Project_4

November 27, 2024

1 Introduction to Quantum Information and Quantum Machine Learning

Instructor: Dr Sci. Eng. Przemysław Głowacki

Kacper Dobek 148247

[93]:
```
import numpy as np
from math import sqrt, floor
from numpy import pi
from qiskit import *
from qiskit.quantum_info import Statevector, Operator
from qiskit.circuit.library.standard_gates import XGate
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from time import process_time
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
```

[94]:
```
backend = BasicAer.get_backend("qasm_simulator")
```

1.0.1 Task 1: Optimal r

[95]:
```
def calculate_optimal_r(n: int) -> int:
    return math.floor((pi / 4) * sqrt(2**n))


n_values = [2, 3, 4, 5, 6]

for n in n_values:
    optimal_r = calculate_optimal_r(n)
    print(f"Optimal r for n={n} is {optimal_r}")
```

```
Optimal r for n=2 is 1
Optimal r for n=3 is 2
Optimal r for n=4 is 3
Optimal r for n=5 is 4
Optimal r for n=6 is 6
```

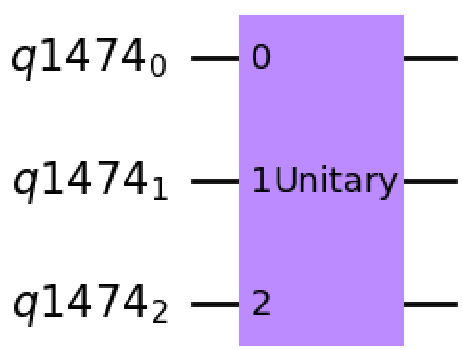
[96]:
```python
# Construction of the Uf matrix for n=3

a = 2
n = 3
oracle = np.identity(2**n)
oracle[a, a] = -1
print("Oracle:")
print(oracle)

Uf = Operator(oracle)
```

```
Oracle:
[[ 1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. -1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.]]
```

The Uf circuit

[97]:
```python
q0 = QuantumRegister(n)
CircuitUf = QuantumCircuit(q0, name="Uf")
CircuitUf.append(Uf, [q0[0], q0[1], q0[2]])
CircuitUf.draw(output="mpl")
```

[97]:



1.0.2 Task 2

The class below implements necessary functions for performing the experiments.

[98]:
```python
class GroverQuantumExperiment:
    def __init__(self, n: int, a: int, backend, plot=False) -> None:
        self.n = n
        self.a = a
        self.r = calculate_optimal_r(n)
        self.shots = 2 ** (n + 7)
        self.backend = backend
        self.mccx = XGate().control(n - 1)
        self.uf_circuit = self.construct_uf_circuit()
        self.final_circuit = self.construct_final_circuit()
        if plot:
            display(self.final_circuit.draw(output="mpl"))

    def change_r(self, r: int) -> None:
        self.r = r

    def run_experiment(self, circuit: QuantumCircuit = None) -> dict:
        if circuit is None:
            circuit = self.final_circuit
        job_sim0 = execute(circuit, self.backend, shots=self.shots)
        sim_result0 = job_sim0.result()
        return sim_result0.get_counts(circuit)

    def run_experiment_stepwise(self, upper: int) -> None:
        intermediate_results = []
        q = QuantumRegister(self.n)
        c = ClassicalRegister(self.n)
        Circuit = QuantumCircuit(q, c)
        for i in range(self.n):
            Circuit.h(q[i])
        Circuit.barrier()
        for s in range(1, upper + 1):
            Circuit = self.extend_circuit_WV(Circuit, q, c)
            Circuit_to_measure = Circuit.copy()
            for i in range(self.n):
                Circuit_to_measure.measure(q[i], c[i])
            intermediate_result = self.run_experiment(Circuit_to_measure)
            intermediate_results.append(intermediate_result)

        a_proba = self.process_intermediate_results(intermediate_results)
        self.plot_probabilities(a_proba)

    def plot_probabilities(self, a_proba: list) -> None:
        fig, ax = plt.subplots(figsize=(10, 5))
        ax.bar(range(1, len(a_proba) + 1), a_proba)
        ax.xaxis.set_major_locator(MaxNLocator(integer=True))
        ax.set_xlabel("Step number")
```

