Scientific Research Publishing

American Journal of Computational Mathematics, 2024, 14, 1-10
https://www.scirp.org/journal/ajcm
ISSN Online: 2161-1211
ISSN Print: 2161-1203

Time Complexity of the Oracle Phase in Grover's Algorithm

Ying Liu

Department of Engineering Technology, Savannah State University, Savannah, Georgia
Email: liuy@savannahstate.edu

How to cite this paper: Liu, Y. (2024) Time Complexity of the Oracle Phase in Grover's Algorithm. *American Journal of Computational Mathematics*, **14**, 1-10.
https://doi.org/10.4236/ajcm.2024.141001

Received: February 15, 2024
Accepted: March 22, 2024
Published: March 25, 2024

Abstract

Since Grover's algorithm was first introduced, it has become a category of quantum algorithms that can be applied to many problems through the exploitation of quantum parallelism. The original application was the unstructured search problems with the time complexity of $O(\sqrt{N})$. In Grover's algorithm, the key is Oracle and Amplitude Amplification. In this paper, our purpose is to show through examples that, in general, the time complexity of the Oracle Phase is $O(N)$, not $O(1)$. As a result, the time complexity of Grover's algorithm is $O(N)$, not $O(\sqrt{N})$. As a secondary purpose, we also attempt to restore the time complexity of Grover's algorithm to its original form, $O(\sqrt{N})$, by introducing an $O(1)$ parallel algorithm for unstructured search without repeated items, which will work for most cases. In the worst-case scenarios where the number of repeated items is $O(N)$, the time complexity of the Oracle Phase is still $O(N)$ even after additional preprocessing.

Keywords

Quantum Computing, Oracle, Amplitude Amplification, Grover's Algorithm

1. Introduction

Quantum Computing [1] [2] [3] [4] [5] is a field of computing that leverages the principles of Quantum Mechanics. Traditional computers use bits as the fundamental unit of information, which can exist in one of two states: 0 or 1. Quantum computers, on the other hand, use qubits, which can exist in multiple states simultaneously. Quantum Computing has strange phenomena known as Superposition and Entanglement.

Grover's algorithm [6] is one of the most famous quantum algorithms that provide a quadratic speedup over the best classical algorithms for unstructured

search problems, *i.e.* from $T = O(N)$ to $T = O(\sqrt{N})$. It was proposed by Lov Grover [6] in 1996 and is a fundamental algorithm in the field of Quantum Computing. Its efficiency arises from the exploitation of quantum parallelism and quantum interference. Furthermore, it has evolved into a category of algorithms that can be applied to many problems, such as SAT [7], and Subset Sum [8].

Grover's algorithm [6] is:

Initialization;

Oracle;

for ($i = 0$; $i < O(\sqrt{N})$, i++)

    Amplitude Amplification;

Measurement;

where:

Initialization: Start with a superposition of all possible states. If there are $N = 2^n$ possible solutions, where n is the number of qubits, this superposition is created over N states.

Oracle: Introduce an Oracle gate that identifies the target solution.

Amplitude Amplification: Apply a series of quantum operations that amplify the amplitude of the marked state and suppress the amplitudes of the other states.

Repeat Amplification: Amplifications are repeated for a certain number of iterations.

Measurement: The quantum state is measured, and with high probability, the correct solution is obtained.

We will divide Grover's algorithm into two phases: Oracle Phase and Amplitude Amplification Phase, where the Oracle Phase consists of Initialization and Oracle. The job of the Oracle is to mark the solution. Amplitude Amplification is the constructive and destructive interference that occurs during the Amplitude Amplification step. The amplitudes of incorrect states experience destructive interference, reducing their probabilities, while the amplitude of the correct state experiences constructive interference, increasing its probability.

To date, no one has challenged the main conclusion of Grover's algorithm: $T = O(\sqrt{N})$ for both Amplitude Amplification and the entire algorithm. Inexplicitly, the algorithm assumes that Oracle Phase, Initialization and Oracle, has $T = O(1)$. In Grover's algorithm, the starting superposition of all possible states is given in Equation (5), which is indeed O(1). In this paper, we will show that $T = O(1)$ for the Oracle Phase only applies to scenarios where:

- the target number can be found in the unstructured data, and
- there are no repeated items in the unstructured data.

We will show that if any one of the above conditions is violated, then, $T = O(N)$ for the Oracle Phase. Furthermore, if both of the above conditions are satisfied, the algorithm's behavior is predictable, *i.e.* there is no need for the algorithm at all.

