Optimizing Quantum Network Performance: Performance evaluation of Quantum Dijkstra’s Algorithm

Nada Ikken

LPHE-MS

جامعة محمد الخامس بالرباط
Université Mohamed V de Rabat

ASP2024 alumni online presentation

15th October 2024

Plan

Introduction to Quantum Dijkstra Algorithm

1
- Importance of finding the shortest path in graphs

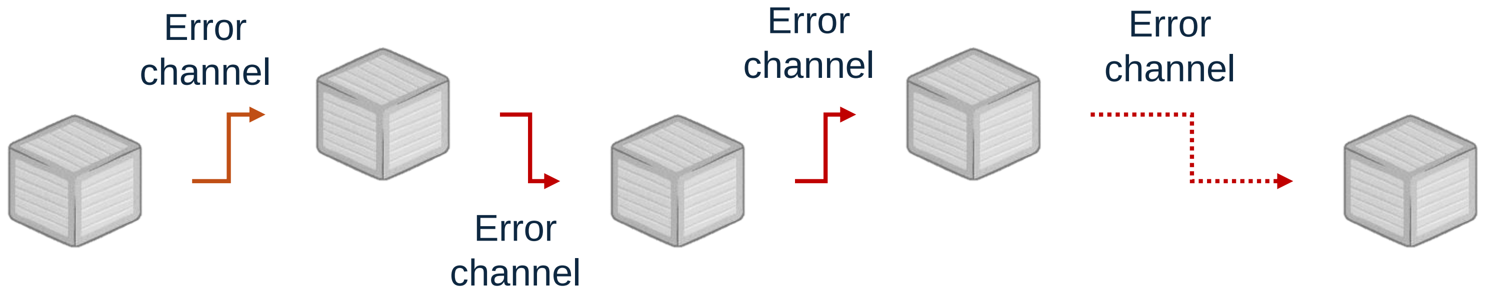
2
- Challenges with Dijkstra's classical algorithm (high time complexity)

3
- Quantum computing’s potential to solve this problem with improved time complexity

Multi-Nodes

Multi-Nodes teleportation has significant value due to long-distance delivery of quantum information. The possibility of multi-Nodes teleportation constituted by partially entangled pairs relates to the number of nodes.

The information transmitted from the source node to the destination node in this network are the quantum bits (qubits) instead of the classical bits (c-bits) in the classical network.



Quantum Algorithms Dijkstra for Shortest Path

- Nodes

```
# Add nodes to the graph
nodes = ['Alice', 'Node1', 'Node2', 'Node3', 'Node4', 'Bob']
```

- Weights

```
# Add edges to the graph with weights representing distances
edges = [('Alice', 'Node1', {'weight': 0.2}),
         ('Alice', 'Node2', {'weight': 0.3}),
         ('Node1', 'Node3', {'weight': 0.4}),
         ('Node2', 'Node3', {'weight': 0.5}),
         ('Node2', 'Node4', {'weight': 0.6}),
         ('Node3', 'Bob', {'weight': 0.7}),
         ('Node4', 'Bob', {'weight': 0.8})]
```