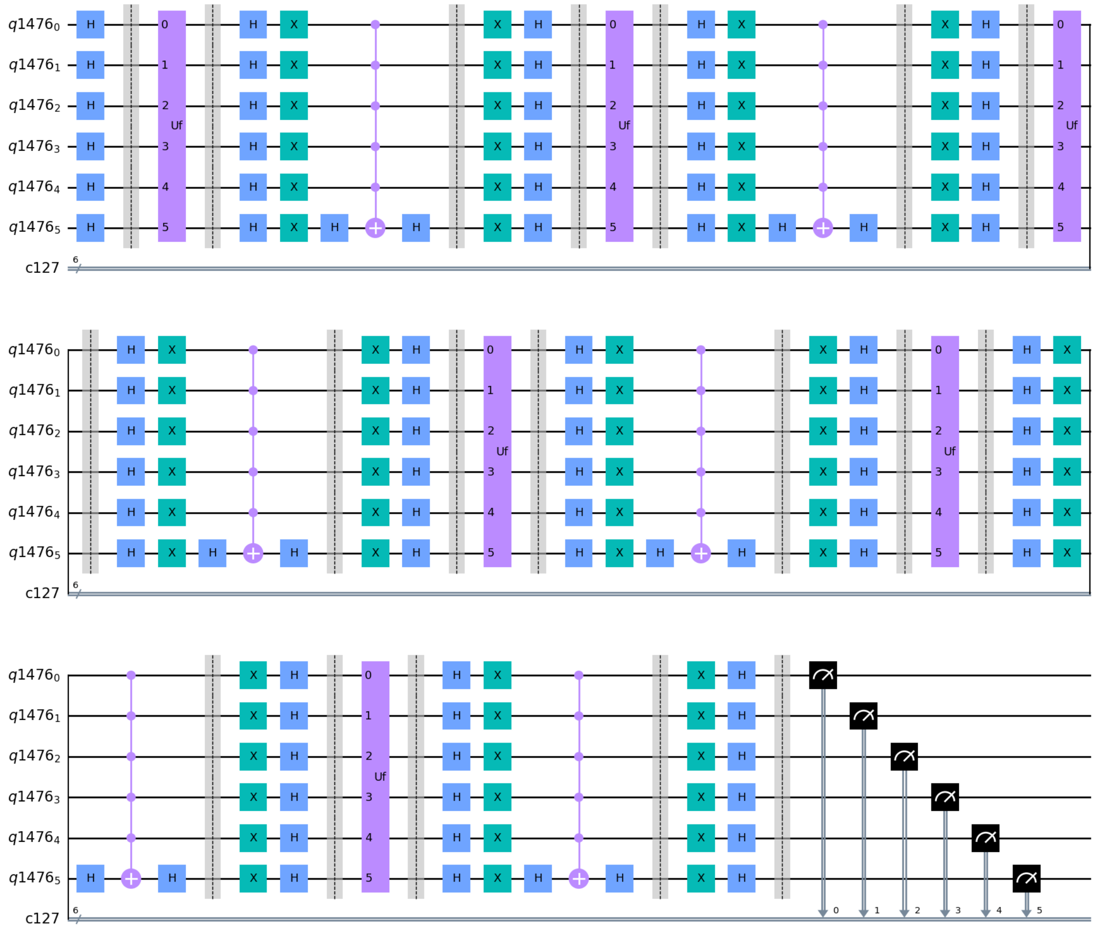
```
        ax.set_ylabel("Probability of detecting |a>")
        ax.set_title(f"Probability of detecting a = {self.a} in {len(a_proba)}␣
↪steps")
        plt.show()

    def process_intermediate_results(self, intermediate_results: list) -> list:
        a_proba = []
        # Get the binary representation of a
        a_binary = format(self.a, f"0{self.n}b")
        for intermediate_result in intermediate_results:
            if a_binary in intermediate_result:
                a_proba.append(
                    intermediate_result[a_binary] / sum(intermediate_result.
↪values())
                )
        return a_proba

    def construct_uf_circuit(self) -> QuantumCircuit:
        oracle = np.identity(2**self.n)
        oracle[self.a, self.a] = -1
        Uf = Operator(oracle)
        q0 = QuantumRegister(self.n)
        CircuitUf = QuantumCircuit(q0, name="Uf")
        CircuitUf.append(Uf, [q0[i] for i in range(self.n)])
        uf = CircuitUf.to_gate()
        return uf

    def construct_final_circuit(self) -> QuantumCircuit:
        q = QuantumRegister(self.n)
        c = ClassicalRegister(self.n)
        Circuit = QuantumCircuit(q, c)
        # |fi> State initiation
        for i in range(self.n):
            Circuit.h(q[i])
        Circuit.barrier()
        for _ in range(self.r):
            Circuit = self.extend_circuit_WV(Circuit, q, c)
        for i in range(self.n):
            Circuit.measure(q[i], c[i])
        return Circuit

    def extend_circuit_WV(self, Circuit, q, c) -> QuantumCircuit:
        Circuit.append(self.uf_circuit, [[i] for i in range(self.n)])
        Circuit.barrier()
        # Beginning of the implementation of the W diffusion operator
        for i in range(self.n):
            Circuit.h(q[i])
```

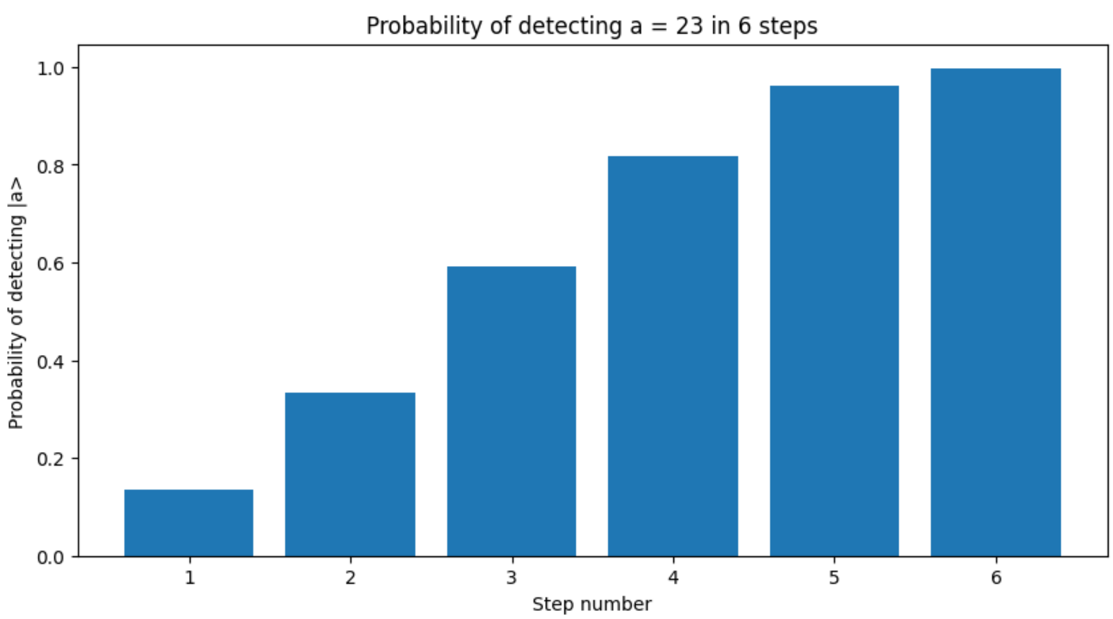
```
            Circuit.x(q[i])
        Circuit.h(q[self.n - 1])
        Circuit.append(self.mccx, [q[i] for i in range(self.n)])
        Circuit.h(q[self.n - 1])
        Circuit.barrier()
        for i in range(self.n):
            Circuit.x(q[i])
            Circuit.h(q[i])
        # The end of the implementation of the W diffusion operator
        Circuit.barrier()

        return Circuit
```

1.0.3 Task 3

[99]:
```
n = 6
a = 148247 % 2**n
r = calculate_optimal_r(n)

print(f"n={n}, a={a}")

gqe = GroverQuantumExperiment(n, a, backend, plot=True)
gqe.run_experiment_stepwise(upper=r)
```

```
n=6, a=23
```

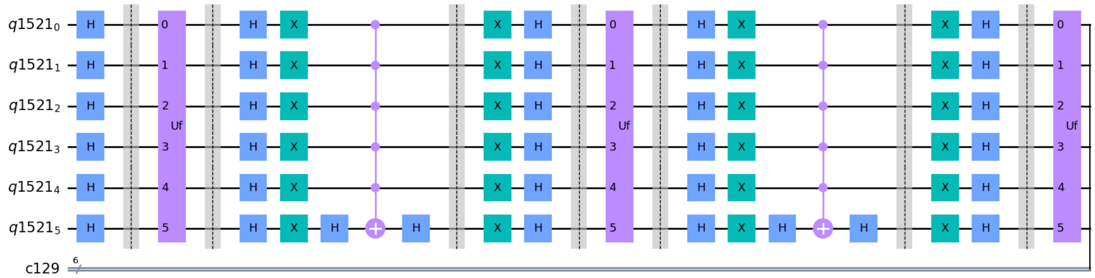

q1476_0
q1476_1
q1476_2
q1476_3
q1476_4
q1476_5
c127
H
X
Uf
6

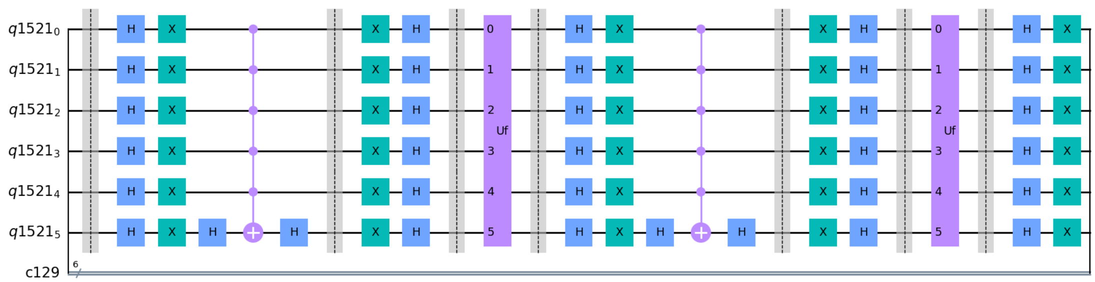

Probability of detecting a = 23 in 6 steps
Probability of detecting |a>
Step number
0.0
0.2
0.4
0.6
0.8
1.0
1
2
3
4
5
6

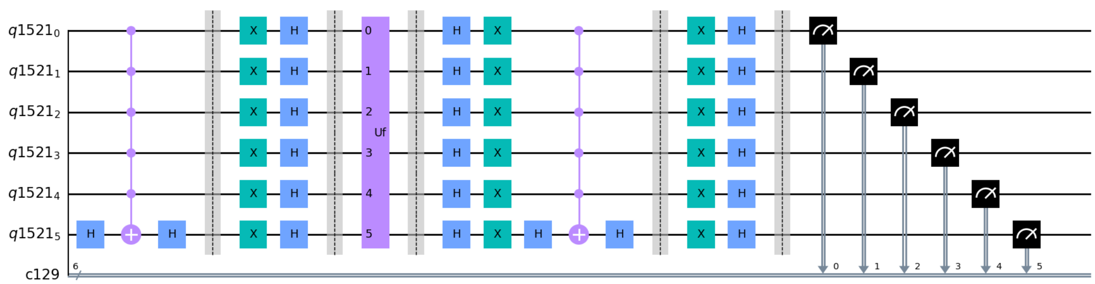
1.0.4 Task 4

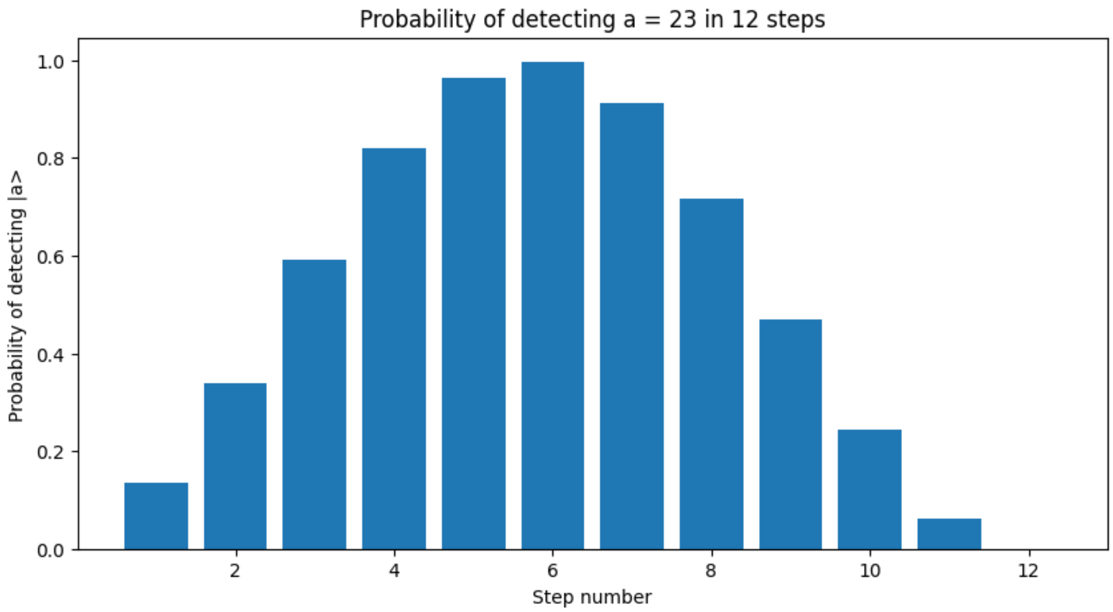
[100]:
```
n = 6
a = 148247 % 2**n
upper = floor(pi / 2 * sqrt(2**n))

print(f"n={n}, a={a}")

gqe = GroverQuantumExperiment(n, a, backend, plot=True)
gqe.run_experiment_stepwise(upper=upper)
```

```
n=6, a=23
```





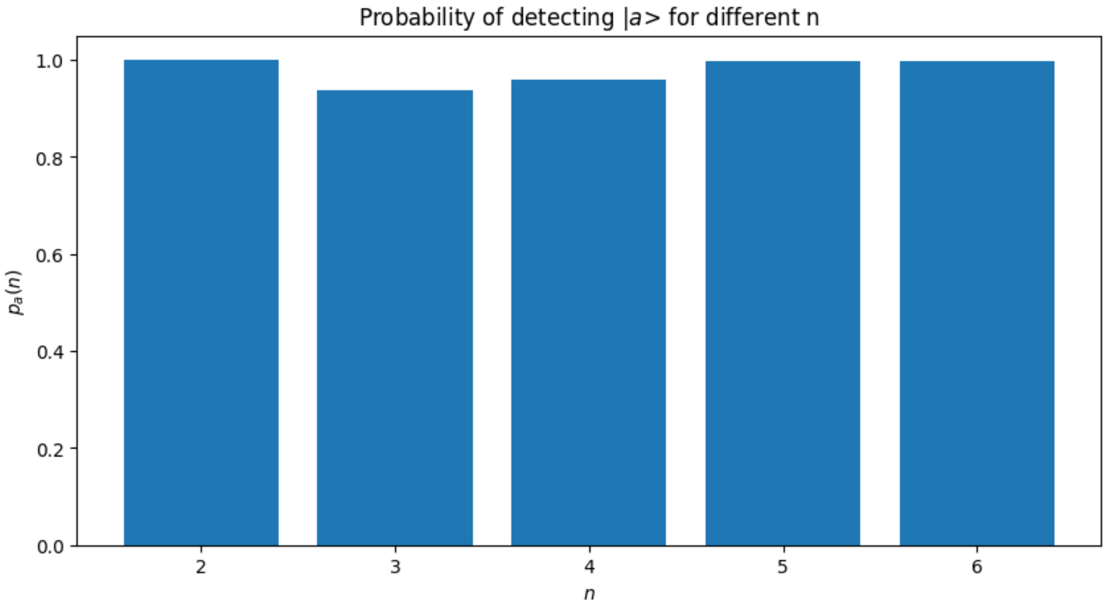




1.0.5 Task 5

[101]:
```
n_a_proba = []

for n in n_values:
    a = 148247 % 2**n
    print(f"n={n} a={a}")
    gqe = GroverQuantumExperiment(n, a, backend, plot=False)
    results = gqe.run_experiment()
    a_proba = gqe.process_intermediate_results([results])
    n_a_proba.append(a_proba[0])

fig, ax = plt.subplots(figsize=(10, 5))
ax.bar(n_values, n_a_proba)
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
ax.set_xlabel("$n$")
ax.set_ylabel("$p_{a}(n)$")
ax.set_title(f"Probability of detecting |$a$> for different n")
plt.show()
```

```
n=2 a=3
n=3 a=7
n=4 a=7
n=5 a=23
n=6 a=23
```


Probability of detecting |a> for different n
$p_a(n)$
0.0
0.2
0.4
0.6
0.8
1.0
2
3
4
5
6
n