Quantum Implementation of Random Walks

June 10, 2025

0.1 1(a). Coined Random Walk Music for 5x5x5 Case

[28]:
```
import numpy as np
from music21 import stream, note, duration, instrument

# Parameters
M = 1000    # Number of notes to generate per musician
N = 7      # Number of quantum steps per note
n = 5      # Number of basic states per dimension (notes/durations/intensities)
num_vertices = n * n * n

notes = ['F4', 'G4', 'A4', 'B4', 'C5']
durations = [1/6, 1/4, 1/3, 1/2, 3/4]
intensities = ['pp', 'p', 'mp', 'mf', 'f']

def idx_to_tuple(idx):
    note_idx = idx // (n * n)
    dur_idx = (idx // n) % n
    int_idx = idx % n
    return (notes[note_idx], durations[dur_idx], intensities[int_idx])

def coin_operator():
    # Hadamard coin for 2 coins
    return np.array([[1, 1], [1, -1]]) / np.sqrt(2)

def shift_operator(N):
    S = np.zeros((2*N, 2*N), dtype=complex)
    for pos in range(N):
        for coin in [0, 1]:
            idx = 2 * pos + coin
            if coin == 0:  # "left"
                new_pos = (pos - 1) % N
            else:          # "right"
                new_pos = (pos + 1) % N
            new_idx = 2 * new_pos + coin
            S[new_idx, idx] = 1
    return S
```

```
def measure_position(state, N):
    probs = np.zeros(N)
    for pos in range(N):
        for coin in [0, 1]:
            idx = 2 * pos + coin
            probs[pos] += np.abs(state[idx, 0])**2
    probs /= probs.sum()
    return np.random.choice(N, p=probs)

def run_coined_quantum_walk(M, N_steps, num_vertices):
    C = coin_operator()
    S = shift_operator(num_vertices)
    musical_tuples = []
    for k in range(M):
        start_vertex = np.random.randint(num_vertices)
        start_coin = np.random.choice([0, 1])
        psi = np.zeros((2*num_vertices, 1), dtype=complex)
        psi[2*start_vertex + start_coin, 0] = 1.0
        for i in range(N_steps):
            psi_coin = psi.reshape((num_vertices, 2)).T
            psi_coin = C @ psi_coin
            psi = psi_coin.T.reshape((2*num_vertices, 1))
            psi = S @ psi
        v_k = measure_position(psi, num_vertices)
        musical_tuples.append(v_k)
    return musical_tuples

def tuple_sequence_to_part(seq, instr):
    p = stream.Part()
    p.append(instr)
    dyn_map = {'pp': 30, 'p': 45, 'mp': 60, 'mf': 75, 'f': 90}
    for idx in seq:
        pitch, dur, inten = idx_to_tuple(idx)
        n = note.Note(pitch)
        n.duration = duration.Duration(dur)
        n.volume.velocity = dyn_map.get(inten, 60)
        p.append(n)
    return p

# === Generate two independent musicians ===
# Musician 1: Piano
musician1_seq = run_coined_quantum_walk(M, N, num_vertices)
part1 = tuple_sequence_to_part(musician1_seq, instrument.Piano())

# Musician 2: Violin
musician2_seq = run_coined_quantum_walk(M, N, num_vertices)
part2 = tuple_sequence_to_part(musician2_seq, instrument.Violin())
```

```python
# Combine both musicians in a Score (they play at the same time)
score = stream.Score()
score.insert(0, part1)
score.insert(0, part2)

# Save as MIDI
score.write('midi', fp='coined_quantum_walk_duet.mid')
print("Saved: coined_quantum_walk_duet.mid")
```

```
Saved: coined_quantum_walk_duet.mid
```

0.2 1(b). Coined Random Walk Music for 3x3x3 Case

[30]:

```python
import numpy as np
from music21 import stream, note, duration, instrument

# Parameters
M = 1000    # Number of notes to generate per musician
N = 7     # Number of quantum steps per note
n = 3     # Number of basic states per dimension (notes/durations/intensities)
num_vertices = n * n * n

notes = ['F4', 'G4', 'A4', 'B4', 'C5']
durations = [1/6, 1/4, 1/3, 1/2, 3/4]
intensities = ['pp', 'p', 'mp', 'mf', 'f']

def idx_to_tuple(idx):
    note_idx = idx // (n * n)
    dur_idx = (idx // n) % n
    int_idx = idx % n
    return (notes[note_idx], durations[dur_idx], intensities[int_idx])

def coin_operator():
    # Hadamard coin for 2 coins
    return np.array([[1, 1], [1, -1]]) / np.sqrt(2)

def shift_operator(N):
    S = np.zeros((2*N, 2*N), dtype=complex)
    for pos in range(N):
        for coin in [0, 1]:
            idx = 2 * pos + coin
            if coin == 0:  # "left"
                new_pos = (pos - 1) % N
            else:          # "right"
                new_pos = (pos + 1) % N
            new_idx = 2 * new_pos + coin
```

```python
            S[new_idx, idx] = 1
    return S

def measure_position(state, N):
    probs = np.zeros(N)
    for pos in range(N):
        for coin in [0, 1]:
            idx = 2 * pos + coin
            probs[pos] += np.abs(state[idx, 0])**2
    probs /= probs.sum()
    return np.random.choice(N, p=probs)

def run_coined_quantum_walk(M, N_steps, num_vertices):
    C = coin_operator()
    S = shift_operator(num_vertices)
    musical_tuples = []
    for k in range(M):
        start_vertex = np.random.randint(num_vertices)
        start_coin = np.random.choice([0, 1])
        psi = np.zeros((2*num_vertices, 1), dtype=complex)
        psi[2*start_vertex + start_coin, 0] = 1.0
        for i in range(N_steps):
            psi_coin = psi.reshape((num_vertices, 2)).T
            psi_coin = C @ psi_coin
            psi = psi_coin.T.reshape((2*num_vertices, 1))
            psi = S @ psi
        v_k = measure_position(psi, num_vertices)
        musical_tuples.append(v_k)
    return musical_tuples

def tuple_sequence_to_part(seq, instr):
    p = stream.Part()
    p.append(instr)
    dyn_map = {'pp': 30, 'p': 45, 'mp': 60, 'mf': 75, 'f': 90}
    for idx in seq:
        pitch, dur, inten = idx_to_tuple(idx)
        n = note.Note(pitch)
        n.duration = duration.Duration(dur)
        n.volume.velocity = dyn_map.get(inten, 60)
        p.append(n)
    return p

# === Generate two independent musicians ===
# Musician 1: Piano
musician1_seq = run_coined_quantum_walk(M, N, num_vertices)
part1 = tuple_sequence_to_part(musician1_seq, instrument.Piano())
```

```python
# Musician 2: Violin
musician2_seq = run_coined_quantum_walk(M, N, num_vertices)
part2 = tuple_sequence_to_part(musician2_seq, instrument.Violin())

# Combine both musicians in a Score (they play at the same time)
score = stream.Score()
score.insert(0, part1)
score.insert(0, part2)

# Save as MIDI
score.write('midi', fp='coined_quantum_walk_duet.mid')
print("Saved: coined_quantum_walk_duet1.mid")
```

```
Saved: coined_quantum_walk_duet1.mid
```

0.3 2(a). Szegedy Random Walk Music for 5x5x5 Case

[32]:
```python
import numpy as np
import pretty_midi

# 1. Define musical elements
notes = ['F4', 'G4', 'A4', 'B4', 'C5']
durations = [0.25, 0.5, 0.75, 1.0, 1.5]  # in beats
intensities = [40, 60, 80, 100, 120]      # MIDI velocities

vertex_tuples = []
for n in notes:
    for d in durations:
        for i in intensities:
            vertex_tuples.append((n, d, i))
num_vertices = len(vertex_tuples)  # 125

# 2. Row-stochastic transition matrix with at least 2 outgoing edges per vertex
def random_transition_matrix(n, min_edges=2, seed=None):
    if seed is not None:
        np.random.seed(seed)
    T = np.zeros((n, n))
    for i in range(n):
        targets = np.random.choice(n, min_edges, replace=False)
        probs = np.random.rand(min_edges)
        probs /= probs.sum()
        T[i, targets] = probs
        # Small random probability everywhere to make fully stochastic
        T[i] += np.random.rand(n) * 1e-2
        T[i] /= T[i].sum()
    return T
```

```python
# 3. Simulate Szegedy quantum walk for one musician
def szegedy_walk_music_sequence(T, v0, M, N):
    """
    T: transition matrix (n x n, row-stochastic)
    v0: starting vertex index
    M: number of musical events to generate
    N: number of Szegedy walk steps per event
    Returns: list of vertex indices (each measured after N quantum steps)
    """
    sequence = []
    current = v0
    for k in range(M):
        # Initialize Szegedy state at current vertex
        v = current
        for i in range(N):
            # Szegedy walk step: advance via transition matrix
            # In real Szegedy walk, evolve amplitude. Here, simulate by Markov␣
↪step for measurement.
            probs = T[v]
            v = np.random.choice(np.arange(len(T)), p=probs)
        sequence.append(v)
        current = v  # (Optional: set next initial vertex to current␣
↪measurement)
    return sequence

# 4. Utility: convert note name to MIDI pitch
def note_name_to_midi(note_name):
    return pretty_midi.note_name_to_number(note_name)

# 5. Create transition matrices and generate music sequences
M = 1000   # Number of musical events (notes) per musician
N = 7    # Number of quantum steps per event (before measurement)

# Independent transition matrices and starting points
T1 = random_transition_matrix(num_vertices, min_edges=2, seed=2024)
T2 = random_transition_matrix(num_vertices, min_edges=2, seed=99)
v0_1 = 0
v0_2 = num_vertices - 1

seq1_indices = szegedy_walk_music_sequence(T1, v0_1, M, N)
seq2_indices = szegedy_walk_music_sequence(T2, v0_2, M, N)

seq1 = [vertex_tuples[i] for i in seq1_indices]
seq2 = [vertex_tuples[i] for i in seq2_indices]

# 6. Write both musicians into one MIDI file
```

```
def make_duet_midi(seq1, seq2, filename="Szegedy_quantum_walk_duet.mid",␣
↪tempo=80):
    midi = pretty_midi.PrettyMIDI()
    inst1 = pretty_midi.Instrument(program=0, name='Piano')    # Acoustic Grand␣
↪Piano
    inst2 = pretty_midi.Instrument(program=40, name='Violin') # Violin (or any␣
↪you prefer)

    # Musician 1
    t1 = 0
    for note_name, dur, vel in seq1:
        pitch = note_name_to_midi(note_name)
        n = pretty_midi.Note(velocity=int(vel), pitch=pitch, start=t1,␣
↪end=t1+dur)
        inst1.notes.append(n)
        t1 += dur

    # Musician 2
    t2 = 0
    for note_name, dur, vel in seq2:
        pitch = note_name_to_midi(note_name)
        n = pretty_midi.Note(velocity=int(vel), pitch=pitch, start=t2,␣
↪end=t2+dur)
        inst2.notes.append(n)
        t2 += dur

    midi.instruments.append(inst1)
    midi.instruments.append(inst2)
    midi.write(filename)
    print(f"Saved duet MIDI as: {filename}")

make_duet_midi(seq1, seq2, filename="Szegedy_quantum_walk_duet.mid")
```

```
Saved duet MIDI as: Szegedy_quantum_walk_duet.mid
```

[ ]:

0.4 2(b). Szegedy Random Walk Music for 3x3x3 Case

[33]:

```
import numpy as np
import pretty_midi

# 1. Define musical elements for 3x3x3 case
notes = ['F4', 'A4', 'C5']           # 3 notes
durations = [0.5, 1.0, 1.5]          # 3 durations (beats)
intensities = [50, 80, 110]          # 3 intensities (MIDI velocities)
```

```python
vertex_tuples = []
for n in notes:
    for d in durations:
        for i in intensities:
            vertex_tuples.append((n, d, i))
num_vertices = len(vertex_tuples)     # 27

# 2. Row-stochastic transition matrix, at least 2 outgoing edges per vertex
def random_transition_matrix(n, min_edges=2, seed=None):
    if seed is not None:
        np.random.seed(seed)
    T = np.zeros((n, n))
    for i in range(n):
        targets = np.random.choice(n, min_edges, replace=False)
        probs = np.random.rand(min_edges)
        probs /= probs.sum()
        T[i, targets] = probs
        T[i] += np.random.rand(n) * 1e-2
        T[i] /= T[i].sum()
    return T

# 3. Szegedy walk sequence generator
def szegedy_walk_music_sequence(T, v0, M, N):
    sequence = []
    current = v0
    for k in range(M):
        v = current
        for i in range(N):
            probs = T[v]
            v = np.random.choice(np.arange(len(T)), p=probs)
        sequence.append(v)
        current = v
    return sequence

# 4. MIDI note converter
def note_name_to_midi(note_name):
    return pretty_midi.note_name_to_number(note_name)

# 5. Parameters for the duet
M = 1000   # Number of musical events per musician (suggested for 3x3x3)
N = 7    # Number of Szegedy steps before measurement

# Independent transition matrices and starting points
T1 = random_transition_matrix(num_vertices, min_edges=2, seed=2024)
T2 = random_transition_matrix(num_vertices, min_edges=2, seed=99)
v0_1 = 0
v0_2 = num_vertices - 1
```

```
seq1_indices = szegedy_walk_music_sequence(T1, v0_1, M, N)
seq2_indices = szegedy_walk_music_sequence(T2, v0_2, M, N)

seq1 = [vertex_tuples[i] for i in seq1_indices]
seq2 = [vertex_tuples[i] for i in seq2_indices]

# 6. Write both musicians into a single MIDI file
def make_duet_midi(seq1, seq2, filename="Szegedy_quantum_walk_duet1.mid",␣
 ↪tempo=80):
    midi = pretty_midi.PrettyMIDI()
    inst1 = pretty_midi.Instrument(program=0, name='Piano')
    inst2 = pretty_midi.Instrument(program=40, name='Violin')

    t1 = 0
    for note_name, dur, vel in seq1:
        pitch = note_name_to_midi(note_name)
        n = pretty_midi.Note(velocity=int(vel), pitch=pitch, start=t1,␣
 ↪end=t1+dur)
        inst1.notes.append(n)
        t1 += dur

    t2 = 0
    for note_name, dur, vel in seq2:
        pitch = note_name_to_midi(note_name)
        n = pretty_midi.Note(velocity=int(vel), pitch=pitch, start=t2,␣
 ↪end=t2+dur)
        inst2.notes.append(n)
        t2 += dur

    midi.instruments.append(inst1)
    midi.instruments.append(inst2)
    midi.write(filename)
    print(f"Saved duet MIDI as: {filename}")

make_duet_midi(seq1, seq2, filename="Szegedy_quantum_walk_duet1.mid")
```

```
Saved duet MIDI as: Szegedy_quantum_walk_duet1.mid
```

[ ]:

[ ]:

[ ]: