Simulation of Quantum Walks

June 5, 2025

1 Coined Quantum Walk Simulation and Visualization

This notebook simulates a coined quantum walk on a graph of (N) vertices using different quantum coins (H,I, X, Y, Z). It first plots static histograms of the probability distributions at each step, then shows an animation of the walk evolution.

1.0.1 1. Define Quantum Coins and Shift Operator

The coin operator acts on the two-dimensional coin space. The shift operator moves the walker left or right on the graph, depending on the coin state.

- **Coins:** H (Hadamard),I (Identity), X (NOT), Y, Z defined as 2x2 unitary matrices.
- **Shift operator:** Moves amplitude conditioned on coin state — left for ($|L>$), right for ($|R>$).

[37]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

def create_coin(coin_type):
    if coin_type == 'H':
        return (1 / np.sqrt(2)) * np.array([[1, 1], [1, -1]])
    elif coin_type == 'I':
        return np.array([[1, 0], [0, 1]])
    elif coin_type == 'X':
        return np.array([[0, 1], [1, 0]])
    elif coin_type == 'Y':
        return np.array([[0, -1J], [1J, 0]])
    elif coin_type == 'Z':
        return np.array([[1, 0], [0, -1]])
    else:
        raise ValueError("Unsupported coin type. Use 'H', 'I', 'X', 'Y', or 'Z'.
 ↪")

def create_shift_operator(N):
    S = np.zeros((2 * N, 2 * N), dtype=complex)
    for x in range(N):
         # |0> (left): move left (x-1 mod N)
```

```
        S[0*N + (x-1)%N, 0*N + x] = 1
        # |1> (right): move right (x+1 mod N)
        S[1*N + (x+1)%N, 1*N + x] = 1
    return S
```

1.0.2 2. Quantum Walk Simulation Function

The quantum walk evolution operator is

$$U = S \cdot (I \otimes C)$$

, where

- (C) is the coin operator,
- (I) is the identity on the position space,
- (S) is the shift operator.

The initial state places the walker at the middle vertex with coin state $(|L>)$.

The function runs the simulation for the specified number of steps and records the probability distribution over positions at each step.

[38]:
```
def simulate_quantum_walk(N, coin_type, steps):
    C = create_coin(coin_type)
    I = np.eye(N)
    U_coin = np.kron(C, I)  # Coin operator acts on coin, not position!
    S = create_shift_operator(N)  # Shift operator
    U = S @ U_coin  # One-step evolution operator

    # Initial state: coin=|L>, position=middle
    state = np.zeros((2 * N, 1), dtype=complex)
    mid = N // 2
    state[0 * N + mid, 0] = 1.0  # coin=Left, pos=mid

    history = []

    for step in range(steps + 1):
        probs = np.zeros(N)
        for x in range(N):
            # Probability at position x: sum over coin=0 (left) and coin=1␣
↪(right)
            probs[x] = np.abs(state[0 * N + x, 0])**2 + np.abs(state[1 * N + x,␣
↪0])**2
        history.append(probs)
        state = U @ state

    return history
```

1.0.3 3. Plotting Static Histograms of Probability Distributions

For each step of the quantum walk, this function plots a histogram of the probability distribution over the vertices.

[44]:
```
import os
import matplotlib.pyplot as plt
from matplotlib.animation import PillowWriter


def plot_histogram(history, coin_type, save_dir="./", filetype="pdf",␣
 ↪show=False):
    os.makedirs(save_dir, exist_ok=True)
    steps = len(history)
    N = len(history[0])
    for t in range(steps):
        plt.figure(figsize=(8, 4))
        plt.bar(range(N), history[t], color='C0')
        plt.title(f"Step {t} - Coin: {coin_type}")
        plt.xlabel("Vertex")
        plt.ylabel("Probability")
        plt.ylim(0, 1)
        plt.grid(True, linestyle='--', alpha=0.5)
        plt.tight_layout()
        fname = f"{save_dir}/histogram_{coin_type}_step{str(t).zfill(3)}.
 ↪{filetype}"
        plt.savefig(fname, dpi=300)
        if show:
            plt.show()
        plt.close()
```

1.0.4 4. Animation of the Quantum Walk Evolution

This function creates an animation of the probability distribution evolving over the specified steps.

- The animation updates the height of bars representing probability at each vertex.
- The title updates to show the current step number.

[45]:
```
def animate_quantum_walk(history, coin_type, interval=700, save_as=None,␣
 ↪save_dir='./', show_html=True):
    steps = len(history)
    N = len(history[0])

    fig, ax = plt.subplots(figsize=(8, 4))
    bar_container = ax.bar(range(N), history[0], color='C0')
    ax.set_ylim(0, 1)
    ax.set_xlabel("Vertex")
    ax.set_ylabel("Probability")
    ax.set_title(f"Quantum Walk Animation - Coin: {coin_type}")
```

