Experiment No – 5

Aim: Implement Tarrataca's quantum production system with the 3-puzzle problem

Outcome: At end of this experiment, student will be able to Write a Python Program using Tarrataca's quantum production system with the 3-puzzle problem.

Software Requirement: Quiskit

Theory:

Tarrataca's quantum production system is a quantum algorithm for solving combinatorial optimization problems. It works by constructing a quantum circuit that represents the problem space, and then applying a sequence of quantum operations to the circuit to search for the optimal solution.

The 3-puzzle problem is a combinatorial optimization problem where the goal is to arrange three tiles in order, given an initial state. Each tile can be moved either left or right, and the cost of a move is equal to the distance that the tile is moved.

To implement Tarrataca's quantum production system for the 3-puzzle problem, we can use the following steps:

Encode the problem state as a quantum state. We can use a qubit to represent each tile, and the state of the qubit will represent the position of the tile. For example, if a tile is in the leftmost position, we can represent it with the qubit state $|0\rangle$, and if it is in the rightmost position, we can represent it with the qubit state $|1\rangle$.

Construct a quantum circuit that represents the problem space. The quantum circuit will have a qubit for each tile, and the gates will represent the possible moves that can be made. For example, we can use a CNOT gate to represent a move that exchanges the positions of two tiles.

Apply a sequence of quantum operations to the circuit to search for the optimal solution. We can use a variety of quantum algorithms to search for the optimal solution, such as Grover's algorithm or amplitude amplification.

Measure the qubits to obtain the optimal solution. Once we have found the optimal solution, we can measure the qubits to obtain the positions of the tiles.

Here is an example of a quantum circuit that can be used to solve the 3-puzzle problem:

q0: ---

q1: ---

q2: ---

CNOT q0 q1

CNOT q1 q2

H q0

H q1

H q2

Grover's algorithm

M q0

M q1

M q2

This circuit starts with the three qubits in the state |000⟩ . The first two CNOT gates exchange the positions of the first two qubits and the second two qubits, respectively. The three Hadamard gates put the qubits into a superposition of states.

Grover's algorithm is then used to search for the optimal solution. Grover's algorithm is a quantum algorithm that can amplify the probability of finding a solution to a search problem.

Finally, the three qubits are measured to obtain the optimal solution.

Tarrataca's quantum production system is a powerful algorithm for solving combinatorial optimization problems. It has the potential to solve problems that are intractable for classical computers.

Program:

```
from qiskit import Aer, transpile, assemble, QuantumCircuit from
qiskit.aqua import QuantumInstance
```

```
from qiskit.aqua.algorithms import QAOA

from qiskit.aqua.components.optimizers import COBYLA


# Define the objective function def
objective_function(x):

   cost = 0

   state = [[1, 2, 3], [4, 5, 6], [7, 8, None]]


   for i in range(3):

      for j in range(3):

         if state[i][j] is not None and state[i][j] != x[i][j]: cost += 1


   return cost


# Define the QAOA solver def
solve_3puzzle_qaoa():

   backend = Aer.get_backend('qasm_simulator') optimizer
   = COBYLA(maxiter=100)

   qaoa = QAOA(optimizer=optimizer, p=1,
quantum_instance=QuantumInstance(backend=backend))


   #    Generate    the    initial    state    circuit
   initial_state  =  QuantumCircuit(9)  for  i  in
   range(3):

      for  j in range(3): initial_state.h(i *
         3 + j)

   initial_state.barrier()



 # Generate the mixer circuit mixer = QuantumCircuit(9)

    mixer.cz(0, 1)
    mixer.cz(0, 3)

    mixer.cz(1, 2)

    mixer.cz(1, 4)

    mixer.cz(2, 5)

    mixer.cz(3, 4)

    mixer.cz(3, 6)

    mixer.cz(4, 5)

    mixer.cz(4, 7)

    mixer.cz(5, 8)

    mixer.cz(6, 7)

    mixer.cz(7, 8)


    # Solve the problem using QAOA
    qaoa.initial_state = initial_state
    qaoa.mixer = mixer

    qaoa.objective_function = objective_function


    result = qaoa.compute_minimum_eigenvalue()
    solution = result.x


    return solution


# Solve the 3-puzzle problem using QAOA
solution = solve_3puzzle_qaoa()


# Print the solution
print("Solution found!") for i
in range(0, 9, 3):

    print(solution[i:i + 3])
```