When the target is not in the list, the target must be rejected by the Oracle Phase so the algorithm will not enter into the next phase. However, if the list does not contain the target item, the oracle operation will not find any valid solution to mark and stop the algorithm. This step requires all of the items in the unsorted list to be encoded into the initial superposition state, which is O(N).

When the list has repeated items, the normalization of the initial superposition state must be reconstructed if it is required to find all of the matching items. All of the items in the unsorted list need to be encoded into the initial superposition state, which is also O(N).

This paper will make a high-level logical discussion rather than whether a particular task is possible or not at the quantum circuit level. For example, we will simply assume that we can produce an initial superposition state based on a given list with respect to each item in the list.

Clearly, if the time complexity of the Oracle Phase is O(N), then Grover's algorithm is also O(N). Even if we have shown that the Oracle has $T = O(N)$, we will need to at least attempt to show that this is not trivial, *i.e.* there are no trivial solutions to restore the time complexity of Oracle Phase to O(1). We can restore the time complexity of Grover's algorithm to $O(\sqrt{N})$ by introducing an O(1) parallel SIMD algorithm for unstructured search without repeated items, which will work for most cases. In the worst-case scenarios where the number of repeated items is O(N), the time complexity of the Oracle Phase is still O(N). The SIMD architecture is not readily available, especially when a search list is large, so this is only a solution in theory.

This paper is organized as follows:

Section '2. Basic Notation and Background' first introduces $X = \{0,1\}^d$ space. Then, we will introduce the notation for superposition vectors.

Section '3. $T = O(1)$ for the Oracle Phase' describes the situation where the initialization is fixed and is independent of a particular unsorted list to be searched.

Section '4. Two Problems for the Oracle Phase' describes the necessity of encoding the initial superposition states from the unsorted list.

Section '5. An O(1) Parallel Algorithm for Unstructured Search without Repeated Items' attempts to restore the time complexity of Grover's algorithm to $O(\sqrt{N})$ with a SIMD algorithm. This algorithm works for most cases. Because it is not really practical to build a SIMD architecture for a large amount of processing units, this is a solution in theory.

Section '6. A Parallel Algorithm for Unstructured Search with Repeated Items' shows that in the worst-case scenarios, the time complexity of the Oracle Phase is still O(N).

2. Basic Notation and Background

Throughout this paper,

n is the number of qubits;

$N = 2^n$ is the number of states;

$L <= N$ is the number of items in an unsorted list;

M is the unsorted list.

The unstructured search problems have:

$$T = \mathrm{O}(N) = \mathrm{O}(2^n) \tag{1}$$

Grover's algorithm [6] provides a quadratic speedup over the best classical algorithms for unstructured search problems, *i.e.* from $T = \mathrm{O}(N)$ to $T = \mathrm{O}(\sqrt{N})$. Today, Grover's algorithm is also a category of algorithms that can be applied to many problems [7] [8].

An Instance Space, $X = \{0, 1\}^n$, is a set of all instances given in Equation (2):

$$X = \{0\cdots00, 0\cdots01, 0\cdots10, 0\cdots11, \cdots\} \tag{2}$$

An instance of X is $x \in X$, where $x = 00\cdots0$, or, $0\cdots01$, $\cdots$, and $|X| = 2^n$. An instance, x, is a binary string, which can be converted into a decimal number:

$$X = \{0, 1, 2, 3, \cdots, N-1\} \tag{3}$$

An instance, x, can be the qubits. It can appear as a binary string or a decimal number. Examples are: $|000\rangle = |0\rangle$, $|001\rangle = |1\rangle$, $\cdots$, $|111\rangle = |7\rangle$.

In Grover's algorithm, the starting superposition of all possible states,

$$|\psi\rangle = \sum_{x=0}^{N-1} a(x)|x\rangle, \tag{4}$$

is always:

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \tag{5}$$

Here $a(x)$ in Equation (4) are the amplitudes. Equation (5) comes from:

$$|\psi\rangle = |x_{n-1}\rangle \otimes \cdots \otimes |x_1\rangle \otimes |x_0\rangle \tag{6}$$

where:

$$\begin{aligned} |x_0\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle = H|0\rangle \\ |x_1\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle = H|0\rangle \\ &\vdots \end{aligned} \tag{8}$$

where H is the Hadamard gate. The time complexity for Equation (5) is O(1). Note that although there are n qubits and each qubit will go through Equation (9):

$$|+\rangle = H|0\rangle \tag{9}$$

the n steps proceed in parallel so the time for Equation (7), (8), … , together is O(1). For the same reason, the superposition of ψ in equation (6) from n qubit, $\{\cdots, x_2, x_1, x_0\}$, is also O(1).

