Implementation of Grover's Search Algorithm

A MINI PROJECT REPORT

18CSE310J - Quantum Computation

Submitted by

Ayush Garg **[RA2111003011823]**

Sashikanta Mohanty **[RA2111003011830]**

Anurag Bose **[RA2111003011849]**

Under the Guidance of

Dr. Raghavendra V

Assistant Professor, Department of Computing Technologies

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING
of
FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

JULY 2024

COLLEGE OF ENGINEERING & TECHNOLOGY
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
S.R.M. NAGAR, KATTANKULATHUR – 603 203
Chengalpattu District

BONAFIDE CERTIFICATE

Registration number **RA211103011823, RA2111003011830, RA2111003011849** certified to be the bonafide work done by **Ayush Garg, Sashikanta Mohanty, Anurag Bose** of IV Year/VII Sem B.Tech Degree Course in the **18CSE310J - Quantum Computation** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**, Kattankulathur during the academic year 2024 – 2025.

Faculty Incharge
Dr. Raghavendra V
Assistant Professor
Department of Computing Technologies
SRMIST – KTR.

Head of the Department
Dr. M. Pushpalatha
Professor and Head
Department of Computing Technologies
SRMIST – KTR.

ABSTRACT

This project delves into the implementation and analysis of Grover's Search Algorithm, a pivotal quantum algorithm that revolutionises the efficiency of searching through unsorted databases by reducing the search complexity from $O(N)$ in classical algorithms to $O(\sqrt{N})$ in quantum computing. The project provides a thorough examination of the theoretical foundations of Grover's Algorithm, elucidating its mathematical framework and operational principles. It includes a step-by-step guide to implementing the algorithm using quantum circuits and demonstrates its application on quantum computing platforms such as IBM Quantum Experience and Quantum Inspire. Performance metrics are rigorously analysed, showcasing the algorithm's accuracy and speedup in various qubit configurations, while also addressing the current limitations and challenges in quantum hardware. By presenting practical results alongside theoretical insights, the project highlights the profound implications of Grover's Algorithm in fields like cryptography, database search, and optimization problems. Ultimately, this project underscores the transformative potential of quantum computing technologies in enhancing computational efficiency and solving complex problems that are infeasible for classical computers.Additionally, the project explores circuit optimization techniques, such as gate simplification and parallelization, to improve the algorithm's performance further. It also delves into practical applications, including solving Boolean satisfiability problems (SAT), where Grover's Algorithm shows promise in outperforming classical solvers. Through detailed analysis and comprehensive implementation, this project not only demonstrates the current capabilities of quantum search algorithms but also paves the way for future advancements in quantum computing.

TABLE OF CONTENT

CHAPTER 1
INTRODUCTION

Grover's Algorithm, first proposed by Lov Grover in 1996, is one of the most significant quantum algorithms, second only to Shor's algorithm in terms of its impact on the field of quantum computing. The algorithm provides a quadratic speedup for unstructured search problems, which is a substantial improvement over classical algorithms. This paper delves into the intricacies of Grover's Algorithm, its implementation, and its implications for solving various computational problems. Quantum computing leverages the principles of quantum mechanics to process information in fundamentally different ways compared to classical computing. Classical computers use bits as the basic unit of information, which can be either 0 or 1. In contrast, quantum computers use quantum bits or qubits, which can exist in superpositions of states. This allows quantum computers to perform many calculations simultaneously, providing a potential exponential speedup for certain types of problems. The development of quantum computing has been driven by the limitations of classical computers. As the miniaturisation of transistors approaches physical limits, quantum computing offers a pathway to overcome these barriers by exploiting the unique properties of quantum mechanics. Key concepts in quantum computing include superposition, entanglement, and quantum interference, all of which are harnessed in Grover's Algorithm to achieve its speedup.

Grover's Algorithm addresses the search problem, where the goal is to find a specific item in an unsorted database. Classical algorithms typically require $O(N)$ operations to find the target item, where N is the number of items in the database. Grover's Algorithm, however, can find the target item in $O(\sqrt{N})$ operations, providing a quadratic speedup. The algorithm operates by iteratively applying a sequence of quantum operations to amplify the probability amplitude of the target item. The main steps of Grover's Algorithm are as follows:

1. Initialization: All qubits are initialised to the state $|0>$.
2. Superposition: A Hadamard gate is applied to each qubit.
3. Oracle: An oracle function is applied, which inverts the amplitude of the target state.
4. Diffusion Operator: The Grover diffusion operator is applied to amplify the probability amplitude of the target state.
5. Iteration: Steps 3 and 4 are repeated approximately $\sqrt{N}$ times.
6. Measurement: The state of the qubits is measured, and the target item is obtained with high probability.