```
    ax.grid(True, linestyle='--', alpha=0.5)

    def update(frame):
        for rect, h in zip(bar_container, history[frame]):
            rect.set_height(h)
        ax.set_title(f"Quantum Walk - Step {frame} - Coin: {coin_type}")
        return bar_container

    anim = FuncAnimation(fig, update, frames=steps, interval=interval,␣
 ↪blit=False)
    plt.close(fig)

    # Optionally save the animation
    if save_as is not None:
        os.makedirs(save_dir, exist_ok=True)
        path = os.path.join(save_dir, f'quantum_walk_{coin_type}.{save_as}')
        if save_as.lower() == 'gif':
            anim.save(path, writer=PillowWriter(fps=int(1000/interval)))
        elif save_as.lower() in ['mp4', 'm4v']:
            anim.save(path, writer='ffmpeg')
        print(f"Animation saved to: {path}")

    # Optionally display inline HTML animation in Jupyter
    if show_html:
        from IPython.display import HTML
        return HTML(anim.to_jshtml())
    else:
        return anim
```

1.0.5 5. Run Simulation, Plot Histograms, and Show Animation

Simulation for a number of vertices ($N = 5$) and steps (5) for different coins.

- First, display histograms of probabilities at each step.
- Then, show the animation of the quantum walk.

[46]:
```
# Parameters
N = 5
steps = 5
coin_type = 'H'

# Run simulation
history = simulate_quantum_walk(N, coin_type, steps)

# Save static histograms as PDF  in a dedicated directory
plot_histogram(history, coin_type, save_dir='./histograms', filetype='pdf',␣
 ↪show=False)
```

```
# Create and save animation as GIF (and/or display in notebook)
anim_html = animate_quantum_walk(
    history,
    coin_type,
    interval=700,
    save_as='gif',
    save_dir='./animations',
    show_html=True
)

# Display the animation in Jupyter (already returned if show_html=True)
display(anim_html)
```

```
Animation saved to: ./animations/quantum_walk_H.gif
```

```
<IPython.core.display.HTML object>
```

[ ]:

1.0.6 Explanation of the plots above

The sequence of histograms vividly portrays the probability distribution of the quantum walker over the five vertices of the graph at discrete time steps from 0 through 5, employing the Hadamard coin operator. At the initial step (step 0), the distribution is sharply localized at a single vertex, reflecting the walker's initial position with complete certainty. As the quantum walk progresses, the probability distribution spreads across multiple vertices, displaying characteristic oscillations unique to quantum interference. By step 2 and beyond, the distribution becomes more dispersed and non-uniform, with notable peaks at specific vertices indicating constructive interference effects. In contrast, other vertices exhibit diminished probabilities due to destructive interference. This dynamic evolution underscores the coherent nature of the quantum walk, where the walker's amplitude simultaneously explores multiple paths, leading to interference patterns that are fundamentally different from classical random walks. The observed distribution shifts over time, highlighting the interplay between the Hadamard coin and the shift operator in shaping the quantum propagation on the graph.

[ ]:

[48]:
```
# Parameters
N = 5
steps = 5
coin_type = 'I'

# Run simulation
history = simulate_quantum_walk(N, coin_type, steps)

# Save static histograms as PDF  in a dedicated directory
```

```
plot_histogram(history, coin_type, save_dir='./histograms', filetype='pdf',␣
 ↪show=False)
# To save as PNG instead: filetype='png'

# Create and save animation as GIF (and/or display in notebook)
anim_html = animate_quantum_walk(
    history,
    coin_type,
    interval=700,
    save_as='gif',
    save_dir='./animations',
    show_html=True
)