The advantage of the unstructured search problem is that the Oracle is easy to construct, *i.e.* the Oracle merely marks the target number, which is given. For example, let $n = 3$, and we want to search a target, $|5\rangle = |101\rangle$, from an arbitrary random list, $M = \{3, 4, 6, 0, 1, 2, 7, 5\}$. Since we know $|101\rangle$ is the solution, the Oracle simply marks the $|101\rangle$ state.

3. $T = O(1)$ for the Oracle Phase

Grover's algorithm starts with the initialization, which starts with a superposition of all possible states. If there are $N = 2^n$ possible solutions, where n is the number of qubits, this superposition is created over N states given in Equation (5). Already, we have noticed that the Initialization is search-list independent. The only requirements are that the random list:

$$M = \{m_0, m_1, \cdots, m_{L-1}\} \tag{10}$$

can be encoded into n qubits in theory.

The Oracle identifies the target; since the target in the random-list search problem is given, the Oracle's job is simply to mark it. One thing is immediately obvious. Assuming that we want to find 5 from two arbitrary search lists, say, {3, 4, 6, 0, 1, 2, 7, 5} and {3, 4, 6, 0, 7, 5, 1, 2}, the initial superposition states will be the same in both cases, because the initial superposition state is Equation (5).

In fact, Grover's algorithm, based on the initial superposition state alone, can always find a target from {0, 1, 2, … , $N-1$}. Any search problem will result in exactly the same answer: Found. Furthermore, the behavior of the algorithm is fixed and predictable; therefore, there is no need to even run through Grover's algorithm at all, because we already know the answer: Found.

In other words, the very existence of Grover's algorithm is not necessary, because we know the outcome of the algorithm ahead of time: there is only one answer: Found. This, of course, is exactly the problem, as we can see from the examples below.

Example 1. Finding 5 from {3, 4, 6, 0, 1, 2, 7, 5}.

Answer: found 5, because algorithm has only one answer.

Example 2. Finding 6 from {3, 4, 6, 0, 7, 5, 1, 2}.

Answer: found 6, because algorithm has only one answer.

Example 3. Finding 7 from {3, 3, 0, 0, 4, 2, 0, 0}.

Answer: found 7, because algorithm has only one answer.

You can see the problem in Example 3; the target is not in the list and the answer is wrong.

4. Two Problems for the Oracle Phase

The Amplitude Amplification Phase cannot reject a state. When the target is not in the list, the target must be rejected by the Oracle Phase. The Oracle is designed to mark the target solution in the quantum state. The Oracle would flip the sign of the amplitude of the state representing the target. However, if the list

does not contain the target item, the Oracle will not find any valid solution to mark and stop the algorithm.

Equation (5), which contains all of the possible items in the unsorted list, is no longer valid now. All of the items in the list, and only all of the items in the list, can be encoded into the initial superposition state. Given an unsorted list,

$$M = \{m_0, m_1, \cdots, m_{L-1}\},$$

the initial superposition state should be:

$$|\psi\rangle = \sum_{x=0}^{L-1} a(m_x)|m_x\rangle. \tag{11}$$

This initialization is problem-dependent. Equation (11) is built from Equation (10) by looping each item in Equation (10). The time complexity for the loop is O(N) rather than O(1). This is the cost of correcting Example 3.

Example 3. Finding 7 from {3, 3, 0, 0, 4, 2, 0, 0}.

The initial superposition state below has O(N) steps by going through each item in the unsorted list:

$$\begin{aligned}|\psi\rangle &= \frac{1}{\sqrt{8}}\left(\frac{1}{\sqrt{2}}|011\rangle + \frac{1}{\sqrt{2}}|011\rangle + \frac{1}{\sqrt{4}}|000\rangle + \frac{1}{\sqrt{4}}|000\rangle\right.\\ &\quad \left. + |100\rangle + |010\rangle + \frac{1}{\sqrt{4}}|000\rangle + \frac{1}{\sqrt{4}}|000\rangle\right)\\ &= \frac{1}{\sqrt{8}}\left(\sqrt{2}|011\rangle + \sqrt{4}|000\rangle + |100\rangle + |010\rangle\right)\end{aligned}$$

The Oracle is designed to mark the target solution in the quantum states. In this example, however, the list does not contain the target item, $|7\rangle = |111\rangle$. The Oracle would stop the algorithm. The cost for the Oracle is O(N) instead of O(1).