In our project, we implemented Grover's Algorithm using the cQASM quantum programming language, complemented by Python for constructing and optimizing the quantum circuits. cQASM, a low-level quantum assembly language, provides the flexibility needed for precise control over quantum operations, making it suitable for implementing complex algorithms like Grover's. The implementation began with initializing the qubits and creating the superposition using Hadamard gates. The oracle function was designed to flip the phase of the target state, and the diffusion operator was applied to enhance the target state's amplitude. The circuit was iterated the optimal number of times, and various optimizations were applied to reduce the gate count and improve execution efficiency. This included replacing larger groups of gates with smaller equivalent groups and parallelizing gate operations where possible. We extended Grover's Algorithm to search a database of $N = 2^n$ entries for multiple search terms and implemented a solver for arbitrary SAT problems to evaluate the algorithm's performance. Our results indicate that Grover's Algorithm can outperform optimal classical k-SAT solvers under certain conditions, suggesting its potential for solving complex real-world problems efficiently. However, when considering practical factors such as noise, decoherence, and error rates in current quantum hardware, the applicability of Grover's Algorithm must be evaluated cautiously. While theoretical performance shows promise, real-world implementation challenges remain a significant hurdle.

Grover's Algorithm represents a powerful application of quantum computing, demonstrating a clear advantage over classical search algorithms. By providing a quadratic speedup, it opens up new possibilities for solving large-scale search problems efficiently. Our implementation in cQASM showcases the practical aspects of deploying Grover's Algorithm on quantum hardware, highlighting both its potential and the challenges that need to be addressed. As quantum computing technology continues to evolve, Grover's Algorithm will likely play a crucial role in realizing the full potential of quantum search and optimization. This project contributes to the ongoing exploration of quantum algorithms and their applications, paving the way for future advancements in quantum computing.

CHAPTER 2
BACKGROUND

A. Quantum Computing

Quantum computing represents a fundamental shift from classical computing paradigms, leveraging the principles of quantum mechanics to perform computations. At the heart of quantum computing is the quantum bit or qubit, which, unlike a classical bit that is either 0 or 1, can exist in a superposition of both states simultaneously. This property is encapsulated in the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers that describe the probability amplitudes of the qubit's states. This superposition enables quantum computers to process a vast amount of possibilities concurrently.

Furthermore, qubits can be entangled, a phenomenon where the state of one qubit is dependent on the state of another, regardless of the distance between them. This entanglement, coupled with superposition, allows quantum computers to solve certain problems more efficiently than classical computers. For instance, while a classical computer would need to check each possible solution sequentially, a quantum computer can explore many solutions simultaneously, thanks to its ability to perform multiple computations in parallel.

Quantum gates manipulate qubits through unitary transformations, altering the probability amplitudes of their states. These gates, analogous to classical logic gates, form the building blocks of quantum circuits. Quantum algorithms, which are sequences of these gates, can solve specific problems faster than their classical counterparts. Notable quantum algorithms include Shor's algorithm for factoring large numbers and Grover's algorithm for searching unsorted databases.

B. Grover’s Algorithm

Grover’s algorithm, proposed by Lov Grover in 1996, is a quantum search algorithm that provides a quadratic speedup over classical search algorithms. Given an unsorted database of N elements, a classical algorithm would require $O(N)$ operations to find a target element. In contrast, Grover’s algorithm can locate the target element in $O(\sqrt{N})$ operations, making it significantly faster for large datasets.

The algorithm's effectiveness stems from its ability to amplify the probability amplitude of the correct solution through a series of iterations. Here is a brief outline of the algorithm:

1. Initialization: The algorithm begins by preparing a superposition of all possible states. For an n-qubit system, this superposition is created by applying the Hadamard gate to each qubit, resulting in a uniform distribution of all 2^n states.

2. Oracle Application: An oracle, a quantum subroutine designed to recognize the correct solution, is applied. This oracle flips the phase of the target state, effectively marking it.

3. Amplitude Amplification: The algorithm then applies the Grover diffusion operator, which inverts the amplitudes of the states around the average amplitude. This step increases the probability amplitude of the target state while decreasing the amplitudes of the incorrect states.

4. Iteration: Steps 2 and 3 are repeated $O(\sqrt{N})$ times to sufficiently amplify the probability of the correct solution.

5. Measurement: Finally, the quantum state is measured, collapsing the superposition to one of the basis states. With high probability, this state is the target element.

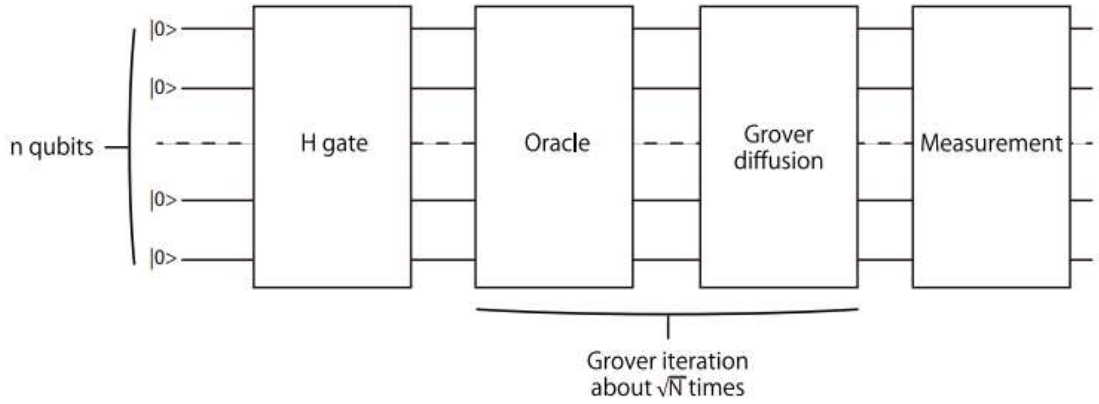

Figure 1: Schematic circuit for the Grover's algorithm

The algorithm's quadratic speedup makes it particularly useful for applications involving large search spaces, such as database search, optimization problems, and cryptographic analysis. It has been implemented and tested on various quantum computing platforms, demonstrating its practical viability and potential for real-world applications.