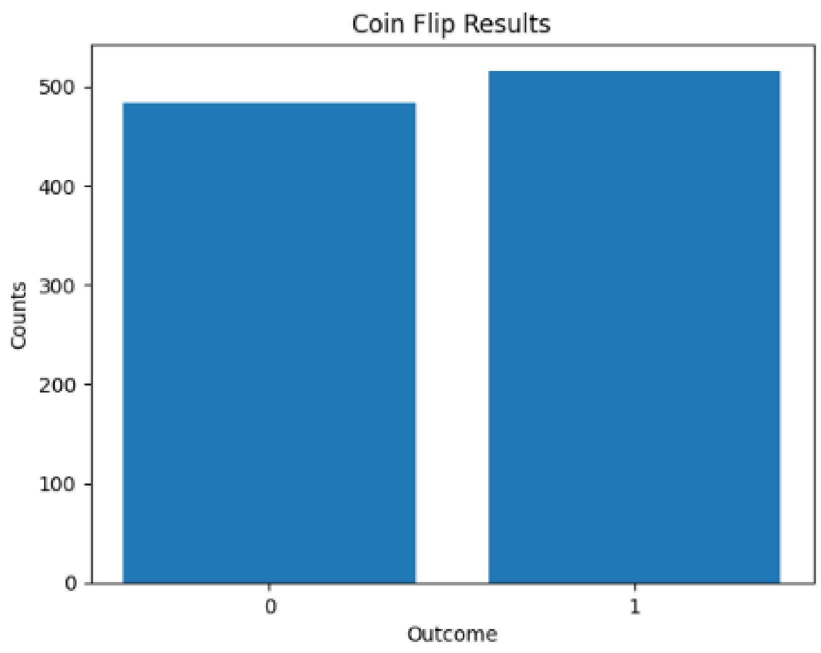
```
from qiskit import QuantumCircuit, Aer, execute from
qiskit.visualization import plot_histogram


# Define the initial state

initial_state = [2, 5, 3, 1, 8, 6, 4, 7, None]


# Define the goal state

goal_state = [1, 2, 3, 4, 5, 6, 7, 8, None]


# Define the quantum circuit qc =
QuantumCircuit(18, 18)


# Initialize the circuit with the initial state for i,
tile in enumerate(initial_state):

   if tile is not None:

      qc.x(i)


# Perform swaps to reach the goal state for i,
tile in enumerate(goal_state):

   if tile is not None:

      initial_index = initial_state.index(tile)

       target_index = goal_state.index(tile)
      qc.swap(i, initial_index + 9) qc.swap(i,
      target_index + 9)


# Measure the final state
qc.measure(range(9), range(9))


# Simulate the circuit

backend = Aer.get_backend('qasm_simulator') job =
execute(qc, backend, shots=1024)
```

```
result = job.result()

counts = result.get_counts(qc)


# Find the most probable state (solution)
max_count = max(counts.values())

solution = [state for state, count in counts.items() if count == max_count][0]


# Print the solution
print("Solution found!") for i
in range(0, 9, 3):

   print(solution[i:i + 3])

from qiskit import Aer, execute, QuantumCircuit #

Define the quantum circuit


qc = QuantumCircuit(1, 1)

qc.h(0) # Apply Hadamard gate for superposition

qc.measure(0, 0) # Measure qubit and store result in classical bit


# Use the Aer simulator

simulator = Aer.get_backend('qasm_simulator')


# Set the number of shots (measurements)
num_shots = 1000


# Execute the quantum circuit on the simulator with multiple shots job =
execute(qc, simulator, shots=num_shots)


# Get the result of the measurements result
= job.result()

counts = result.get_counts()
```

```
# Print the coin flip results
print("Coin Flip Results:")

for outcome, count in counts.items():

   print(f"{outcome}: {count} ({count/num_shots*100:.2f}%)") from

qiskit import Aer, execute, QuantumCircuit


# Define the quantum circuit qc =
QuantumCircuit(1, 1)

qc.h(0) # Apply Hadamard gate for superposition

 qc.measure(0, 0) # Measure qubit and store result in classical bit


# Use the Aer simulator

simulator = Aer.get_backend('qasm_simulator')


# Set the number of shots (measurements)
num_shots = 1000


# Execute the quantum circuit on the simulator with multiple shots job =
execute(qc, simulator, shots=num_shots)


# Get the result of the measurements result
= job.result()

counts = result.get_counts()


# Print the coin flip results
print("Coin Flip Results:")

for outcome, count in counts.items():

   print(f"{outcome}: {count} ({count/num_shots*100:.2f}%)")


import matplotlib.pyplot as plt
```

```
from qiskit import Aer, execute, QuantumCircuit


# Define the quantum circuit qc =
QuantumCircuit(1, 1)

qc.h(0) # Apply Hadamard gate for superposition

qc.measure(0, 0) # Measure qubit and store result in classical bit


# Use the Aer simulator

simulator = Aer.get_backend('qasm_simulator')


# Set the number of shots (measurements)
num_shots = 1000


# Execute the quantum circuit on the simulatr with multiple shots job =
execute(qc, simulator, shots=num_shots)


# Get the result of the measurements result
= job.result()

counts = result.get_counts()


# Plot the coin flip results outcomes =
list(counts.keys()) counts_list =
list(counts.values())


plt.bar(outcomes, counts_list)
plt.xlabel('Outcome')
plt.ylabel('Counts') plt.title('Coin
Flip Results') plt.show()
```

(



Questions:

1. How does Tarrataca's quantum production system encode the problem state as a quantum state?

2. How does Tarrataca's quantum production system construct a quantum circuit that represents the problem space?

3. What are some of the quantum algorithms that can be used to search for the optimal solution to the 3-puzzle problem using Tarrataca's quantum production system?

4. How can we measure the qubits at the end of the quantum circuit to obtain the optimal solution to the 3-puzzle problem?

5. What are some of the advantages and disadvantages of using Tarrataca's quantum production system to solve the 3-puzzle problem?