Also, repeated items will introduce an extra factor because the superposition state is normalized; for example, in the above example, because item, 0, appears four times, it has an extra normalization factor:

$$\frac{1}{\sqrt{4}}|000\rangle$$

The extra factor depends on the number of times an item is repeated, which can be found by looping through the list, which again is O(N).To summarize, there are problems:

- The search target is not in the random list;
- There are repeated items in the random search list.

If Equation (11) has O(N), Grover's algorithm has lost all of its advantages, at least for the unstructured search problem. Even if we have shown that the Oracle has T = O(N), we will need to at least attempt to show that there are no trivial solutions to restore the time complexity of Oracle to O(1). In the following, we attempt to restore the time complexity of Grover's algorithm to O($\sqrt{N}$) with a SIMD algorithm. This algorithm works for most cases. Because it is not really practical to build a SIMD architecture for a large number of processing units,

this is a solution in theory.

5. An O(1) Parallel Algorithm for Unstructured Search without Repeated Items

In this section, we will introduce an O(1) parallel algorithm for unstructured search without repeated items.

SIMD [9] [10] [11] [12] [13] stands for Single Instruction, Multiple Data. It is a parallel computing architecture that performs the same operation on multiple data points simultaneously. In SIMD, a single instruction is executed across multiple processing elements, each operating on a different set of data.

We assume the number of processing elements is the same as the number of items in the random list. We further assume that the processing elements are fully connected.

Message passing is a communication paradigm used in parallel computing to enable communication and coordination between different processing units, such as processing elements. In message passing, processes communicate by sending and receiving messages through a communication network or interconnect. Each processing element has its own address, but can exchange data and synchronize with other processes using messages.

Send and Receive Operations [9] [10] [11] [12] [13] are defined as processes that can send messages to each other using "Send" operations and receive messages using "Receive" operations. A process initiates a send operation to transmit a message to another processing element, specifying the destination processing element and the data to be sent. The destination process then initiates a receive operation to receive the message. The pseudo codes for Send and Receive are:

Send (destination-address, message),

Receive ().

Each processing element is labeled by its address:

$$P = \{0,1,2,3,\cdots,N-1\} \tag{12}$$

i.e. processing element, P_0 or $P[0]$, has an address of 0, … .

We will assign a variable (or array), m, to each processing element, for receiving messages with the following initialization:

$$\begin{aligned} P_0 \cdot m &= -1, \\ P_1 \cdot m &= -1, \\ &\vdots \end{aligned}$$

Given an unsorted list,

$$M = \{m_0, m_1, \cdots, m_{L-1}\},$$

And a SIMD array in Equation (12), one can simply match each item in the unsorted list to its corresponding processing element, *i.e.* match m_0 with P_0, m_1 with P_1, … . By assumption, each processing element will send and receive once. The element, $P[i] = P_i$, $i = 0, 1, 2, 3, \ldots$, will send a message:

P_i.Send(m_i,i);

and receive a message from some other element:

$P_i.m$ = Receive();

By assumption, there is no repeated item in the unsorted list, so receiving once is enough. We will use P_i and $P[i]$ interchangeably. We will use an example to show the algorithm.

Example. Finding 5 and 3 from a random list: {1, 3, 7, 2, 4, 6, 0}.

Unsorted list: $M = \{1,3,7,2,4,6,0\}$

Processing element: $P \cdot m = \{-1,-1,-1,-1,-1,-1,-1,-1\}$

Step 0: Input

- $P[0]$ matches $M[0] = 1$,
- $P[1]$ matches $M[1] = 3$,
- …

Step 1. Send

- $P[0]$ sends its address, 0, to element, $M[0] = 1$, *i.e.* sends to $P[1]$
- $P[1]$ sends its address, 1, to element, $M[1] = 3$, *i.e.* sends to $P[3]$
- …

Step 2. Receive

- $P[0]\cdot m$ receives 7,
- $P[1]\cdot m$ receives 0,
- …

Now: $P \cdot m = \{7,0,3,1,5,-1,6,2\}$

Step 3. Find 5 and 6.