C. Applications and Implementations

Grover's algorithm has been implemented on several quantum computing platforms, including IBM's quantum computers. These implementations vary in the number of qubits used and the complexity of the circuits. For example, a 2-qubit implementation of Grover's algorithm can identify one of four possible states with high accuracy. More complex implementations, such as 3-qubit and 4-qubit versions, have also been developed, achieving varying degrees of accuracy and efficiency based on the number of iterations and the specific hardware used .

In practice, the accuracy of Grover's algorithm is influenced by several factors, including the fidelity of the quantum gates, the coherence time of the qubits, and the number of iterations performed. Experimental results have shown that increasing the number of iterations improves the accuracy, as the probability amplitude of the correct solution is amplified with each iteration. The development and refinement of Grover's algorithm continues to be an active area of research in quantum computing. Researchers are exploring ways to optimise the algorithm's performance, reduce its resource requirements, and extend its applicability to a broader range of problems. As quantum hardware improves, the practical implementation of Grover's algorithm is expected to become more feasible, opening up new possibilities for solving complex computational problems efficiently . In addition to database searching, Grover's Algorithm has significant implications for various other domains. In cryptography, it poses a potential threat to cryptographic systems by drastically reducing the time required to brute-force search keys. This necessitates the development of quantum-resistant cryptographic algorithms to ensure security in a post-quantum world. Furthermore, Grover's Algorithm can be applied to optimization problems, where it helps find optimal solutions by efficiently searching through large solution spaces.

Advancements in quantum hardware, such as error-corrected qubits and improved gate fidelities, will enhance the practical viability of Grover's Algorithm. As quantum computers scale up and become more accessible, Grover's Algorithm will likely become a cornerstone in the toolkit of quantum algorithms, driving innovation across various scientific and industrial fields. Ongoing research is also exploring hybrid approaches that combine classical and quantum computing techniques. These hybrid algorithms aim to leverage the strengths of both paradigms, potentially offering even greater speedups and practical solutions to complex problems.

CHAPTER 3
IMPLEMENTATION OF GROVER'S ALGORITHM

We implemented Grover's algorithm (GA) using the cQASM quantum programming language, in combination with Python. More specifically, the cQASM code was constructed using Python code. A GitHub link to the raw Python code can be found in the Appendix. To get Grover's algorithm to work, one first needs to initialize the qubits to an even superposition of all inputs using Hadamard gates, then apply an oracle operation, and finally Grover's diffusion operator. This circuit is then iterated over r = $\pi/4\sqrt{N}$ times. First, we implemented a trivial oracle which flips the phase for a specific hard-coded value and thus returns this value after running the algorithm. The circuit was then generalized for n qubits. As the next extension, we investigated finding multiple entries in the search space which satisfy the condition in the oracle. This means n search bits, and therefore a database of size 2n, with M search terms (M < 2^n), To accomplish this, the oracle was repeated for different search terms. The order the circuit is then: oracle 1- oracle 2-...- oracle M diffusion operator. This circuit is iterated over r times, with r = $\pi/4\sqrt{N/M}$.

3.1 Circuit Optimization

Next, we optimized the circuit by replacing larger groups of gates with smaller equivalent groups where possible: XX and HH can simply be replaced by I, HXH = Z, and ZX = iY, et cetera. These replacements reduce the overall gate count, thereby minimizing potential sources of error and decoherence. Additionally, we developed an algorithm to parallelize the circuit by combining gate calls that acted on different qubits, enabling them to run simultaneously. This parallelization significantly reduces the execution time of the quantum circuit by exploiting the inherent parallelism in quantum operations. The cQASM produced by these optimizations was also run through a "clean-up" stage that made the resulting code shorter and more readable, enhancing both the performance and the maintainability of the quantum programs. The clean-up process involved removing redundant gates, simplifying gate sequences, and reordering operations to achieve a more efficient execution flow. The combined effect of these optimizations is discussed in Section 4, specifically Figure 5, where we present a comparative analysis of circuit performance before and after optimization.

These improvements not only demonstrate the potential for significant efficiency gains in quantum computing but also provide a framework for future developments in quantum circuit design and optimization, highlighting the importance of both theoretical and practical advancements in this rapidly evolving field.

3.2 CNOT implementation

We discovered that n-bit CNOT gates were essential to extend the circuit to an arbitrary size, a necessity for scalable quantum algorithms. To achieve this, we developed a method to express any n-bit CNOT gate using Toffoli gates and n - 3 ancillary qubits. These ancillary qubits do not interact with the search algorithm itself but serve to store intermediate data, facilitating the construction of complex quantum operations. An example of this circuit is shown in Figure 2, derived from a similar circuit in Nielsen and Chuang [6]. This approach effectively modularizes the implementation, enabling the construction of larger circuits by breaking down complex operations into manageable components.

