Grover's Algorithm: implementation in cQASM and performance analysis

Quantum Engineering Project

Group 2

David Bakker	**4675932**
Hitesh Dialani	**4648153**
Rembrandt Klazinga	**4674642**
Maurits van der Tol	**4476654**

Quantum Science and Quantum Information Minor TN-Mi-219
Technische Universiteit Delft
Netherlands

January 29, 2020

1 INTRODUCTION

The two most well-known quantum algorithms are Shor's algorithm and Grover's algorithm. The speedup Shor provides over its classical counterpart can be clearly demonstrated, but the performance of Grover's Algorithm compared to classical algorithms is less obvious. In this paper we examine Grover's algorithm in more detail by implementing it in cQASM and extending the algorithm to search a database of $N = 2^n$ entries for M different search terms. Furthermore, we implement a solver for an arbitrary SAT problem, which is used to examine Grover's performance in solving k-SAT problems, a field in which Grover is said to provide large benefits. Our results suggest optimal classical k-SAT solvers can be beaten by Grover circuits in the near future. However, when also considering other real world factors, the applicability of GA becomes questionable.

2 IMPLEMENTATION OF GROVER'S ALGORITHM IN CQASM

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 $r \approx \frac{\pi}{4}\sqrt{N}$ 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 2^n, 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 $r \approx \frac{\pi}{4}\sqrt{N/M}$.

2.1 Circuit optimization

Next, we optimised 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. Additionally, we developed an algorithm to parallelize a circuit: combining gate calls that acted on different qubits and that could therefore run simultaneously. The cQASM produced by these optimizations was also run through a "clean-up" stage that made the resulting code shorter and more readable. The combined effect of these optimizations is discussed in Section 3, specifically Figure 4.

2.2 CNOT implementations

We discovered that n-bit CNOT gates were needed to extend the circuit to an arbitrary size. For this, we found a simple method to express any n-bit CNOT gate using Toffoli gates and $n - 3$ ancillary qubits, which do not interact with the search algorithm itself, but only serve to store intermediate data. An example of this circuit is shown in Figure 1 derived from a similar circuit in Nielsen and Chuang [6].

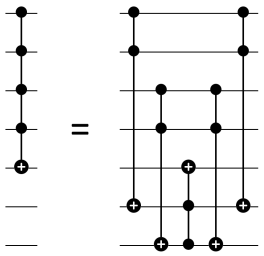

Figure 1: Example of a n-bit CNOT, which uses $n - 3$ ancillary qubits.

In practice, it is difficult to physically implement a Toffoli gate, and it is wasteful to use such a large number of ancillary qubits. Hence, we found different circuit representations in work by Barenco et al. [1], which we used to implement an n-bit CNOT. One particular version uses no ancillary qubits, but has the property that the number of gates required for an n-bit CNOT scales with 2^n. It is constructed using a Gray code sequence of operations, which results in the advantage that for every rotation only one qubit needs to be targetted by a CNOT instead of multiple. We show this circuit in Figure 2. The scaling properties can be seen in Figure 4.

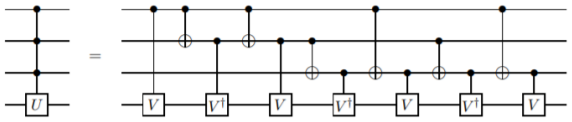

Figure 2: Example of a n-bit controlled gate, without the use of ancillary qubits[1].

2.3 Searching an unstructured database

When comparing the number of operations needed to search an unstructured database, naively comparing GA and the classical algorithm shows that classical requires $O(N)$, while Grover requires $O(\sqrt{N})$. However, when searching an unstructured database, it first needs to be encoded in the quantum state in order to apply GA. This makes the number of operations $O(N + \sqrt{N})$, not $O(\sqrt{N})$ [8]. As a result, the classical search algorithm will always outperform GA in this area, unless the database happens to lend itself to an efficient encoding.

2.4 SAT problems

The SAT problem, or Boolean satisfiability problem, is an example of a problem that needs no such encoding. It

involves a Boolean expression like $\neg(x_1 \vee (x_2 \wedge x_3))$, for which one wants to find all combinations of Boolean values for x_1, x_2 and x_3 such that the whole statement returns TRUE. The SAT problem is a practical example where GA has the potential to outperform any classical solver. [5].

Before building a quantum circuit, we use a Python module developed by S. Krämer [4] to parse a Boolean expression and simplify it according to Boolean logic rules. After that, the focus of solving the SAT problem using GA is generating a suitable oracle matching the Boolean expression: to this end, we developed two algorithms.