- Quantum Walk-based algorithm

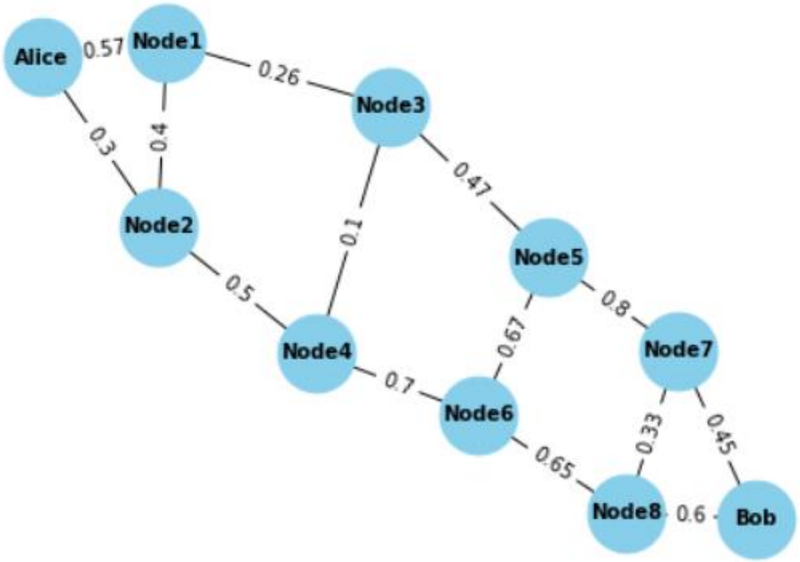
```
# Apply the quantum walk operator for a given number of steps
for _ in range(steps):
    for node in graph.nodes:
        for neighbor in graph.neighbors(node):
            qc.cx(qr[node_indices[node]], qr[node_indices[neighbor]])
            # Apply error channels
            for qubit_index in range(num_nodes):
                apply_error(qc, error_probabilities, qubit_index)
```

```
Quantum walk results:
{'100100': 17, '001101': 15, '111100': 20, '000100': 12, '110011': 16, '101101': 18, '011100': 17,
'111010': 13, '001111': 17, '100110': 15, '011111': 17, '001100': 23, '100011': 19, '111101': 15,
'110000': 13, '000111': 20, '101111': 21, '110001': 16, '000110': 15, '110100': 15, '000011': 14,
'101100': 23, '101010': 16, '000001': 14, '110110': 22, '110010': 10, '101110': 18, '000101': 18,
'010101': 18, '100010': 20, '111110': 23, '001011': 14, '001000': 15, '010000': 11, '100101': 14,
'001110': 21, '111011': 18, '001010': 17, '100001': 10, '111111': 15, '101011': 11, '000010': 18,
'110101': 17, '111000': 13, '010010': 11, '010011': 16, '011000': 14, '011101': 19, '011011': 9, '0
11110': 12, '000000': 14, '110111': 18, '010100': 14, '010111': 19, '100111': 11, '101001': 16, '10
0000': 16, '010110': 11, '101000': 20, '010001': 13, '011001': 20, '011010': 16, '001001': 16, '111
001': 15}
Shortest path from Alice to Bob : ['Alice', 'Node1', 'Node3', 'Bob']
```

Quantum Dijkstra Algorithms results

Quantum communication methods are extremely sensitive; even little changes in a parameter could make them ineffective. Our access to online quantum computers has demonstrated that most protocols, Quantum Dijkstra's Algorithm was used to calculate the graph path of 10 node between Alice and Bob.

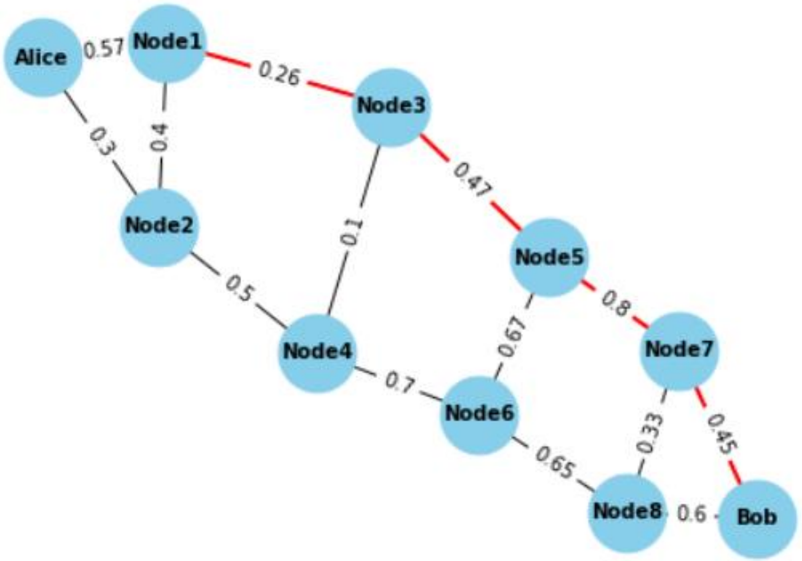
```python
# Quantum error channels
def apply_error(qc, error_probabilities, qubit_index):
    for error, prob in error_probabilities.items():
        if random.random() < prob:
            if error == "amplitude_damping":
                error_model = amplitude_damping_error(prob)
            elif error == "phase_flip":
                error_model = pauli_error([('Z', prob), ('I', 1 - prob)])
            elif error == "bit_flip":
                error_model = pauli_error([('X', prob), ('I', 1 - prob)])
            elif error == "depolarizing":
                error_model = depolarizing_error(prob, 1)
```