- To find item 5, go to $P[5]$ and get $P[5]\cdot m$ = -1, *i.e.*, not found.
- To find item 3, go to $P[3]$ and get $P[3]\cdot m$ = 1, *i.e.*, the item found at position 1.

The SIMD algorithm is:

Input data:

Match

$$M = \{m_0, m_1, \cdots, m_{L-1}\};$$

With

$$P = \{0,1,2,3,\cdots,N-1\}$$

Element i:

Send: P_i.Send(m_i,i);

Receive: $P_i.m$ = Receive();

Search:

let P_k address be the target, return $P_k.m$.

The time complexity of this algorithm is O(1) because of SIMD parallelism. If we preprocess the data with the above SIMD hardware, Grover's algorithm will still have T = O($\sqrt{N}$). One can argue that (1) this algorithm has already solved the search problem; (2) the proposed SIMD architecture is hard to build; and (3) the proposed SIMD architecture is impossible for a large list; however, Grover's

algorithm is a category of algorithms that has gone beyond the unstructured search [7] [8].

6. A Parallel Algorithm for Unstructured Search with Repeated Items

For a random list with repeated items, the parallel algorithm proposed in the last section will need to be modified. Assuming the maximum number of repeated items is K, the single receiving step, $P_i.m$ = Receive(), will be replaced by

```
k = 0;
while ( (P_i.m[k] = Receive())    != -1 )
    k++;
```

This algorithm will solve the second problem in Section 5, the normalization problem for repeated items. The time complexity is $T = O(K)$. In the worst scenarios where K is order of N, for example, $K = N/2$ or $K = N/3$, time complexity is $T = O(N)$.

7. Conclusion

In this paper, we have shown that only under special cases where the outcome is predictable and Grover's algorithm is unnecessary, the time complexity of the Oracle Phase is O(1) and the time complexity of Grover's algorithm is $O(\sqrt{N})$. In general, the time complexity of the Oracle Phase is $O(N)$ and the time complexity of Grover's algorithm for the unstructured search problems is $O(N)$.

We attempt to restore the time complexity of Grover's algorithm to $O(\sqrt{N})$ with a non-trivial SIMD architecture, which is an O(1) parallel algorithm for unstructured search without repeated items. This SIMD algorithm will work for most cases. In the worst-case scenarios where the number of repeated items is $O(N)$, the time complexity of the Oracle Phase is still $O(N)$.

Acknowledgements

I would like to thank Gina Porter for proof reading this paper.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Nielsen M.A. and Chuang, I.L. (2010) Quantum Computation and Quantum Information. Cambridge University Press, Cambridge.

[2] Rieffel, E. and Polak, W. (2011) Quantum Computing: A Gentle Introduction. The MIT Press, Cambridge.

[3] Johnston, E.R., Harrigan, N. and Gimeno-Segovia, M. (2019) Programming Quantum Computers: Essential Algorithms and Code Samples. O'Reilly Media, Sebastopol.

[4] Hidary, J.D. (2019) Quantum Computing: An Applied Approach. Springer, Cham.

https://doi.org/10.1007/978-3-030-23922-0

[5] Aaronson, S. (2013) Quantum Computing Since Democritus. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9780511979309

[6] Grover, L.K. (1996) A Fast Quantum Mechanical Algorithm for Database Search. *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (*STOC*'96), New York, 22-24 May 1996, 212-219. https://doi.org/10.1145/237814.237866

[7] Berti, A. (2022) Behind Oracles: Grover's Algorithm. https://towardsdatascience.com/behind-oracles-grovers-algorithm-amplitude-amplification-46b928b46f1e

[8] Shirgure, S. (2024) Solving the Subset Sum Problem on a Quantum Computer. https://sumeetshirgure.github.io/assets/pdfs/presentations/ee_520.pdf

[9] Grama, A. and Karypis, G. (2003) Introduction to Parallel Computing. Addison-Wesley, Boston.

[10] Wilkinson, B. and Allen, M. (1999) Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers. Prentice Hall, Upper Saddle River.

[11] Hennessy, J.L. and Patterson, D.A. (2017) Computer Architecture: A Quantitative Approach. Morgan Kaufmann, Burlington.

[12] Reinders, J. and Jeffers, J. (2014) High Performance Parallelism Pearls: Multicore and Many-Core Programming Approaches. Morgan Kaufmann, Burlington.

[13] Quinn, M.J. (2003) Parallel Programming in C with MPI and OpenMP. McGraw-Hill Education, New York.