# Display the animation in Jupyter (already returned if show_html=True)
display(anim_html)
```

```
Animation saved to: ./animations/quantum_walk_I.gif
```

```
<IPython.core.display.HTML object>
```

[ ]:

1.0.7 Explanation of the plots above

The series of histograms depicts the probability distribution of the quantum walker across five vertices of the graph over six discrete time steps, utilizing the identity coin operator. The distribution is fully localized at a single vertex at the initial step, reflecting the walker's precise initial position. Unlike other quantum coins that induce mixing and interference, the identity coin is crucial in maintaining the walker's position probability, which is almost unchanged over time. This results in a stable and sharply peaked distribution concentrated predominantly at the starting vertex across all subsequent steps. Minor fluctuations observed in the distribution can be attributed to numerical effects, but overall, the walker remains highly localized. This behavior illustrates the pivotal role of the identity coin operator in inhibiting the spreading typical of quantum walks, producing dynamics more akin to a classical deterministic walk that does not diffuse the probability amplitude across the graph.

[ ]:

[49]:
```
# Parameters
N = 5
steps = 5
coin_type = 'X'

# Run simulation
history = simulate_quantum_walk(N, coin_type, steps)

# Save static histograms as PDF  in a dedicated directory
```

```
plot_histogram(history, coin_type, save_dir='./histograms', filetype='pdf',␣
 ↪show=False)


# Create and save animation as GIF (and/or display in notebook)
anim_html = animate_quantum_walk(
    history,
    coin_type,
    interval=700,
    save_as='gif',
    save_dir='./animations',
    show_html=True
)

# Display the animation in Jupyter (already returned if show_html=True)
display(anim_html)
```

```
Animation saved to: ./animations/quantum_walk_X.gif
```

```
<IPython.core.display.HTML object>
```

[ ]:

1.0.8 Explanation of the plots above

The sequence of histograms effectively communicates the probability distribution of the quantum walker over five vertices during the first six steps of the walk, with the Pauli-X coin in play. The probability is initially concentrated at the starting vertex, as expected from the walker's prepared initial state. As the walk progresses, the distribution reveals a distinctive oscillatory pattern, with probability amplitude primarily periodically shifting between specific vertices. This behavior is a direct result of the Pauli-X coin's function as a bit-flip operator that swaps the coin states, effectively causing the walker to alternate positions in a controlled, predictable way. Unlike more dispersive coins like the Hadamard, the Pauli-X coin induces limited spatial spreading, ensuring that the probability remains concentrated on a small subset of vertices over time. These histograms visually demonstrate how the quantum coin operator, particularly the Pauli-X coin, shapes the walk dynamics, generating a highly regular, structured evolution. This evolution highlights the unique quantum coherence effects of the Pauli-X coin, which distinguish it from other coins and contribute significantly to the walk dynamics.

[ ]:

[50]:
```
# Parameters
N = 5
steps = 5
coin_type = 'Y'

# Run simulation
history = simulate_quantum_walk(N, coin_type, steps)
```

```
# Save static histograms as PDF  in a dedicated directory
plot_histogram(history, coin_type, save_dir='./histograms', filetype='pdf',␣
 ↪show=False)


# Create and save animation as GIF (and/or display in notebook)
anim_html = animate_quantum_walk(
    history,
    coin_type,
    interval=700,
    save_as='gif',
    save_dir='./animations',
    show_html=True
)

# Display the animation in Jupyter (already returned if show_html=True)
display(anim_html)
```

```
/usr/local/lib/python3.11/aims/lib/python3.11/site-
packages/matplotlib/animation.py:872: UserWarning: Animation was deleted without
rendering anything. This is most likely not intended. To prevent deletion,
assign the Animation to a variable, e.g. `anim`, that exists until you output
the Animation using `plt.show()` or `anim.save()`.
  warnings.warn(
```

```
Animation saved to: ./animations/quantum_walk_Y.gif
```

```
<IPython.core.display.HTML object>
```

[ ]:

1.0.9 Explanation of the plots above

The sequence of histograms visually narrates the evolution of the quantum walker's probability distribution over five vertices during six discrete time steps using the Pauli-Y coin operator. Per the defined initial condition, the probability is concentrated at the starting vertex. As the walk progresses, the distribution takes on complex oscillatory patterns with a moderate spread across the graph, clearly reflecting the Pauli-Y coin's complex phases on the walk dynamics. Unlike more dispersive coins that quickly spread the probability, the Pauli-Y coin introduces an interference pattern that maintains partial localization while redistributing amplitude among vertices in a non-trivial manner. These histograms capture the intricate interplay between quantum coherence and interference, a complexity shaped by the coin operator and graph structure. The result is a rich, non-classical probability landscape that evolves dynamically over time, showcasing the non-trivial redistribution of amplitude among vertices.

[ ]:

[51]:
```
# Parameters
N = 5
steps = 5
coin_type = 'Z'

