Introduction to Quantum Computing MidTerm Eval

Aakarsh Jain
Roll Number - 2021507
IIIT - Delhi
aakarsh21507@iiitd.ac.in

Raghav Sakhuja
Roll Number - 2021274
IIIT - Delhi
raghav21274@iiitd.ac.in

Siddhant Rai Viksit
Roll Number - 2021565
IIIT - Delhi
siddhant21565@iiitd.ac.in

3 April 2024

1 Problem

For a function $f : [1, N] \longrightarrow [a, b]$, we want to find i, j s.t $f(i) = f(j)$ and $i \neq j$. A classical algorithm for this problem would be $O(n)$, however, with quantum search we can do better than this. In this report, we present an approach using Grover's search that runs in $O(n^{3/4})$.

2 Solution

2.1 Algorithm

We follow the algorithm described in [2].

2.1.1 Grover's search

For a given f, we define and implement a unitary U_f which gives us query access to the function.

$$U_f |x\rangle |i\rangle = |x\rangle |f(x) \oplus i\rangle$$

To construct our oracle, we use a combination of two oracles, which are the above defined oracle U_f, and an $A_f(M)$, which takes as a parameter a set of numbers M, and uses phase kickback if the input $x \in M$. These two combine to form our quantum oracle, which forms the basis for our Grover iterator. Since we don't know the number of pairs of values which map to the same value in f, we use the version of Grover's search designed for an unknown θ, as described in lectures, with $m_0 = 2$ and $\lambda = 1.2$.

2.2 Analysis

Since our algorithm's time complexity is $O(n^{3/4})$, we would expect, on average, for around $n^{3/4}$ queries to be made to the oracle. Our algorithm succeeds with probability $\epsilon = \frac{const}{\sqrt{n}}$[1], and we can get it to a constant probability within $O(\frac{1}{\sqrt{\epsilon}})$, which, since Grover's search is $O(\sqrt{n})$, means a total time complexity of $O(n^{1/2}n^{1/4}) = O(n^{3/4})$

3 Result

The python implementation for the above can be found here. We ran 1000 tests on array of size $N = 100$, tracking the number of calls made to the oracle.

To run the experiment locally, clone the above repository, and run the following commands -

```
python3 -m venv ./.venv
source ./.venv/bin/activate
pip install -r requirements.txt
python3 elemental_distinctiveness.py
```

The number of test iterations and size of the array can be modified inside the code if required.

When we set the array to only contain elements $1 \ldots N$(i.e.,$f : [1, n] \to [1, N]$), the number of calls were 51.62, which is around the same bound that the authors propose of $O(\sqrt{n})$. The error measured (Arrays misclassified by Quantum algorithm) is 13%

The circuits used have also been added to the attached .ipynb file if required.

4 Conclusions

As mentioned in our proposal, we have implemented [2] for element distinctiveness. We now aim to implement [3], which has a lower asymptotic complexity, and apply it to the problem of triangle finding in a graph.

References

[1] Andris Ambainis. *Quantum search algorithms*. 2005. arXiv: quant-ph/0504012 [quant-ph].

[2] Andris Ambainis. "Quantum walk algorithm for element distinctness". In: *SIAM Journal on Computing* 37.1 (2007), pp. 210–239.

[3] Frederic Magniez, Miklos Santha, and Mario Szegedy. *Quantum Algorithms for the Triangle Problem*. 2005. arXiv: quant-ph/0310134 [quant-ph].