Quantum Dijkstra Algorithms results

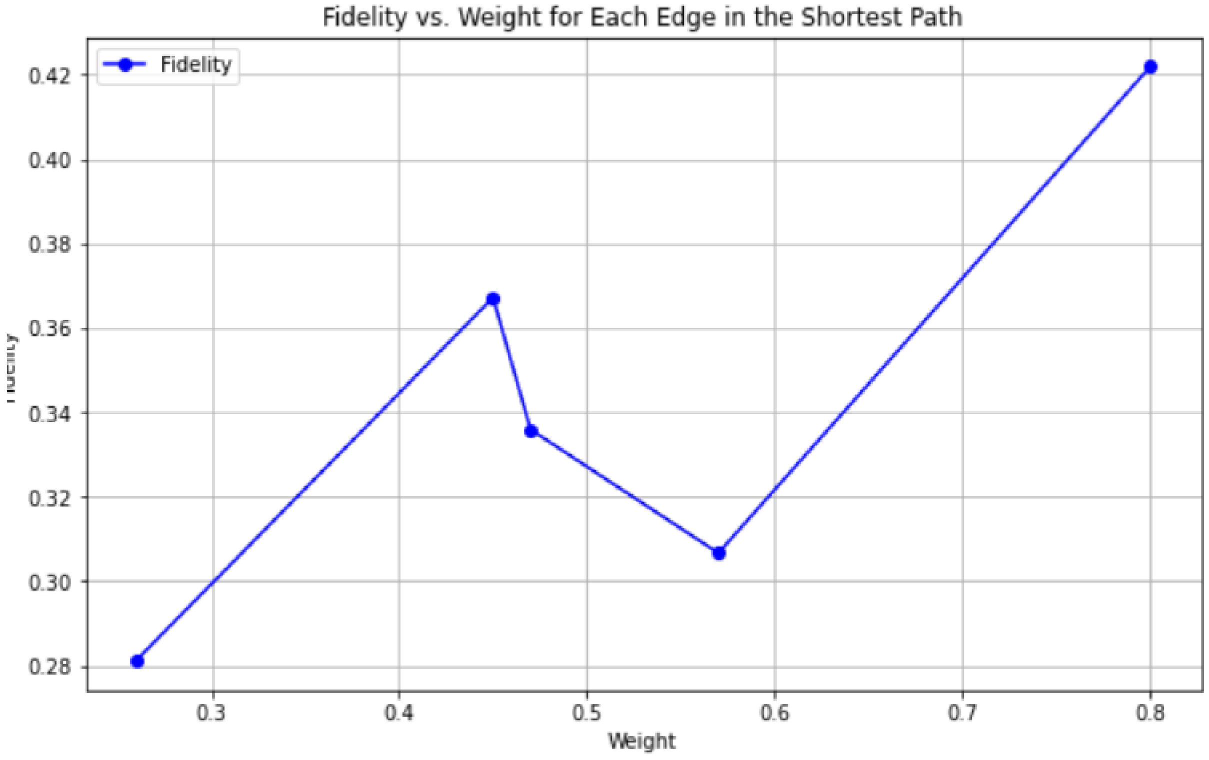
The main technique applied in this work is the quantum adaptation of Dijkstra's algorithm. The quantum circuit model, which is the standard structure in quantum computing, is used to implement this method. The IBM Qiskit framework, an open-source software development kit for quantum computing, is used in the implementation.

```
Shortest path from Alice to Bob : ['Alice', 'Node1', 'Node3', 'Node5', 'Node7', 'Bob']
```




```
Edge: (Alice, Node1), Weight: 0.57, Fidelity: 0.4296875
Edge: (Node1, Node3), Weight: 0.26, Fidelity: 0.306640625
Edge: (Node3, Node5), Weight: 0.47, Fidelity: 0.2802734375
Edge: (Node5, Node7), Weight: 0.8, Fidelity: 0.2724609375
Edge: (Node7, Bob), Weight: 0.45, Fidelity: 0.42578125
```

Quantum Dijkstra Algorithms Fidelity



Edge: (Alice, Node1), Weight: 0.57, Fidelity: 0.4296875
Edge: (Node1, Node3), Weight: 0.26, Fidelity: 0.306640625
Edge: (Node3, Node5), Weight: 0.47, Fidelity: 0.2802734375
Edge: (Node5, Node7), Weight: 0.8, Fidelity: 0.2724609375
Edge: (Node7, Bob), Weight: 0.45, Fidelity: 0.42578125

Conclusion

The aim of this research is to develop the Dijkstra algorithm, which is widely used in determining Information trajectory in computer networks. The use of quantitative calculation in the development of the Dijkstra algorithm is done How to invest the high potential for parallel treatment found in this method of calculation in the quantitative environment To deal with the increasing requirements resulting from the increase in the size of Internet networks

Thank You!