However, implementing Toffoli gates physically presents significant challenges due to their complexity and the high fidelity required for accurate quantum operations. Additionally, using a large number of ancillary qubits is often impractical, as it consumes valuable qubit resources and increases the potential for error. To address these issues, we explored different circuit representations based on the work by Barenco et al. [1], which offered alternative methods for implementing n-bit CNOT gates.

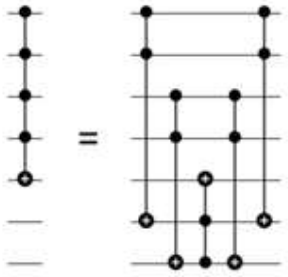

Figure 2: Example of a n-bit CNOT, which uses n 3 ancillary qubits.

One particular representation we adopted uses no ancillary qubits, which is advantageous in reducing resource overhead. However, this version has the property that the number of gates required for an n-bit CNOT scales exponentially with 2^n .

This implementation utilizes a Gray code sequence of operations, resulting in the benefit that for each rotation, only one qubit needs to be targeted by a CNOT gate instead of multiple. This significantly reduces the complexity of the operation at each step, making the circuit more efficient and easier to implement on current quantum hardware. We illustrate this circuit in Figure 3.

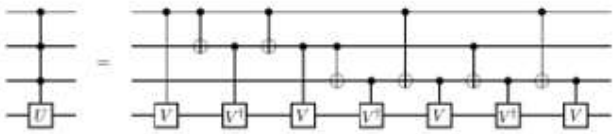

Figure 3: Example of a n-bit controlled gate, without the use of ancillary qubits.

Furthermore, the scaling properties of these implementations were analyzed and are presented in Figure 5. The comparison highlights the trade-offs between different approaches, demonstrating how our optimizations balance the need for fewer ancillary qubits with the increased gate count required by the Gray code method. These insights are crucial for optimizing quantum circuits, especially as we move towards larger and more complex quantum computations.

In practical quantum computing, optimizing the balance between gate count and ancillary qubit usage is critical for maximizing performance and minimizing error rates. Our work shows that while it is possible to construct n-bit CNOT gates with minimal ancillary qubits, the choice of implementation depends on the specific requirements of the quantum algorithm and the limitations of the available quantum hardware. As quantum technology continues to evolve, these optimization strategies will play a vital role in the development of efficient and scalable quantum circuits. Our findings contribute to the broader effort of making quantum computing more practical and accessible, paving the way for more complex quantum algorithms and applications in the future.

3.3 Searching an unstructured database

When comparing the number of operations needed to search an unstructured database, naively comparing Grover's Algorithm (GA) and the classical algorithm shows that the classical approach requires $O(N)$ operations, where N is the number of items in the database, while Grover's Algorithm requires $O(\sqrt{N})$. However, this straightforward comparison does not consider the additional complexity involved in quantum search. When searching an unstructured database, the data must first be encoded into a quantum state, a process that introduces significant overhead.

This initial encoding step adds an $O(N)$ complexity to the overall process, making the total number of operations for Grover's Algorithm $O(N + \sqrt{N})$, not simply $O(\sqrt{N})$ [8].

This added complexity implies that for large databases, the classical search algorithm will always outperform Grover's Algorithm unless the database is amenable to an exceptionally efficient encoding scheme. Efficient encoding of classical data into a quantum state remains one of the significant challenges in practical quantum computing. The process of encoding is non-trivial and often involves complex quantum circuits that can negate the theoretical speedup offered by Grover's Algorithm.

Moreover, the practical implementation of Grover's Algorithm is further constrained by the limitations of current quantum hardware. Factors such as qubit coherence time, gate fidelity, and error rates play a crucial role in the performance of the algorithm. These hardware limitations can significantly impact the actual runtime and reliability of Grover's Algorithm, making it less competitive compared to classical algorithms in real-world applications.

In addition to the encoding overhead, there are also considerations related to the decoherence and noise in quantum systems. As the size of the database increases, the quantum circuit becomes more complex, requiring more gates and more qubits. This increases the susceptibility of the system to errors, necessitating sophisticated error correction techniques. Error correction, in turn, requires additional qubits and operations, further increasing the resource requirements and complexity of the quantum search process.

Furthermore, the initialization of the quantum state, the repeated application of the oracle and diffusion operators, and the final measurement step all contribute to the total computational cost. Each of these steps involves a series of quantum gates that need to be executed with high precision to maintain the integrity of the quantum state. The cumulative effect of these operations can diminish the theoretical advantages of Grover's Algorithm, especially when compared to the simplicity and robustness of classical search algorithms.

3.4 SAT Problems

The SAT problem, or Boolean satisfiability problem, is an example of a problem that does not require complex quantum state encoding. It involves a Boolean expression, such as $\neg$ (($x_1 \vee (x_2 \wedge x_3)$) , where the goal is to find all combinations of Boolean values for x_1, x_2, and x_3 that satisfy the expression, making it TRUE. The SAT problem is a practical example where Grover's Algorithm (GA) has the potential to outperform any classical solver [5]. Before building a quantum circuit to solve the SAT problem, we utilize a Python module developed by S. Krämer [4] to parse the Boolean expression and simplify it according to Boolean logic rules. This preprocessing step ensures that the Boolean expression is in its most simplified form, reducing the complexity of the subsequent quantum operations.

