a discrete version of the harmonic mean value property: the probability at any point equals the average of the probabilities at its neighbors.

Additionally, the **Maximum Principle** for harmonic functions asserts that a harmonic function attains its maximum and minimum values only on the boundary of its domain (Kleinberg and Tardos 2005). In the case of a random walk, the maximum value is typically 1, and the minimum value is 0, both of which occur at the boundaries.

2.1.4 Introduction to Electric Network in 2-Dimension

A **2D electric network** is a grid-like structure of interconnected nodes (represented by points) linked by resistors (Doyle and Snell 1984). Each connection allows for the flow of electric current, driven by a potential difference (voltage) between the nodes.

Voltage (V):

Voltage is the electric potential difference between two nodes in the network. It drives the flow of current through the network and is measured in volts (V). Mathematically, the voltage difference between two nodes i and j is:

$$v_{ij} = v_i - v_j.$$

Current (I):

Current is the rate of flow of electric charge through a resistor or edge in the network, measured in amperes (A). It is proportional to the voltage difference across the resistor and inversely proportional to the resistance. According to **Ohm's Law**, the current flowing through an edge between two nodes i and j is:

$$i_{ij} = \frac{v_{ij}}{R}.$$

Resistance (R):

Resistance quantifies the opposition to current flow through a resistor or edge in the network. It is measured in ohms (Ω). Its reciprocal $C = \frac{1}{R}$ is known as the conductance. Throughout this project, we assume uniform resistance in most cases.

Kirchhoff's Current Law:

The law states that for any vertex in the network, the total current flowing into the vertex equals the total current flowing out. Mathematically, for a vertex x in the graph:

$$\sum_{y:\ x\sim y} i_{xy} = 0 \quad \forall x \in V,$$

where i_{xy} is the current flowing from vertex x to its neighbor y (Kleinberg and Tardos 2005).

By combining Kirchhoff's Law with Ohm's Law and assuming equal resistance, the current contributions from all neighboring points can be expressed as:

$$\frac{v(a+1,b)-v(a,b)}{R} + \frac{v(a-1,b)-v(a,b)}{R} + \frac{v(a,b+1)-v(a,b)}{R} + \frac{v(a,b-1)-v(a,b)}{R} = 0.$$

Simplifying this equation, we have:

$$v(a,b) = \frac{v(a+1,b)+v(a-1,b)+v(a,b+1)+v(a,b-1)}{4}.$$

The source and target nodes serve as fixed boundary conditions in an electric network, defining the voltage extremes and enabling the flow of current. The source node is set to a fixed voltage (1 volt in this project), typically representing the starting point of current flow. The target node, often referred to as the ground, is the node where the potential is fixed to 0 volts.

Similar to the probabilistic equation in a random walk, the voltage equation above satisfies the properties of a harmonic function.

2.2 Details of the Topic

2.2.1 Goal 1: Approximate Voltage Distribution

The primary goal of this project is to use simulated random walks to approximate the voltage distribution of electric circuits. This approach builds upon the theoretical equivalence between the probability distribution in random walks and the voltage distribution in electric networks.

In both systems, the solutions satisfy the harmonic property, which states that the value at any interior point is the average of its neighboring points. When combined with fixed boundary conditions, these harmonic solutions form a Dirichlet problem.

The **Dirichlet problem** is defined by its **Uniqueness Principle** (Doyle and Snell 1984), which states that if two functions satisfy the same partial differential equation within a domain and have identical boundary conditions, they must be identical. This guarantees that the solutions for random walk probabilities and voltage distribution are the same, implying $p(x) = v(x)$.

2.2.2 Goal 2: Approximate Current Distribution

This section discusses how simulated random walks can be used to approximate the current distribution in electric circuits. Here, we discuss the general case, where resistance is not necessarily constant across the circuit.

Conductance:

Conductance, denoted as C_{xy}, is the reciprocal of resistance (R). Mathematically, $C_{xy} = \frac{1}{R_{xy}}$.

The conductance is connected to the transition probabilities of a random walk on the graph of the network, where the probability P_{xy} of transitioning from node x to node y is given by:

$$P_{xy} = \frac{C_{xy}}{C_x},$$

where $C_x = \sum_y C_{xy}$ is the total conductance at node x (Doyle and Snell 1984).

When resistors are equal, transition probabilities to neighbors are uniform. With unequal resistors, probabilities are proportional to edge conductance, and favour lower resistance paths.

Metrics in the Random Walk Model:

- **Expected Number of Visits (u_x)**:

  The expected number of times a walker visits node x before eventually reaching the target node b, starting from the source node a (Doyle and Snell 1984).

- **Normalized Voltage (v_x)**:

  Defined as:

  $$v_x = \frac{u_x}{C_x},$$

  where C_x is the total conductance at node x. This relationship highlights the proportionality between normalized visit counts and the potential at a node (Doyle and Snell 1984).

Current Flow Between Nodes (i_{xy})

The current i_{xy} flowing between two nodes x and y can be expressed as:

$$i_{xy} = u_x P_{xy} - u_y P_{yx},$$

and it can be interpreted as the **net expected number of movements** along the edge from x to y in a random walk (Doyle and Snell 1984).

In this case, i_{xy} represents the unit current flow, where the total current entering at the source node a and leaving at the target node b sums to 1, expressed as:

$$\sum_y i_{ay} = 1.$$

The derived currents i_{xy} are proportional to the actual currents in an electric circuit with a unit voltage applied. This proportionality validates the consistency between the probabilistic distributions from the random walk model and the physical distributions observed in the electric network. The constant of proportionality is also known as the **effective resistance** of the circuit, which serves as a measure of the circuit's opposition to current. Theoretically, a lower effective resistance suggests a more efficient circuit for transferring current between the two nodes.

---

3 Application

3.1 Goal 1: Approximate Voltage Distribution

The primary goal is to approximate the voltage distribution in an electric network using simulated random walks. We will investigate two circuits, one on a regular 2-dimensional lattice grid and one on an star-shaped figure.

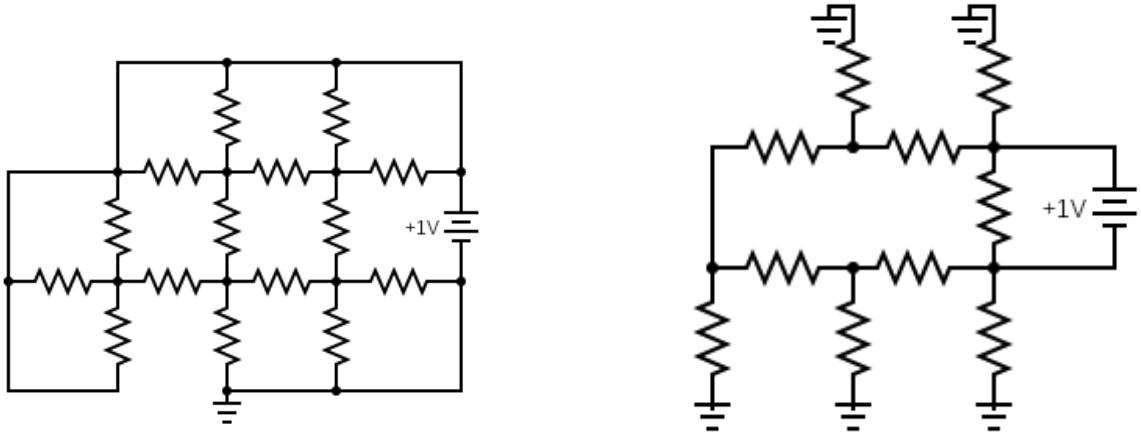

Figure 1: (a) 2-Dimensional Lattice Circuit (Doyle and Snell 1984), (b) Star-Shaped Circuit.

In Figure 1, the zigzag symbols represent resistors, each with a resistance of 1 ohm. The vertical lines marked with "+" and "−" represent a power source providing a voltage of 1, often considered the "upper" boundary point with a potential of 1. In the context of a random walk, this corresponds to a boundary point where the probability equals 1. The three decreasing horizontal lines represent the ground, serving as the reference point with zero potential. In a random walk, ground points correspond to locations where the probabilities are 0.

3.1.1 Methodology

We begin by explaining the algorithms that we be used in our simulation, which included generating electric circuits and stimulating random walks.

3.1.1.1 Construction of Electric Circuits In this project , LTspice was utilized to generate the potential distribution at each node of the electric network.

The circuit was constructed by selecting components from the LTspice component library, including 1-ohm resistors (R), a voltage source (V) of 1 volt, and a ground node (GND) to match the circuit diagram. We

then used the wire tool to connect the components according to the circuit designs shown in Figure 1. After constructing the circuit, LTspice was used to simulate and calculate values of potential at each internal node.

These simulated values represent the theoretical voltage distribution of the circuit and serve as the reference for validating the random walk simulation.

3.1.1.2 Simulation of Random Walks We used Python to simulate the random walk models. Prior to this, we converted the circuit diagrams into their corresponding random walk graphs by following these steps:

1. **Identify Nodes:**
    - Circuit junctions where resistors connect become random walk states, and are often represented by points on the graph.
    - Each internal node in the circuit corresponds to an internal point in the random walk graph.
2. **Connections:**
    - The resistors between the nodes in the circuit correspond to the edges in the graph. Each edge represents a possible step in the random walk. Transition probabilities are determined by the conductance between nodes. In this section, probabilities are equal since we assumed equal resistance.
3. **Boundary Points:**
    - Nodes directly connected to the voltage source, with no resistor in between, correspond to the boundary condition “1” in the graph.
    - Nodes directly connected to the ground, with no resistor in between, correspond to the boundary condition “0” in the graph.
    - Both nodes serve as absorbing states, where the walk cannot leave once reached the state.

As a result, we created two random walk diagrams corresponding to the two circuits in Figure 1:

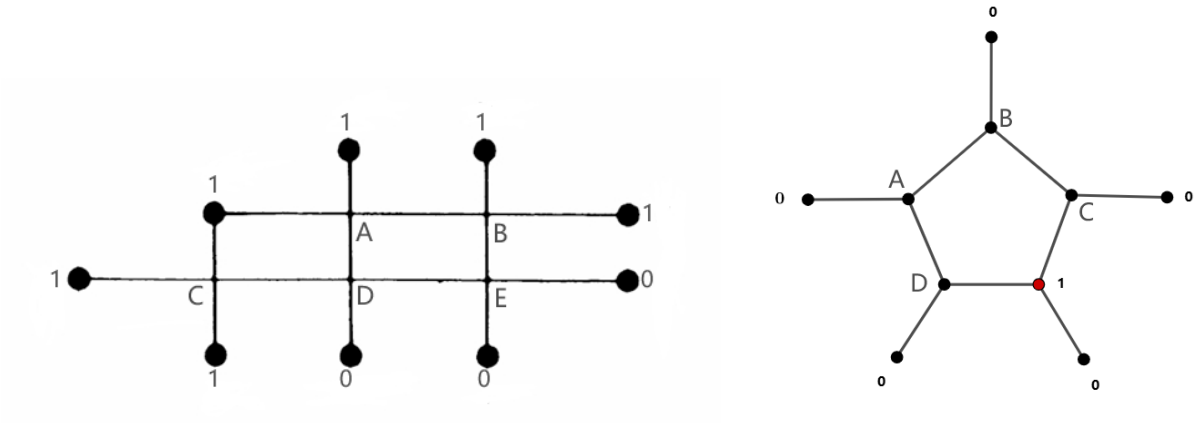

Figure 2: (a) 2-Dimensional Lattice Random Walk (Doyle and Snell 1984), (b) Star-Shaped Random Walk.

We then used Python to simulate the respective random walk on a 2D graph, where each point is represented by a position and has defined neighboring connections. Starting from internal nodes, the algorithm performs 1000 simulations to determine the probability of reaching boundary nodes labeled as "1" before those labeled as "0". Each random walk terminates upon reaching either boundary, and probabilities are calculated based on these outcomes. The results are labeled on the graph to illustrate the probabilities of each internal point.

To analyze the relationship between the circuit's voltage distribution and the random walk's probability distribution, we compare the voltage at each internal node in the circuit, derived from LTspice simulations, with the probability of reaching any "1" node before a "0" node in the random walk simulation.

3.1.2 Results and Analysis

The graphs generated using LTspice and Python will be shown below. We will observe the distribution approximation for each shape respectively.

3.1.2.1 2-Dimensional Lattice

We first observe the results for the 2-Dimensional lattice grid.

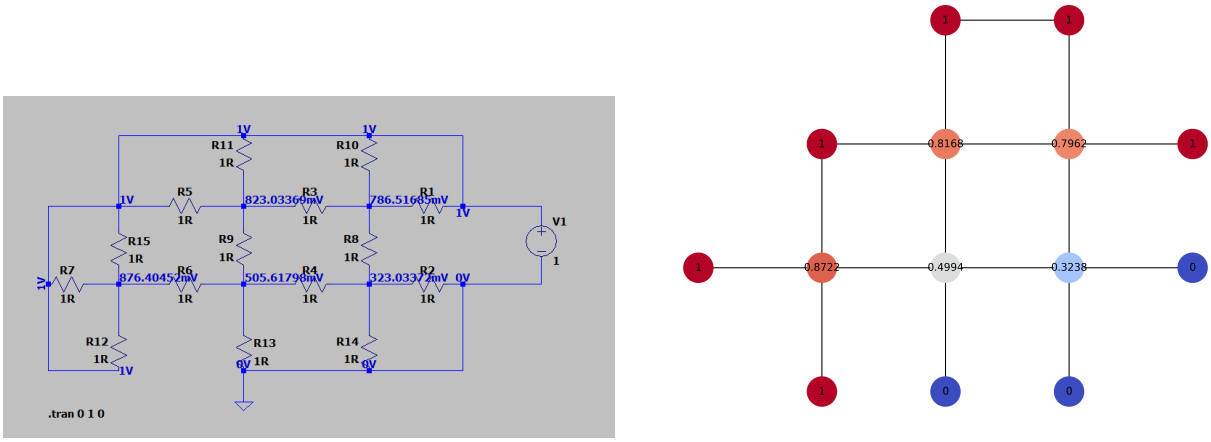

Figure 3: (a) Voltage distribution of the electric circuit generated using LTspice, showing the potential at each node with 1V applied to source nodes and 0V at ground nodes. (b) Probability distribution of reaching "1" before "0" for internal points in the random walk simulation.

Focusing on the internal points (nodes not directly connected to the voltage source or ground) in Figure 3, we observe that nodes closer to "1" nodes exhibit higher values, while those near "0" show lower values.

An example of an internal node is the one located between R_3 and R_5, where the potential is $832.0337mV$ or $0.8320V$. Its corresponding point in the random walk model has a simulated value of 0.8168, showing a close approximation. Observing the values for all internal points in Table 1 reveals a strong alignment between the random walk probability distribution and the voltage distribution in the electric network.

Table 1: Comparison of Voltage and Probability Distributions

Node	Voltage Distribution (V)	Probability Distribution	Absolute Difference
A	0.8230	0.8168	0.0062
B	0.7865	0.7962	0.0097
C	0.8764	0.8722	0.0042
D	0.5056	0.4994	0.0062
E	0.3230	0.3238	0.0008

3.1.2.2 Star Shape

The simulation procedures for the star-shaped model are very similar to those used for the lattice grid.