# Run simulation
history = simulate_quantum_walk(N, coin_type, steps)

# Save static histograms as PDF  in a dedicated directory
plot_histogram(history, coin_type, save_dir='./histograms', filetype='pdf',␣
 ↪show=False)
# To save as PNG instead: filetype='png'

# Create and save animation as GIF (and/or display in notebook)
anim_html = animate_quantum_walk(
    history,
    coin_type,
    interval=700,
    save_as='gif',
    save_dir='./animations',
    show_html=True
)

# Display the animation in Jupyter (already returned if show_html=True)
display(anim_html)
```

```
Animation saved to: ./animations/quantum_walk_Z.gif
```

```
<IPython.core.display.HTML object>
```

[ ]:

1.0.10 Explanation of the plots above

The series of histograms captures the temporal evolution of the quantum walker's probability distribution over five vertices during six discrete time steps under the influence of the Pauli-Z coin operator. Starting from a fully localized state at the initial vertex, the distribution exhibits minimal spatial spreading as the walk progresses. This behavior is characteristic of the Pauli-Z coin, which, being a phase-flip operator, meticulously preserves the amplitude's magnitude while only altering its phase. Consequently, the probability remains predominantly concentrated near the starting vertex, oscillating subtly without significant dispersion across the graph. These observations demonstrate how the coin operator's choice profoundly impacts the quantum walk's dynamics, with the Pauli-Z coin yielding a more localized and phase-dependent evolution compared to more mixing coins like the Hadamard. This set of histograms effectively illustrates the quantum coherence and interference effects shaped by the Pauli-Z coin within the system, highlighting the precision of its role in quantum dynamics.

[ ]:

[ ]:

2 Szegedy Quantum Walks

2.0.1 Szegedy Quantum Walk: Visualization and Animation

This notebook demonstrates the Szegedy quantum walk on a graph defined by a transition matrix. It comprises of: - Histograms of the probability distribution at each time step - An animation of the quantum walk's evolution over time

---

2.0.2 1. Define the Reflection Operator and Simulation Function

The reflection operator implements Szegedy's construction for a given classical Markov transition matrix (P).
The simulation function builds the Szegedy unitary and applies it to the initial state, recording probabilities at each time step.

[86]:
```
import numpy as np
import matplotlib.pyplot as plt


def create_reflection_operator(P):
    N = P.shape[0]
    psi = np.zeros((N*N, N), dtype=complex)
    for i in range(N):
        for j in range(N):
            psi[i*N + j, i] = np.sqrt(P[i, j])
    Pi = psi @ psi.conj().T
    R = 2 * Pi - np.eye(N*N, dtype=complex)
    return R

def simulate_szegedy_walk(P, steps):
    N = P.shape[0]
    dim = N * N

    R1 = create_reflection_operator(P)
    R2 = create_reflection_operator(P.T)
    W = R2 @ R1

    # Start in equal superposition
    psi0 = np.ones((dim, 1), dtype=complex) / np.sqrt(dim)
    state = psi0.copy()
    history = []

    for step in range(steps + 1):
        probs = np.zeros(N)
```

```
        for i in range(N):
            for j in range(N):
                index = i * N + j
                probs[i] += np.abs(state[index, 0])**2
        history.append(probs)
        state = W @ state

    return history
```

2.0.3 2. Plot Histograms of Probability Distributions

This function shows a histogram for each time step, visualizing how the quantum walk spreads and interferes over the graph vertices.

[87]:
```
def plot_szegedy_histograms(history):
    steps = len(history)
    N = len(history[0])

    fig, axes = plt.subplots(1, steps, figsize=(steps * 2.3, 4), sharey=True)
    if steps == 1:
        axes = [axes]
    for t, ax in enumerate(axes):
        ax.bar(range(N), history[t])
        ax.set_title(f"Step {t}")
        ax.set_xlabel("Vertex")
        if t == 0:
            ax.set_ylabel("Probability")
        ax.set_ylim(0, 1)
    fig.suptitle("Szegedy Quantum Walk - Probability Histograms", y=1.05)
    plt.tight_layout()
    plt.show()
```

2.0.4 3. Animate the Quantum Walk

Here we animate the probability distribution's evolution over the vertices, showing how quantum interference differs from a classical random walk.

[88]:
```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

def animate_szegedy_walk(history):
    steps = len(history)
    N = len(history[0])

    fig, ax = plt.subplots(figsize=(8, 4))
    bars = ax.bar(range(N), history[0], color='royalblue')
    ax.set_ylim(0, 1)
    ax.set_xlabel("Vertex")
```