The core of solving the SAT problem using GA lies in generating a suitable oracle that matches the Boolean expression. To this end, we developed two distinct algorithms for constructing the oracle.

The first algorithm iteratively processes the Boolean expression, applying the Boolean gates in the same structure as the expression. It uses ancillary qubits to store the results of individual gates. An optimization within this approach is to store previously calculated sub-expressions: if a sub-expression reappears later in the computation, the algorithm refers back to the ancillary qubit where the original result is stored. This method limits the number of ancillary qubits required but can still become resource-intensive as the complexity of the Boolean expression increases.

The second algorithm structures the Boolean expression as a binary tree and recursively evaluates each branch. Each sub-tree stores its result on a single ancillary qubit line and ensures that all other ancillaries used in its computation are reset to 0 after their results are stored, making them available for reuse by other parts of the tree. This approach significantly reduces the number of qubits required, as it minimizes the ancillary qubit overhead. However, it increases the number of gate operations needed, as the resetting and reuse of qubits add additional steps to the computation.

The performance of both algorithms is analyzed in Section 4.3. We compare the trade-offs between the two approaches, particularly focusing on the balance between qubit usage and gate operations. Our analysis shows that while the recursive binary tree method is more qubit-efficient, the iterative approach may be preferable in scenarios where gate operation costs are less critical, or where qubit resources are ample.

Example circuits for both algorithms are provided in Appendix B, demonstrating the practical implementation of these methods. These circuits illustrate how the algorithms translate Boolean expressions into quantum operations, showcasing the steps involved in constructing the oracle and performing the Grover search.

CHAPTER 4
RESULTS

4.1 Metric

To compare the different implementations of GA a performance metric is needed. Although Quantum Inspire returns a 'runtime' value, this isn't a useful metric, because it greatly relies on the speed of the simulator. As can be seen in Figure 4, the y-axis has a logarithmic scale, but the trend is still at least quadratic. Hence, the scaling of performance is worse than exponential, and dominated by the number of qubits, regardless of the underlying program. There is also a large difference between the minimum, average and maximum runtime, indicating that the runtime is highly variable.

Instead, the number of lines of executed cQASM is used to compare the performance of the different algorithms. The reasoning for this is that all operations in one line may be executed in parallel, and the assumption is made that all one and two qubit gates take approximately the same amount of time to execute. Note that we specifically replace Toffoli gates by a combination of one and two qubit gates to stay in line with this assumption.

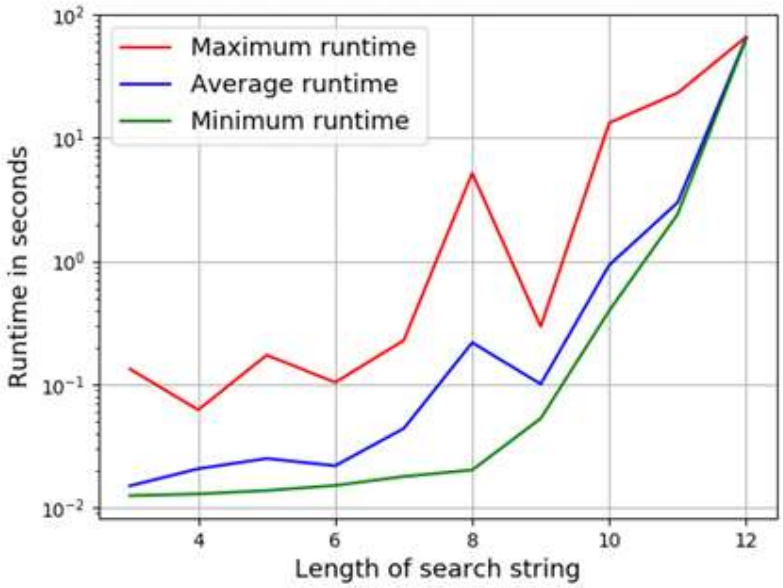

Figure 3: The runtime of GA search for different lengths of search string. (Averaged: 10 shots per data point)

3.2 Analysis of Grover's search

In Figure 4, implementations of n-bit CNOTs are compared in terms of circuit length. A trade-off can be seen between ancillary qubits and speed. A simple replacement for an n bit CNOT uses n - 3 ancillary qubits, but is exponentially faster compared to a method which does not use ancillary qubits. Curve fitting suggests that for a search string of length l, The number of quantum operations scales with $O(1.39^l)$ when using 3 ancillary bits, and with $O(1.73^l)$ when using no ancillary bits. Depending on whether circuit size or qubit count is the limiting physical factor, either version could be adopted.