The first algorithm runs through the expression and applies the Boolean gates in the same structure, using ancillary qubits to store results of individual gates. An optimisation that somewhat limits ancillary use is to store previously calculated sub-expressions: a later reappearance of such an expression will then refer back to the ancillary where the original result is still stored.

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 line, and ensures all other ancillaries used in its computation are reset to 0, so they can be reused by other parts of the tree. This reduced the number of qubits required, at the cost of more gate operations. The performance of both algorithms is analyzed in Section 3.3. Example circuits for both algorithms can be seen in Appendix B.

3 RESULTS

3.1 Metric

To compare the different implementations of GA a performance metric is needed. Although Quantum Inspire returns a 'runtime' value, this isn't an useful metric, because it greatly relies on the speed of the simulator. As can be seen in Figure 3, 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.

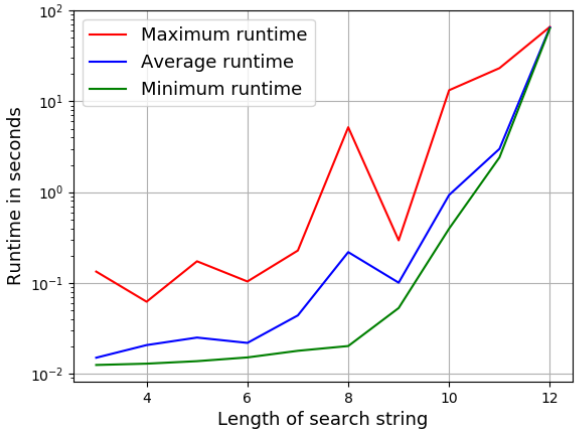

Figure 3: The runtime of GA search for different lengths of search string. (Averaged: 10 shots per data point)

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 $\mathcal{O}(1.39^l)$ when using $n-3$ ancillary bits, and with $\mathcal{O}(1.73^l)$ when using no ancillary bits. Depending on whether circuit size or qubit count is the limiting physical factor, either version could be adopted.

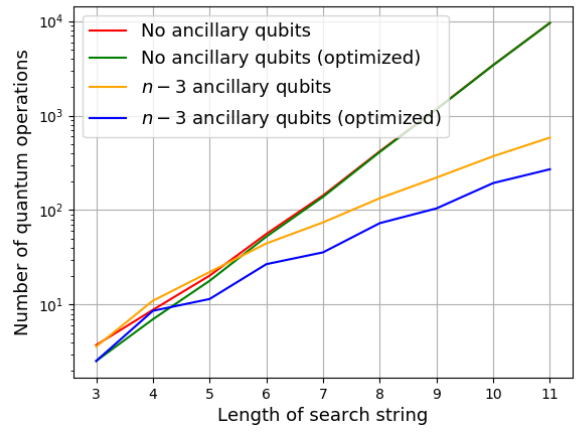

Figure 4: Number of quantum operations as a function of search string size. Compares the $n-3$ ancillary bit CNOT and the no ancillary bits version, and shows the impact of the optimizer. (Averaged: 15 shots per data point)

We also investigate the performance of searcing 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 $r \approx \frac{\pi}{4}\sqrt{N}$ times. This scales linearly with M (see Figure 8). 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 $\sqrt{1/M}$. As can be seen in Figure 8, this results in the optimal version, for which the number of quantum operations against the number of search elements has a better trend.

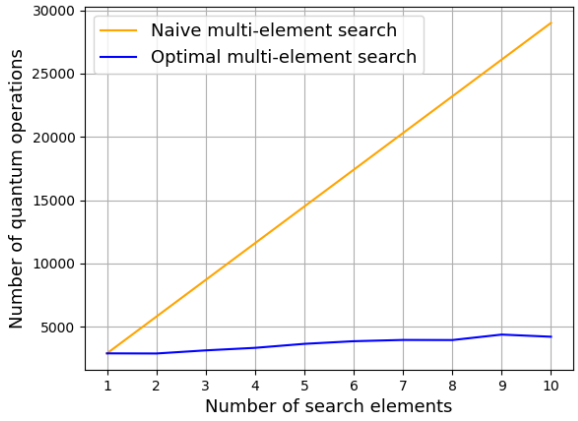

Figure 5: Searching for multiple 10-character long elements in a database of size 2^{10}, naively vs optimally. (Averaged: 15 shots per data point)

3.3 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 $\mathcal{O}(2^n)$. Secondly, a fast classical k-SAT solver, Schönings algorithm[3], which is a multi-start random walk algorithm with a runtime of $\mathcal{O}((2(1-\frac{1}{k}))^n)$.

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 $\mathcal{O}((2(1-\frac{1}{k}))^{\frac{n}{2}})$, 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 10^6. 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}.

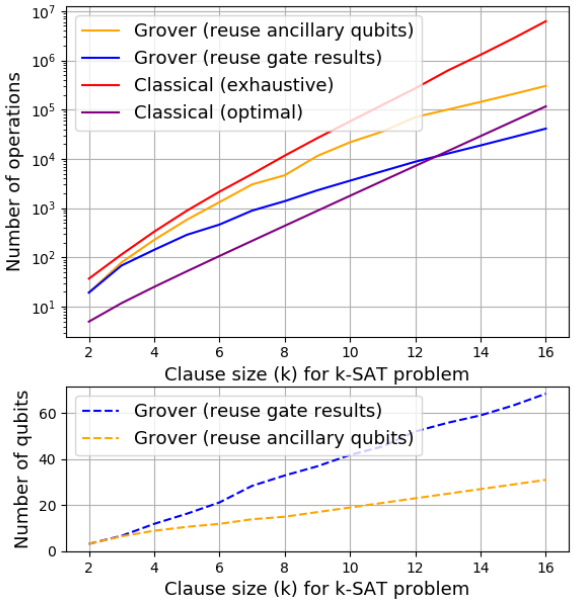

Figure 6: Classical and quantum algorithms are compared in solving k-SAT problems for variable k (top). For both the quantum versions the number of qubits is also shown (bottom). (Averaged: 15 shots per data point)

4 CONCLUSION

After implementing the basic version of Grover's algorithm presented on Quantum Inspire [7], we developed several ways of creating n-bit CNOTs, as well as other optimizations of the circuit. One way of creating an n-bit CNOT uses $n-3$ ancillary qubits, while the other one uses no ancillary qubits, at the expense of a larger circuit: the circuit sizes scale with $\mathcal{O}(1.39^l)$ and $\mathcal{O}(1.73^l)$, respectively.

In multi-element search, it was found that repeating only the oracle improves performance to such a degree that the number of search elements has only a minimal impact.

When comparing GA with a classical algorithm in unstructured search performance, classical will outperform Grover due to the fact that the database first needs to be encoded into a quantum state. This puts a lower bound of $O(N)$ on Grover's time complexity.

Lastly, two methods of solving SAT problems were constructed. It can be concluded that Grover's Algorithm has the potential to be faster for k-SAT problems with k

values larger than 13, when compared to the best classical k-SAT solvers. However, when taking the extra qubits for error correction into account, as well as swap operations and the costs of realising a quantum computer, a practical application for Grover in this field is not realistic in the near 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.

A Code

The code can be found via the following GitHub link:
`https://github.com/FluffyToaster/QuantumGrover/tree/c56166ab7bfdfe740493328927cd166cb465b3d9`

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)))

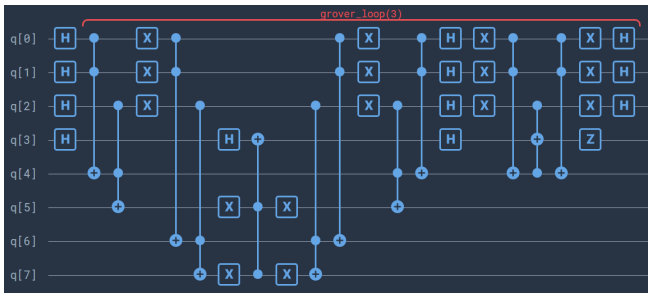

Figure 7: Quantum circuit when reusing gate results (0 are reused in this case)

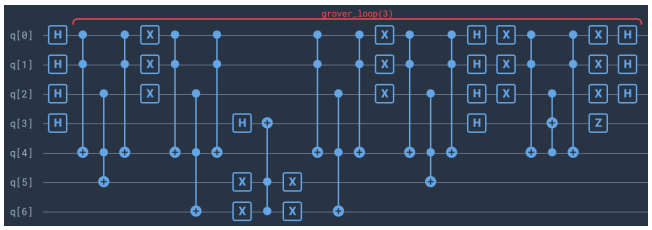

Figure 8: Quantum circuit when reusing ancillary qubits (1 line is reused)