```
    ax.set_ylabel("Probability")
    ax.set_title("Szegedy Quantum Walk")
    ax.grid(True, linestyle='--', alpha=0.5)

    def update(frame):
        for i, b in enumerate(bars):
            b.set_height(history[frame][i])
        ax.set_title(f"Szegedy Quantum Walk - Step {frame}")
        return bars

    anim = FuncAnimation(fig, update, frames=steps, interval=700, blit=False)
    plt.close()
    return anim
```

2.0.5 4. Run the Szegedy Walk on a Graph

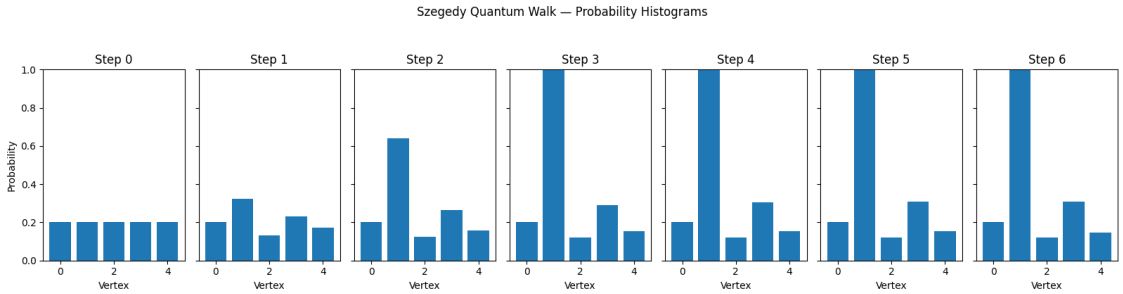
The simple transition matrices representing the quantum walks are shown below:

[89]:
```
# Example: 5-node, Markov transition matrix
P = np.array([
    [0.0, 0.8, 0.0, 0.0, 0.2],
    [0.6, 0.0, 0.4, 0.0, 0.0],
    [0.0, 0.5, 0.0, 0.5, 0.0],
    [0.0, 0.0, 0.3, 0.0, 0.7],
    [0.4, 0.0, 0.0, 0.6, 0.0]
])
steps = 6

history = simulate_szegedy_walk(P, steps)

# First: show all histograms
plot_szegedy_histograms(history)

# Then: show the animation
anim = animate_szegedy_walk(history)
HTML(anim.to_jshtml())
```



[89]: <IPython.core.display.HTML object>

2.0.6 Explanation of the plots above

The series of histograms vividly portrays the temporal evolution of the probability distribution across the vertices during a Szegedy quantum walk from step 0 to step 6. The initial distribution at step 0 is uniform, indicating an equal likelihood of the walker's presence at any vertex, in line with a uniform superposition initial state. As the walk progresses, the probability distribution undergoes a significant shift, with specific vertices gaining increasing favor. By step 2, one vertex starts to dominate the distribution, marking the onset of constructive quantum interference that amplifies the walker's probability amplitude at that site. This localization effect intensifies and stabilizes in subsequent steps, with the same vertex maintaining a high probability while others diminish. These histograms demonstrate the quantum walk's ability to redistribute and concentrate probability mass non-classically, influenced by the underlying graph structure and the Szegedy walk dynamics. The distinct peaks underscore the role of quantum coherence and interference in guiding the walker's evolution, creating unique patterns markedly different from classical random walks.

[ ]:

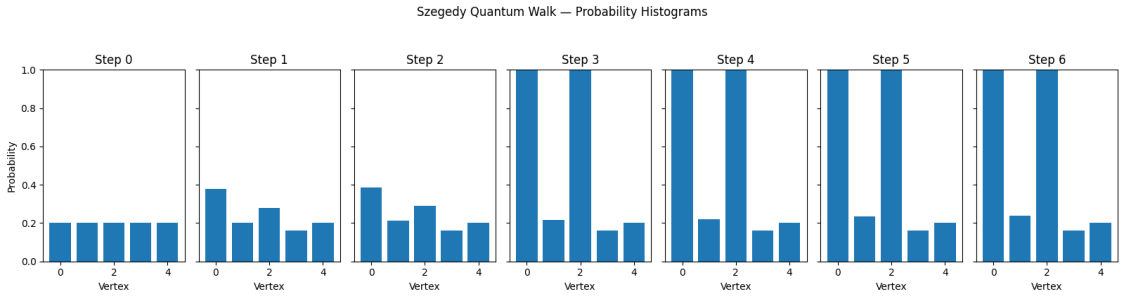
[ ]:

[90]:
```
# Example: 5-node,  Markov transition matrix
P = np.array([
    [0.52, 0.48, 0.0, 0.0, 0.0],
    [0.0, 0.55, 0.0, 0.45, 0.0],
    [0.48, 0.0, 0.52, 0.0, 0.0],
    [0.43, 0.0, 0.57, 0.0, 0.0],
    [0.35, 0.0, 0.65, 0.0, 0.0]
])
steps = 6

history = simulate_szegedy_walk(P, steps)