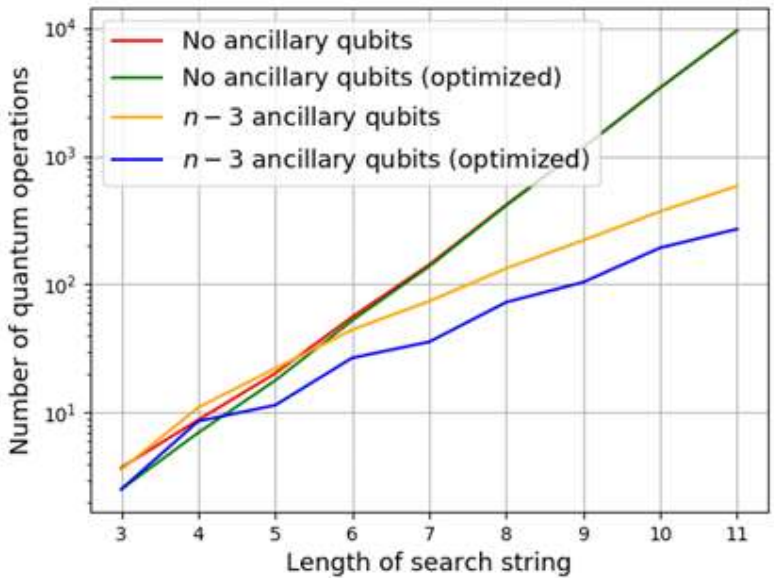

Figure 5: Number of quantum operations as a function of search string size. Compares the n 3 ancillary bit CNOT and the no ancillary bits version, and shows the impact of the optimizer. (Averaged: 15 shots per data point)

We also investigate the performance of searching for M elements simultaneously. The naive method of doing this is to run GA separately for each element: one oracle and one diffusion operator, executed r = $\pi/4\sqrt{N}$ times. This scales linearly with M (see Figure 9). The optimal method expands only the oracle as more search terms are added. In addition, the entire circuit is looped over in one go, and the required number of Grover iterations decreases with a factor of 1M. As can be seen in Figure 9, this results in the optimal version, for which the number of quantum operations against the number of search elements has a better trend.

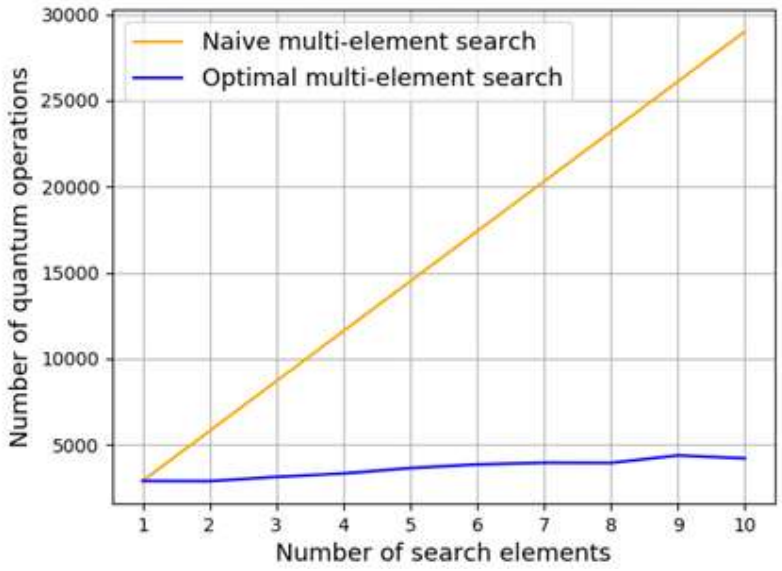

Figure 6: Searching for multiple 10-character long elements in a database of size 210, naively vs optimally. (Averaged: 15 shots per data point)

4.3 Analysis of SAT performance

To examine performance, we wrote a script to randomly generate k-SAT problems with 4 clauses of size k, and k input variables. This expression is passed to both aforementioned SAT solver algorithms. In Figure 6, operation count and qubit count is plotted against an increasing k. The operation count is compared to two classical algorithms. Firstly, a naive exhaustive search, which scales with $O(2^n)$. Secondly, a fast classical k SAT solver, Schönings algorithm[3], which is a multi-start random walk algorithm with a runtime of $O((2(1 - 1/k))^n)$.

In Figure 6 it can also be seen that the quantum algorithm can outperform the fast classical solver for a sufficiently large k. Although the plot suggests this crossover is near k = 12, this is not an accurate estimate: the quantum algorithm used is relatively naive, and further improvements to performance are possible. For example, a quantum implementation of Schönings algorithm has the potential to achieve a runtime of $O((2(1 - 1/k))^{n/2})$, which is almost a quadratic increase of the fastest known classical algorithm[3]. On the other hand, many assumptions made in constructing this circuit likely do not hold in real life, such as the circuit's topology making some gate applications physically unfeasible.

The bottom plot in Figure 6 shows the total number of qubits required for either quantum circuit. The size of quantum computer required to approach classical performance could be reachable within a short timespan, given that quantum computers with qubit counts in the dozens will soon be more commonplace in research labs. However, if we want to outperform classical algorithms and include error correction, our conclusion falls in line with that of Campbell, Khurana and Montanaro[2]: Grover's Algorithm could substantially outperform the best classical algorithms to solve k-SAT problems. For instance, for a 14-SAT problem GA outperforms classical by a factor of 106. Unfortunately it is not mentioned how long it would take to solve the 14-SAT problem classically. However this advantage disappears when the costs and the number of qubits needed for error correction are included. Their article explains that in order to create a practical 14-SAT Grover circuit around 10^{12} qubits are needed, because a Toffoli gate is needed with a depth of 10^{12}.

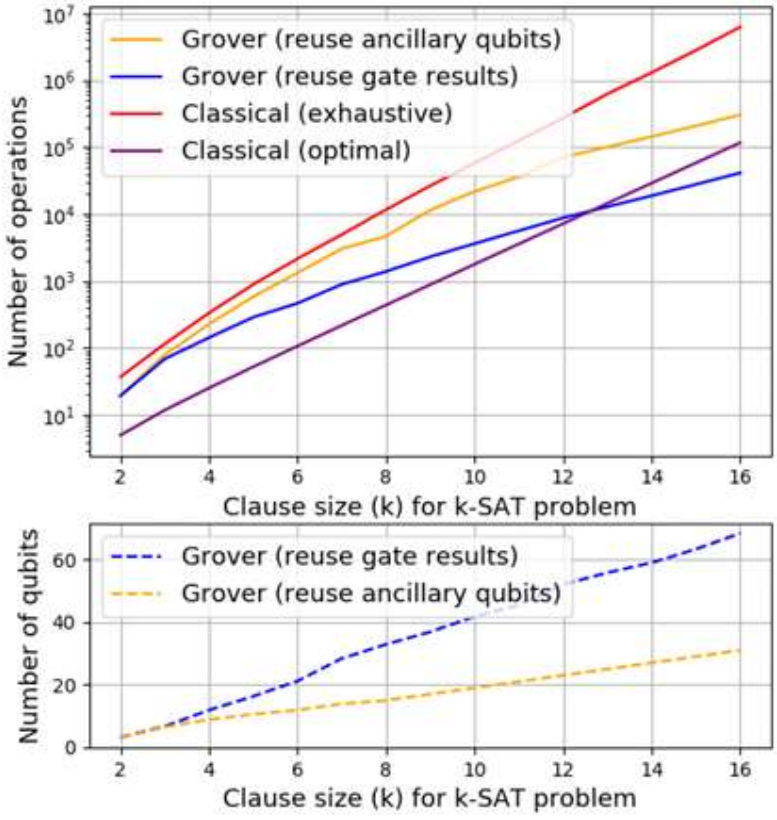

Figure 7: Classical and quantum algorithms are compared in solving k-SAT problems for variable k (top). For both the quantum versions the number of qubits is also shown (bottom). (Averaged: 15 shots per data point)

CHAPTER 5
CONCLUSION

After implementing the basic version of Grover's Algorithm presented on Quantum Inspire, we developed several methods for creating n-bit CNOTs, as well as other optimizations of the circuit. One method for creating an n-bit CNOT uses n + 3 ancillary qubits, while the other method avoids using ancillary qubits, albeit at the expense of a larger circuit. The sizes of these circuits scale with $O(1.39^l)$ and $O(1.73^l)$, respectively.

In multi-element search scenarios, it was observed that repeating only the oracle step significantly enhances performance, to the extent that the number of search elements has a minimal impact on the overall efficiency. This optimization leverages the inherent properties of quantum amplitude amplification, reducing the complexity of the search process.

When comparing Grover's Algorithm with classical algorithms in terms of unstructured search performance, classical algorithms generally outperform Grover's Algorithm due to the initial overhead of encoding the database into a quantum state. This encoding step introduces a lower bound of O(N) on Grover's time complexity, negating some of the algorithm's theoretical advantages in practical applications. Despite this, Grover's Algorithm remains an important proof of concept for demonstrating quantum speedup.

Furthermore, we explored two methods for solving SAT (Satisfiability) problems using Grover's Algorithm. These methods showed that Grover's Algorithm could potentially outperform the best classical k-SAT solvers for values of k larger than 13. However, this advantage is theoretical at this stage. Practical implementation of Grover's Algorithm for SAT problems faces significant hurdles, including the need for extra qubits for error correction, the overhead of swap operations, and the substantial costs associated with realizing a fully functional quantum computer. Consequently, a practical application of Grover's Algorithm in this field is not expected in the near future.

Moreover, the experimental implementations of Grover's Algorithm highlighted several critical factors impacting performance. The accuracy of quantum gates, coherence times of qubits, and the overall fidelity of quantum operations are crucial in determining the success of the algorithm. Error correction techniques, while necessary for maintaining quantum coherence, add substantial complexity to the quantum circuits.

Optimizing Grover's Algorithm also involved addressing the scalability issues associated with increasing qubit counts. As the number of qubits grows, the quantum circuits become more susceptible to noise and decoherence, necessitating more robust error correction methods and more efficient gate operations. Techniques such as qubit recycling and the use of ancillary qubits were explored to mitigate these issues, although they come with their own trade-offs in terms of circuit complexity and resource requirements.

In conclusion, while Grover's Algorithm demonstrates a remarkable theoretical speedup for unstructured search problems, its practical implementation is constrained by current quantum hardware limitations. The need for error correction, the overhead of encoding classical data into quantum states, and the challenges of scaling up quantum systems all contribute to the current impracticality of Grover’s Algorithm for large-scale applications. Nevertheless, ongoing research and advancements in quantum technology hold the promise of overcoming these hurdles, potentially unlocking the full potential of Grover's Algorithm and other quantum algorithms in the future.

REFERENCES

[1] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. Physical review A, 52(5):3457, 1995.

[2] Earl Campbell, Ankur Khurana, and Ashley Montanaro. Applying quantum algorithms to constraint satisfaction problems. Quantum, 3:167, 2019.

[3] Evgeny Dantsin, Vladik Kreinovich, and Alexander Wolpert. On quantum versions of record-breaking algorithms for sat. SIGACT News, 36(4):103–108, December 2005.

[4] S Krämer. Boolean.py python module. https:// github.com/bastikr/boolean.py. Online; accessed 23 January 2019.

[5] Salvatore Mandra, Gian Giacomo Guerreschi, and Alán Aspuru-Guzik. Faster than classical quantum algorithm for dense formulas of exact satisfiability and occupation problems. New Journal of Physics, 18(7):073003, 2016.

[6] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[7] D Voorhoede. Code example: Grover's algorithm. https://www.quantum-inspire.com/kbase/grover-algorithm/. Online; accessed 13 January 2019.

[8] Colin P Williams. Quantum search algorithms in science and engineering. Computing in science & engineering, 3(2):44–51, 2001.

APPENDIX

A Code

The code can be found via the following GitHub link:

https://github.com/Sashi2002/Grovers-Search-Algorithm/blob/main/grover_algorithm_qi.ipynb

B Examples of SAT Circuits

Circuits are listed that solve this Boolean expression: ((A and B and C) or (not(A) and not(B) and not(C)))

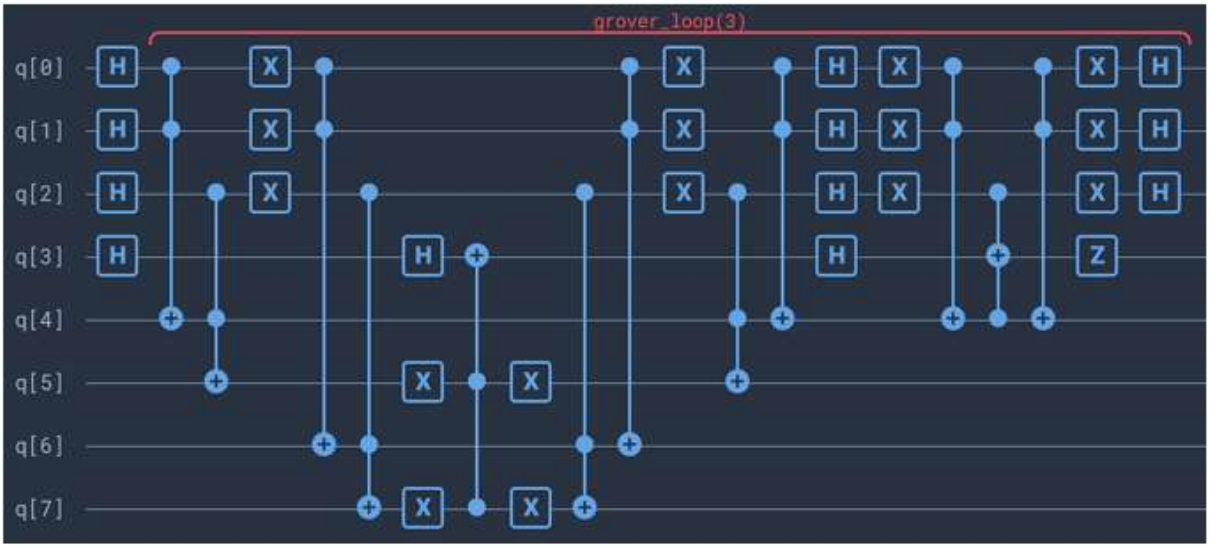

Figure 8: Quantum circuit when reusing gate results (0 are reused in this case)

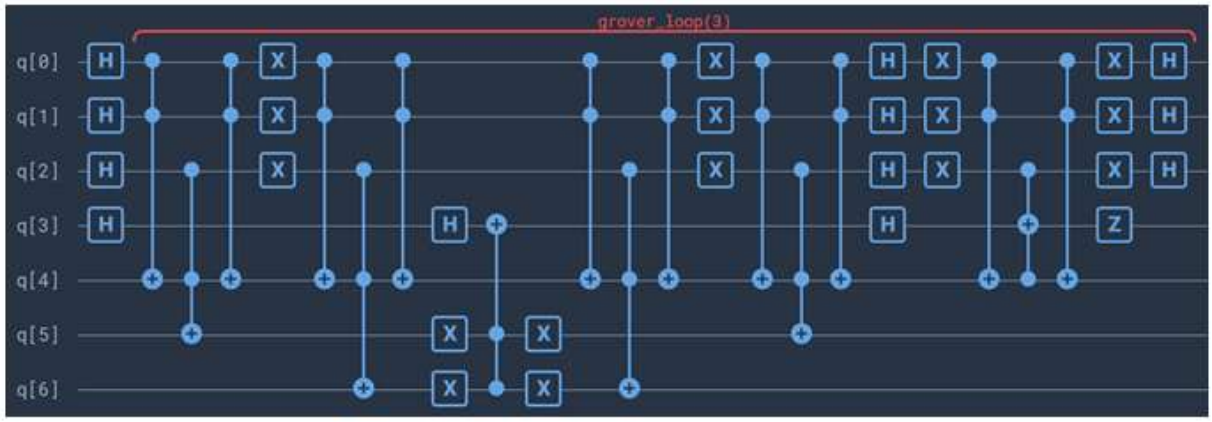

Figure 9: Quantum circuit when reusing ancillary qubits (1 line is reused)