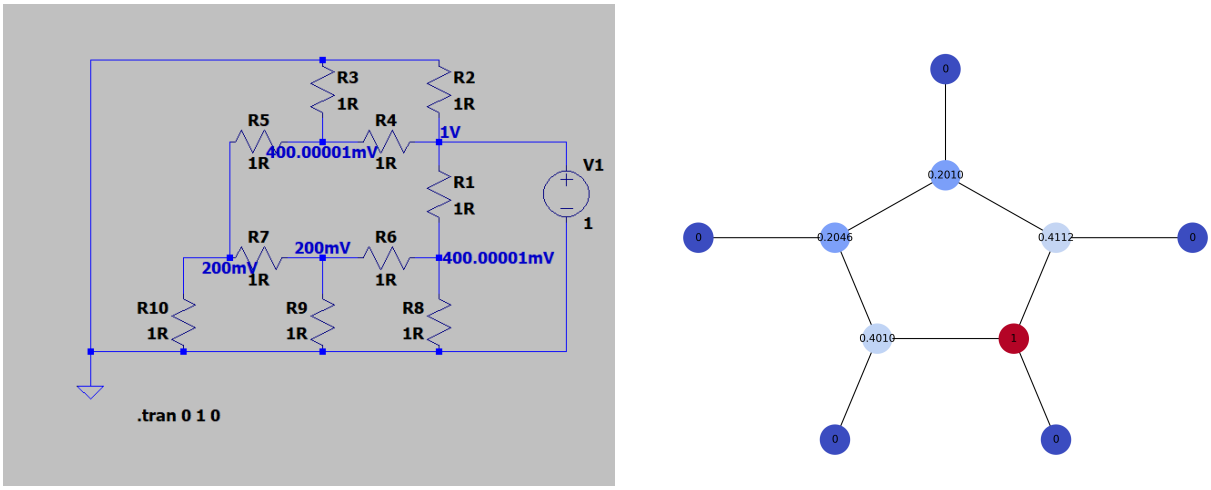

Figure 4: (a) Voltage distribution of the electric circuit generated using LTspice, showing the potential at each node with 1V applied to the source node and 0V at ground nodes. (b) Probability distribution of reaching ”1” before ”0” for internal points in the random walk simulation.

Similar to the 2-dimensional lattice grid, the star-shaped model also demonstrates a pattern where potentials are higher near the source and lower near the ground. As shown in Figure 4(b), the two nodes directly connected to “1” have higher probabilities, approximately 0.4, while the other two are around 0.2.

Table 2: Comparison of Voltage and Probability Distributions

Node	Voltage Distribution (V)	Probability Distribution	Absolute Difference
A	0.2	0.2046	0.0046
B	0.2	0.2010	0.0010
C	0.4	0.4112	0.0112
D	0.4	0.4010	0.0010

Table 2 further validates the alignment between the voltage distribution of the circuit and the probability distribution from the random walk simulation, regardless of the network’s geometry.

3.2 Goal 2: Approximate Current Distribution

The second goal is to approximate the proportional current distribution in an electric network using simulated random walks. To achieve this, we used a textbook example involving resistors of unequal values, as previously discussed.

3.2.1 Methodology

The electric circuit was modeled using LTspice for the subsequent analysis. In this instance, we specified the potential and resistor values, then calculated the expected current flow accordingly. The resulting current flow serves as a reference for comparing the net number of movements along each edge in the simulated random walk.

We used the relation $P_{xy} = \frac{C_{xy}}{C_x}$ to calculate the probability matrix for this circuit. With this matrix, we created the corresponding random walk diagram using the same method outlined in Goal 1.

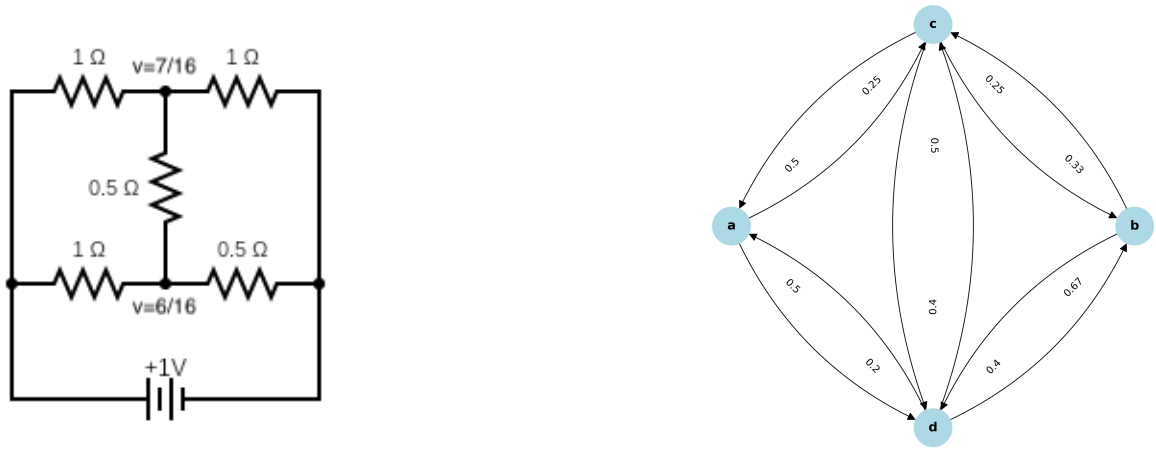

Figure 5: (a) Electric Circuit with Unequal Resistance (Doyle and Snell 1984), (b) Corresponsive Random Walk Model (Doyle and Snell 1984).

Next, we used Python to estimate the net crossings of each edge in a directed graph during a simulated random walk that starts at the source node a and terminates upon reaching the absorbing node b. During each random walk simulation, a walker moves along the graph edges based on the transition probabilities, and the net crossings (forward and reverse) for each edge are tracked. 10,000 simulations were performed to average the crossings, providing an estimate of the net crossing count for each edge.

3.2.2 Results and Analysis

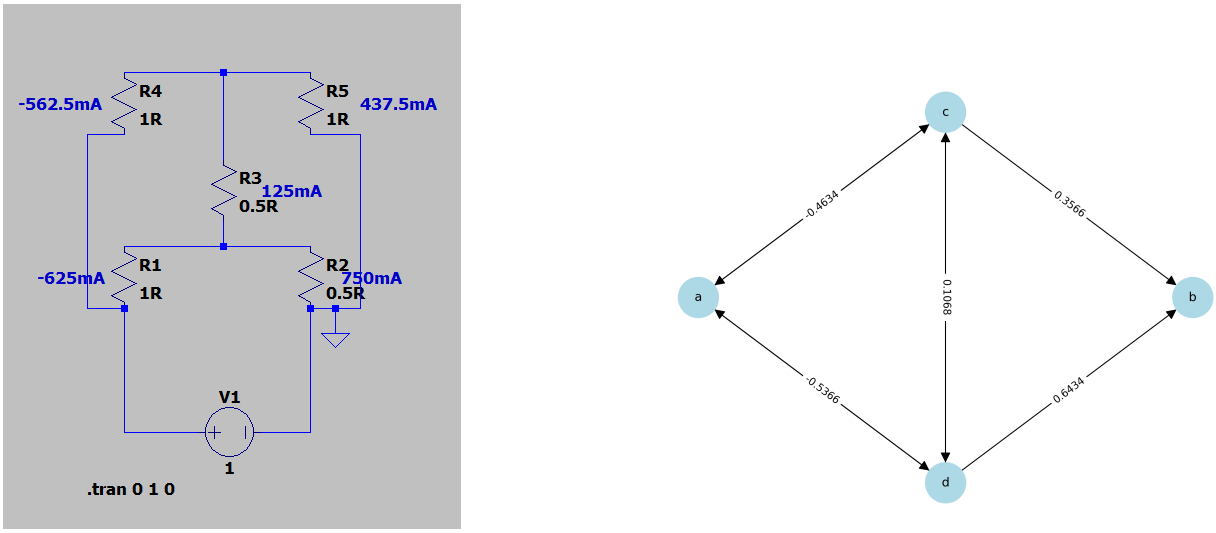

Figure 6: (a) Current flow through each resistor in the electric network, with a 1V source applied between node a (source) and node b (ground). (b) Visualization of the net crossings for each edge in the random walk, starting at node a and terminating at node b (Doyle and Snell 1984).

The graphs in Figure 6 show different values, which is expected since, as previously discussed, the estimated net crossings from the simulated random walk are based on unit current, while the actual circuit is based on unit voltage. As shown in Table 3, despite these differences, the ratios between the two values for each edge are similar, indicating a constant proportionality.

Table 3: Comparison of Current and Net Crossing

Edges	Current Flow (A)	Estimated Net Crossing	Ratio
a->c	-0.4634	-0.5620	0.8245552
a->d	-0.5366	-0.6250	0.8585600
c->d	0.1068	0.1250	0.8544000
c->b	0.3566	0.4375	0.8150857
d->b	0.6434	0.7500	0.8578667

The constant of proportionality, calculated to be approximately $0.85\,\Omega$, represents the effective resistance of the circuit. This relatively low value indicates that the circuit permits a high current flow for a given voltage.

4 Conclusion

This project successfully demonstrated the equivalence between random walks and electric networks in approximating voltage and current distributions, with distinct topologies tailored to each goal.

For **Goal 1** (Voltage Distribution), our project analyzed a 2D lattice and a star-shaped structure, showcasing how random walk probabilities accurately replicate voltage distributions governed by harmonic properties. The close alignment between random walk results and LTspice simulations validates this method for practical applications. These include **heat transfer systems**, where voltage distribution mirrors temperature profiles (Fourier 1822), and **network optimization**, where voltage-based modeling aids in assessing data flow reliability and efficiency Kleinberg and Tardos (2005). This result underscores the computational efficiency of random walks in solving such problems across diverse network geometries.

For **Goal 2** (Current Distribution), the analysis focused on a 2D lattice and a network with unequal resistances. The results revealed a strong proportionality between random walk net crossings and actual current flow, with the proportionality constant representing the effective resistance of the network. The effective resistance, derived from this relationship, provides a quantifiable measure of network efficiency. In the 2D lattice, the uniform resistances demonstrated balanced current paths, while the unequal resistance topology highlighted how heterogeneity concentrates current in low-resistance paths. These findings have direct applications to **power grids**, where effective resistance informs energy loss minimization (Meier 2006), **traffic networks**, where it helps identify bottlenecks and optimize flow (Wu et al. 2016), and **communication networks**, offering insights into efficient data routing strategies (Newman 2018).

By incorporating multiple topologies and validating results through comparison, our project emphasizes the versatility of random walks a computational tool for modeling real-world systems. Future research could extend these methods to dynamic or higher-dimensional networks, expanding their applicability and impact in practical scenarios.

Limitation and Future Work

While we attempted to extend the equivalence between random walks and electric networks to higher dimensions, technical challenges and limitations in our current expertise prevented us from fully implementing and validating these models. Future research could explore higher-dimensional or dynamic networks, as well as develop advanced computational tools to overcome these challenges. This expansion would further enhance the versatility and applicability of random walk models in real-world scenarios.

5 Reference

1.Doyle, Peter G., and J. Laurie Snell. 1984. Random Walks and Electric Networks. Washington, D.C.:Mathematical Association of America.

2.Fourier, Joseph. 1822. Théorie Analytique de La Chaleur. Paris: Firmin-Didot.

3.Kleinberg, Jon, and Éva Tardos. 2005. Algorithm Design. Pearson.

4.Meier, Alexandra von. 2006. Electric Power Systems: A Conceptual Introduction. Wiley.

5.Newman, Mark E. J. 2018. Networks: An Introduction. 2nd ed. Oxford University Press.

6.Strauss, Walter A. 2007. Partial Differential Equations: An Introduction. 2nd ed. John Wiley & Sons.

7.Wu, Xiang, Hao Ma, Wei Zhang, and Wei Wang. 2016. "Traffic Bottleneck Analysis and Optimization Using Network Flow Models." Transportation Research Part C: Emerging Technologies 67: 79–90.21

6 Appendix

6.1 Python Code

```
import random
import matplotlib.pyplot as plt
import networkx as nx
random.seed(520)

# Define valid points and their neighbors based on the diagram
nodes = {
    (0, 1): [(1, 1)],          # Neighbors of point (0, 1)
    (0, 2): [(1, 2)],          # Neighbors of point (0, 2)
    (0, 3): [(1, 3)],          # Neighbors of point (0, 3)
    (1, 0): [(1, 1)],          # Neighbors of point (1, 1)
    (1, 1): [(0, 1), (2, 1), (1, 0), (1, 2)],          # Neighbors of point (1, 1)
    (1, 2): [(1, 1), (1, 3), (0, 2), (2, 2)],
    (1, 3): [(0, 3), (2, 3), (1, 2), (1, 4)],          # Neighbors of point (1, 3)
    (2, 1): [(1, 1), (2, 2)],          # Neighbors of point (2, 1)
    (2, 2): [(2, 1), (2, 3), (1, 2), (3, 2)],
    (2, 3): [(1, 3), (2, 2), (3, 3), (2, 4)],          # Neighbors of point (2, 3)
    (2, 4): [(2, 3)],
    (3, 2): [(2, 2), (3, 3)],  # Neighbors of point (3, 2)
    (3, 3): [(3, 2), (2, 3)],          # Neighbors of point (4, 3)
}

# Define boundary points for E and P
boundary_E = [(0, 1), (1, 0), (2, 1), (3, 2), (3, 3), (2, 4)]
boundary_P = [(0, 3), (0, 2), (1, 4)]

# Function to simulate a single random walk
def random_walk(start, boundary_E, boundary_P, nodes):
    current = start
```

```
    while True:
        # Check if the current position is in a boundary
        if current in boundary_E:
            return 1  # Reached E
        if current in boundary_P:
            return 0  # Reached P

        # Get valid moves from the current position
        valid_moves = nodes.get(current, [])
        if not valid_moves:
            raise RuntimeError("No valid moves available. Check the graph structure.")

        # Randomly choose a valid move
        current = random.choice(valid_moves)

# Function to simulate probabilities for all non-boundary points
def simulate_probability(point, boundary_E, boundary_P, nodes, n_sim=1000):
    results = [random_walk(point, boundary_E, boundary_P, nodes) for _ in range(n_sim)]
    return sum(results) / n_sim  # Probability of reaching E before P

# Simulate probabilities for all internal points
internal_points = [point for point in nodes if point not in boundary_E + boundary_P]
probabilities = {}
for point in internal_points:
    probabilities[point] = simulate_probability(point, boundary_E, boundary_P, nodes,
    n_sim=5000)

# Print the results
print("Probabilities of reaching E before P for internal points:")
for point, prob in probabilities.items():
    print(f"Point {point}: {prob:.4f}")
```

```
# Visualization using networkx
G = nx.Graph()

# Add nodes to the graph
for node in nodes:
    G.add_node(node)

# Add edges between valid neighbors
for node, neighbors in nodes.items():
    for neighbor in neighbors:
        G.add_edge(node, neighbor)

# Flip the graph horizontally and rotate clockwise by 90 degrees
def transform_layout(pos):
    # Flip horizontally: Negate x-coordinates
    flipped_pos = {node: (-x, y) for node, (x, y) in pos.items()}
    # Rotate 90 degrees clockwise: Swap x and y, and negate the new y
    rotated_pos = {node: (y, -x) for node, (x, y) in flipped_pos.items()}
    return rotated_pos

# Define original positions based on node coordinates
original_pos = {node: node for node in G.nodes}

# Apply transformations
pos = transform_layout(original_pos)

# Assign node colors based on probabilities
node_colors = []
for node in G.nodes:
    if node in probabilities:  # Internal nodes
        node_colors.append(probabilities[node])  # Probability as the color (0 to 1)
    elif node in boundary_E:  # E boundary
```

```
        node_colors.append(1.0)  # Maximum value for blue in colormap
    elif node in boundary_P:  # P boundary
        node_colors.append(0.0)  # Minimum value for red in colormap
    else:
        node_colors.append(0.5)  # Default value for undefined nodes

# Visualization
plt.figure(figsize=(8, 6))

# Draw nodes with colors based on probabilities
nx.draw(
    G, pos, with_labels=False, node_color=node_colors, cmap=plt.cm.coolwarm,
    node_size=800, vmin=0, vmax=1, font_size=10, font_color="black"
)

# Add probability labels for internal nodes
labels = {}
for node in G.nodes:
    if node in probabilities:  # Internal nodes
        labels[node] = f"{probabilities[node]:.4f}"
        # Format probabilities to 4 decimal places
    elif node in boundary_E:
        labels[node] = "1"  # Label boundary E
    elif node in boundary_P:
        labels[node] = "0"  # Label boundary P

nx.draw_networkx_labels(G, pos, labels=labels, font_size=10, font_color="black")

# Add title
plt.title("Probabilities of Reaching E Before P")
plt.show()
```

```
import random
import matplotlib.pyplot as plt
import networkx as nx
random.seed(520)

# Define valid points and their neighbors based on the diagram
A = (0.809, 1.539)
B = (0, 0.951)
C = (0.309, 0)
D = (1.309, 0)
E = (1.618, 0.951)
Pa = (0.809, 2.539)
Pb = (-1, 0.951)
Pc = (0, -0.951)
Pd = (1.618, -0.951)
Pe = (2.618, 0.951)
nodes = {
    A: [Pa, B, E],
    B: [A, C, Pb],
    C: [B, D, Pc],
    D: [C, E, Pd],
    E: [D, A, Pe]
}

# Define boundary points for E and P
boundary_E = [D]
boundary_P = [Pa, Pb, Pc, Pd, Pe]

# Function to simulate a single random walk
def random_walk(start, boundary_E, boundary_P, nodes):
    current = start
    while True:
```

```
        # Check if the current position is in a boundary
        if current in boundary_E:
            return 1  # Reached E
        if current in boundary_P:
            return 0  # Reached P

        # Get valid moves from the current position
        valid_moves = nodes.get(current, [])
        if not valid_moves:
            raise RuntimeError("No valid moves available. Check the graph structure.")

        # Randomly choose a valid move
        current = random.choice(valid_moves)

# Function to simulate probabilities for all non-boundary points
def simulate_probability(point, boundary_E, boundary_P, nodes, n_sim=1000):
    results = [random_walk(point, boundary_E, boundary_P, nodes) for _ in range(n_sim)]
    return sum(results) / n_sim  # Probability of reaching E before P

# Simulate probabilities for all internal points
internal_points = [point for point in nodes if point not in boundary_E + boundary_P]
probabilities = {}
for point in internal_points:
    probabilities[point] = simulate_probability(point, boundary_E, boundary_P, nodes
    , n_sim=5000)

# Print the results
print("Probabilities of reaching E before P for internal points:")
for point, prob in probabilities.items():
    print(f"Point {point}: {prob:.4f}")

# Visualization using networkx
```

```
G = nx.Graph()

# Add nodes to the graph
for node in nodes:
    G.add_node(node)

# Add edges between valid neighbors
for node, neighbors in nodes.items():
    for neighbor in neighbors:
        G.add_edge(node, neighbor)

# Assign node colors based on probabilities
node_colors = []
for node in G.nodes:
    if node in probabilities:  # Internal nodes
        node_colors.append(probabilities[node])  # Probability as the color (0 to 1)
    elif node in boundary_E:  # E boundary
        node_colors.append(1.0)  # Maximum value for blue in colormap
    elif node in boundary_P:  # P boundary
        node_colors.append(0.0)  # Minimum value for red in colormap
    else:
        node_colors.append(0.5)  # Default value for undefined nodes

# Draw the graph
plt.figure(figsize=(8, 6))
pos = {node: node for node in G.nodes}  # Use node coordinates as positions

# Draw nodes with colors based on probabilities
nx.draw(
    G, pos, with_labels=False, node_color=node_colors, cmap=plt.cm.coolwarm,
    node_size=800, vmin=0, vmax=1, font_size=10, font_color="black"
)
```

```
# Add probability labels for internal nodes
labels = {}
for node in G.nodes:
    if node in probabilities:  # Internal nodes
        labels[node] = f"{probabilities[node]:.4f}"
        # Format probabilities to 4 decimal places
    elif node in boundary_E:
        labels[node] = "1"  # Label boundary E
    elif node in boundary_P:
        labels[node] = "0"  # Label boundary P

nx.draw_networkx_labels(G, pos, labels=labels, font_size=10, font_color="black")

# Title for the plot
plt.savefig("random_walk_plot.png", dpi=300, bbox_inches="tight")
plt.show()
```

```
import random
import networkx as nx
import matplotlib.pyplot as plt

random.seed(520)

# Define the directed graph with edge weights as transition probabilities
edges = [
    ('a', 'd', 1/2),
    ('a', 'c', 1/2),
    ('c', 'a', 1/4),   # Transition from 'c' to 'a'
    ('d', 'a', 1/5),   # Transition from 'd' to 'a'
    ('c', 'b', 1/4),   # Transition from 'c' to 'b'
    ('c', 'd', 1/2),   # Transition from 'c' to 'd'
    ('d', 'c', 2/5),   # Transition from 'd' to 'c'
```

```
    ('d', 'b', 2/5),   # Transition from 'd' to 'b'
]

# Create a directed graph
G = nx.DiGraph()
G.add_weighted_edges_from(edges)

# Function to simulate a single random walk and track crossings
def random_walk_with_crossings(start, G, terminating_node, max_steps=100):
    current = start
    crossings = {}
    for _ in range(max_steps):
        # Terminate the walk if the current node is the absorbing node
        if current == terminating_node:
            break

        neighbors = list(G.neighbors(current))
        if not neighbors:
            break  # Stop if no neighbors to move to

        # Choose the next node based on transition probabilities
        weights = [G[current][neighbor]['weight'] for neighbor in neighbors]
        next_node = random.choices(neighbors, weights=weights, k=1)[0]

        # Track net crossings
        edge = (current, next_node)
        reverse_edge = (next_node, current)
        crossings[edge] = crossings.get(edge, 0) + 1
        crossings[reverse_edge] = crossings.get(reverse_edge, 0) - 1

        current = next_node
```

```
    return crossings

# Function to estimate net crossings for all edges
def estimate_net_crossings(start, G, terminating_node, n_sim=10000):
    net_crossings = {}
    for _ in range(n_sim):
        crossings = random_walk_with_crossings(start, G, terminating_node)
        for edge, count in crossings.items():
            if edge not in net_crossings:
                net_crossings[edge] = 0
            net_crossings[edge] += count

    # Average crossings over all simulations
    for edge in net_crossings:
        net_crossings[edge] /= n_sim

    return net_crossings

# Start the random walk from 'a' and terminate at 'b'
start_node = 'a'
terminating_node = 'b'
net_crossings = estimate_net_crossings(start_node, G, terminating_node
, n_sim=5000)

# Print the net crossings for each edge
print("Net crossings for each edge:")
for edge, value in net_crossings.items():
    print(f"Edge {edge}: {value:.4f}")

# Visualization of the graph with edge labels for net crossings
plt.figure(figsize=(8, 6))
pos = nx.spring_layout(G)
```

```
pos = {
    'a': (-1, 0),  # Left
    'b': (1, 0),   # Right
    'c': (0, 1),   # Top
    'd': (0, -1),  # Bottom
}

# Draw the graph
nx.draw(
    G, pos, with_labels=True, node_color="lightblue", node_size=1500, font_size=12,
    edge_color="black", arrowsize=20
)

# Add edge labels for net crossings
edge_labels = {edge: f"{net_crossings.get(edge, 0):.4f}" for edge in G.edges}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=10)

# Add title
plt.title("Net Crossings for Each Edge in the Random Walk (Terminating at b)")
plt.show()
```

```
import matplotlib.pyplot as plt
import networkx as nx

# Create a directed multigraph
G = nx.MultiDiGraph()

# Define nodes
nodes = ['a', 'b', 'c', 'd']

# Define bidirectional edges with weights
edges = [
```

```
    ('a', 'c', 1/2), ('c', 'a', 1/4),
    ('a', 'd', 1/2), ('d', 'a', 1/5),
    ('b', 'c', 1/3), ('c', 'b', 1/4),
    ('b', 'd', 2/3), ('d', 'b', 2/5),
    ('c', 'd', 1/2), ('d', 'c', 2/5)
]

# Add nodes and edges to the graph
G.add_nodes_from(nodes)
G.add_weighted_edges_from(edges)

# Define positions for the nodes
pos = {
    'a': (-1, 0),
    'b': (1, 0),
    'c': (0, 1),
    'd': (0, -1)
}

# Draw the graph with parallel edges
plt.figure(figsize=(8, 8))
nx.draw(
    G, pos, with_labels=True, node_color='lightblue', node_size=2000,
    font_size=15, font_weight='bold', arrowsize=20, connectionstyle="arc3,rad=0.2"
)

# Draw edge labels (weights)
edge_labels = {(u, v): f"{round(d['weight'], 2)}" for u, v, d in G.edges(data=True)}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=12, label_pos=0.3)

# Set the title and display the graph
plt.savefig("random_walk_plot.png", dpi=300, bbox_inches="tight")
```

```
plt.show()
```

6.2 R Code

```
# Data for interior points only
data <- data.frame(
  Node = c("A", "B", "C", "D", "E"), # Nodes
  Voltage = c(0.8230, 0.7865, 0.8764, 0.5056, 0.3230), # Voltage from Electric Network
  Probability = c(0.8168, 0.7962, 0.8722, 0.4994, 0.3238), # Probability from Random Walk
  Difference = abs(c(0.8230, 0.7865, 0.8764, 0.5056, 0.3230) -
                   c(0.8168, 0.7962, 0.8722, 0.4994, 0.3238)) # Absolute Difference
)

# Render the table in R Markdown
knitr::kable(data, col.names = c("Node", "Voltage Distribution (V)",
                                 "Probability Distribution",
                                 "Absolute Difference"),
             caption = "Comparison of Voltage and Probability Distributions")
```

```
# Data for interior points only
data <- data.frame(
  Node = c("A", "B", "C", "D"), # Nodes
  Voltage = c(0.200, 0.200, 0.400, 0.400), # Voltage from Electric Network
  Probability = c(0.2046, 0.2010, 0.4112, 0.4010), # Probability from Random Walk
  Difference = abs(c(0.200, 0.200, 0.400, 0.400) -
                   c(0.2046, 0.2010, 0.4112, 0.4010)) # Absolute Difference
)

# Render the table in R Markdown
knitr::kable(data, col.names = c("Node", "Voltage Distribution (V)",
                                 "Probability Distribution",
                                 "Absolute Difference"),
             caption = "Comparison of Voltage and Probability Distributions")
```

```r
data <- data.frame(
  Node = c("a->c", "a->d", "c->d", "c->b", "d->b"), # Edges
  Current = c(-0.4634, -0.5366, 0.1068, 0.3566, 0.6434), # Current from Electric Network
  Probability = c(-0.562, -0.625, 0.125, 0.4375, 0.750), # Net Crossing from Random Walk
  Ratio = c(-0.4634, -0.5366, 0.1068, 0.3566, 0.6434) /
                  c(-0.562, -0.625, 0.125, 0.4375, 0.750)) # ratio


# Render the table in R Markdown
knitr::kable(data, col.names = c("Edges", "Current Flow (A)",
                                 "Estimated Net Crossing", "Ratio"),
             caption = "Comparison of Current and Net Crossing")
```