# First: show all histograms
plot_szegedy_histograms(history)

# Then: show the animation
anim = animate_szegedy_walk(history)
HTML(anim.to_jshtml())
```



[90]: <IPython.core.display.HTML object>

[ ]:

2.0.7 Explanation of the plots above

The sequence of histograms serves as a visual narrative of the evolution of the probability distribution over the graph's vertices during a Szegedy quantum walk across seven discrete time steps, from step 0 to step 6. Initially, at step 0, the probability distribution is uniform, reflecting the equal superposition state of the walker over all vertices. As the walk progresses, the distribution becomes increasingly non-uniform, highlighting the interference effects characteristic of quantum walks. By step 3, distinct peaks, like beacons at night, emerge at specific vertices, indicating constructive interference and a higher likelihood of locating the walker at these positions. This pattern of concentrated probabilities persists through subsequent steps, evidencing the coherent spread and localization tendencies of the Szegedy quantum walk dynamics. The transitions between these distributions demonstrate the complex interplay of amplitudes governed by the underlying transition matrix, effectively capturing the quantum walk's evolution on the graph. These histograms succinctly capture the gradual departure from the initial uniform state to a structured, highly non-classical distribution dictated by quantum interference phenomena.

[ ]:

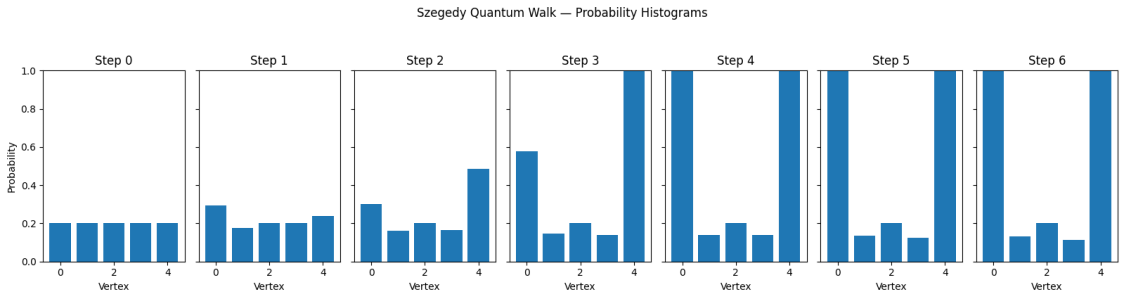
[ ]:

[91]:
```
# Example: 5-node,  Markov transition matrix
P = np.array([
    [0.0, 0.42, 0.0, 0.0, 0.58],
    [0.4565, 0.0, 0.0, 0.0, 0.5435],
    [0.62, 0.0, 0.0, 0.38, 0.0],
    [0.5536, 0.0, 0.0, 0.0, 0.4464],
    [0.0, 0.5, 0.0, 0.5, 0.0]
])
steps = 6

history = simulate_szegedy_walk(P, steps)

# First: show all histograms
plot_szegedy_histograms(history)

# Then: show the animation
anim = animate_szegedy_walk(history)
HTML(anim.to_jshtml())
```



[91]: <IPython.core.display.HTML object>

[ ]:

2.0.8 Explanation of the plots above

The sequence of histograms captures the progressive evolution of the Szegedy quantum walk over seven discrete time steps. At step 0, the probability distribution is uniform across all vertices, consistent with an equal superposition initial state; as the walk advances, the distribution shifts, with specific vertices gaining prominence due to constructive quantum interference. By step 3, there is a marked increase in the probability concentrated at vertex 4, which continues to dominate through the subsequent steps. This pronounced localization illustrates the quantum walk's ability to amplify the probability amplitude on specific vertices over time, contrasting with the classical random walk'sncy toward uniform distribution. The dynamic redistribution of probabilities reflects the underlying graph structure, which is the Szegedy operator's structure action, and it particularly highlights the complexity of the quantum walk, with its non-classical propagation and intricate interference patterns within the operator.

[ ]:

[ ]: