CS 269Q Project Final Report

Nathan Dalal, Hubert Teo
{nathanhd, hteo}@stanford.edu

June 8, 2019

1 Abstract

We present an application of Grover's algorithm to the 3-SAT problem, and verify the resulting square-root speedup over the classical search. We achieve this with reductions from CNFs to phase-flip oracles with depth linear in the number of gates and boolean variables. These reductions were implemented in pyQuil. To optimize the oracle circuits, we evaluated different circuit constructions: one based on Fredkin gates and four based on CCNOT. In our comparison of these different circuit construction and compilation methods, we found that CCNOT-based solutions require less qubits and gates. This is because they admit several tricks to reduce the total number of qubits and gates required.

2 Background

2.1 Boolean satisfiability

The 3-SAT problem is an NP-complete problem that admits a naive $\mathcal{O}(2^n)$-time classical solution via brute-force. A direct application of Grover's algorithm to the search space of Boolean variable assignments produces a $\mathcal{O}(2^{n/2})$-time quantum solution, which is a square-root speedup [1].

We set out to implement this square-root speedup, which leverages the general result of Grover's search algorithm [2]. Note that a faster $\mathcal{O}(1.153\ldots^n \operatorname{poly}(n))$-time quantum solution (based on a better classical algorithm) exists [2], but we will only be implementing the former speedup. The focus of our project will be to optimize the reduction from Boolean formulae to the phase-flip circuits required by Grover's algorithm.

2.2 Grover's Algorithm

Grover's algorithm [2] uses a process of amplitude amplification, a method of determining which outcomes are possible by increasing the probability amplitude of solutions marked by an indicator oracle f. This oracle, the main input to Grover's algorithm, is used to construct a unitary operator that flips the phase of only states representing solutions. In our case, the oracle corresponds to a reversible version of the verifier for 3-SAT.

More generally, we require a phase-flip operator U that flips the phase of only positive results:

$$U|x\rangle = \begin{cases} -|x\rangle & f(x) = 1 \\ |x\rangle & f(x) = 0 \end{cases}$$

Note that this can be constructed from a bit-flip operator U_{bit} which flips an ancilla qubit, by preparing the ancilla qubit in the $|-\rangle$ state. Given a bit-flip operator U_{bit}, $(I \otimes XH)U_{bit}(I \otimes HX)$ is a phase-flip operator:

$$\begin{aligned} U_{bit}|x\rangle|q\rangle &= |x\rangle|f(x) \oplus q\rangle \\ \implies (I \otimes XH)U_{bit}(I \otimes HX)|x\rangle|0\rangle &= (I \otimes XH)U_{bit}|x\rangle|-\rangle \\ &= (I \otimes XH)|x\rangle \frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} \\ &= (I \otimes XH)(-1)^{f(x)}|x\rangle|-\rangle \\ &= (-1)^{f(x)}|x\rangle|0\rangle \end{aligned}$$

Grover's algorithm constructs a uniform superposition over all states corresponding to potential solutions. In particular, ancillary qubits must be left in the pure $|0\rangle$ state, otherwise the algorithm will also find solutions corresponding to non-$|0\rangle$ ancillary qubits. Then, in each iteration, it applies the phase-flip oracle to the state, and then applies the Grover diffusion operator to the result. The phase-flip oracle causes satisfying inputs to decrease in amplitude, so the the average amplitude over all the states is decreased. Then, the Grover diffusion operator reflects the amplitude of all states across the average. Hence, non-satisfying inputs (with an amplitude above the average) decrease in amplitude, but satisfying inputs (with an amplitude below the average) increase in amplitude. It turns out that this amplification operation only requires $\mathcal{O}(\sqrt{N})$ iterations to amplify any (possibly exponentially small) initial amplitude for the satisfying input to one that is arbitrarily close to 1. Hence, after amplification, we can then just measure the result to find that state.

2.3 Multiple solutions

It is known that for oracles with at most one solution, the optimal number of iterations to apply is $\frac{\pi}{4}\sqrt{N}$, where $N = 2^n$ is the size of the search space. However, it is also possible to handle oracles with multiple solutions. For oracles with k solutions, the optimal number of iterations is instead $\frac{\pi}{4}\sqrt{\frac{N}{k}}$. To handle oracles with a possibly unknown number of solutions, the algorithm can be repeated using powers of two for k, which still converges to a total number of iterations proportional to $\sqrt{N}$.

Changing the number of iterations suffices to handle oracles with unknown numbers of solutions. Hence, in our implementation, we restrict ourselves to the case of oracles with at most a single solution, since the circuit reduction is independent of number of satisfying assignments in the Boolean formula.

3 Oracle Construction

3.1 AND and OR Gates

We want to find a way to 3-SAT instances, and more generally Boolean formulae in conjunctive normal form (CNF) to reversible quantum oracles. Critical to this process is the implementation of classical AND and OR gates on qubits. Reversible AND and OR gates can be implemented naively using Fredkin gates, which are universal. They can also be implemented using CCNOT gates.

3.2 Fredkin reduction

The Fredkin (CSWAP) gate can be used to implement AND and OR directly by supplying an ancillary either 0 or 1 as the last wire:

$$\text{CSWAP}(a,b,c) = \begin{cases} (a,c,b) & a \\ (a,b,c) & \neg a \end{cases} \implies \begin{array}{l} \text{CSWAP}(a,b,0) = (a, \neg a \wedge b, a \wedge b) \\ \text{CSWAP}(a,b,1) = (a, a \vee b, \neg(a \wedge \neg b)) \end{array}$$

The first circuit compilation strategy uses these gates to evaluate the Boolean circuit. For each clause, we apply the OR reduction on pairs of qubits to produce intermediate results, and repeat until the entire clause is evaluated. Then, we apply the AND reduction over the results of all the clauses. After the gate applying the circuit's result to the final ancillary qubit, the rest of the circuit prior to that gate is reversed to undo any changes to the input qubits.

However, since CSWAP also changes one of the other incoming wires, each application of CSWAP requires an introduction of an additional ancilla qubit with the same value as the original, which can be prepared with a CNOT gate. Furthermore, negated terms in each clause also require qubit copies for the same reason. Hence, a considerable number of ancillary qubits are needed, though the total number of qubits is still proportional to the number of variables and size of the formula (the total number of terms across all the clauses).

3.3 CCNOT reduction

The CCNOT gate can also be used to implement AND in a more direct fashion: $\text{CCNOT}(a,b,c) = (a,b,c \oplus (a \wedge b))$. This is advantageous because the two input qubits a, b are not changed, so copy qubits are not required. OR can be implemented directly by an application of de Morgan's identity: $a \vee b = \neg(\neg a \wedge \neg b)$.

As above, we apply the OR gate constructed from CCNOT multiple times to produce the result of each clause, and then the AND gate over the clause results to evaluate the circuit. Negated terms are still handled by preparing qubits representing the negation of each variable, and using them whenever they are referenced in a clause.

The lack of qubit copies already saves a few qubits. We apply two further independent operations to reduce the number of qubits and gates required.

- **Negation tracking (NT).** Another way to handle negated terms is to only use one qubit per variable, and apply X to it whenever its negated value is required. Furthermore, instead of reversing X eagarly after it is used, we lazily track the variables that have been negated, and apply X only if needed.
- **Clausal de Morgan (dM).** Instead of directly applying OR $= X_1X_2X_3\text{CCNOT}_{12}X_1X_2$ over pairs of intermediate results in each clause, we apply de Morgan's identity over the entire clause instead, so it is instead expressed as an AND over the terms. This has the effect of eliminating redundant X operations on intermediate results when they will negated again when taking the OR with the next term.

4 Experiment

The above provide 5 oracle constructions: one based on Fredkin gates, and four based on CCNOT gates with the two optimizations (dM, NT) turned either on or off. To evaluate these oracle constructions, we generate several CNF formulae that have a single solution, using the following two methods:

1. **Random bitstrings.** We generate a random bitstring $b \in 0, 1^n$ and create the CNF $\phi(x) = \wedge_{i=1}^{n} \begin{cases} x_i & b_i \\ \neg x_i & \neg b_i \end{cases}$.

   We sampled 5 such formulae for each $n \in [2, 8]$.
2. **Random 3-SAT.** The above method produces formulae with only one term in each clause. To produce 3-SAT instances, we employ the following random procedure: randomly sample a clause, and add it to the formula if it reduces the number of satisfying assignments but does not make the formula unsatisfiable. Repeat until only one assignment is left. We sampled 5 such formulae for each $n \in [2, 4]$. Note that this tends to produce a formula of size exponential in the number of variables (each clause rules out a maximum of 2^{n-3} assignments).

We also verify that the oracle produces the correct result using the implementation of Grover's algorithm in Rigetti's Grove quantum algorithm library [3]. We supply the oracle and the variable qubits as parameters to the Grover Oracle function, and it runs Grover's Algorithm for as many iterations as is necessary to solve the problem. The results are not deterministic, so we can determine the error rate by repeating it 1000 times. Due to computational time constraints, we only simulated Grover's algorithm for instances with a maximum of 25 qubits. For the same reason, we measure the resulting number of qubits and gates used as a function of the number of variables as a proxy for the runtime of the algorithm.

5 Results

The relatively high accuracy of producing the satisfying assignment is shown in the figure below, indicating that our reductions are correct.

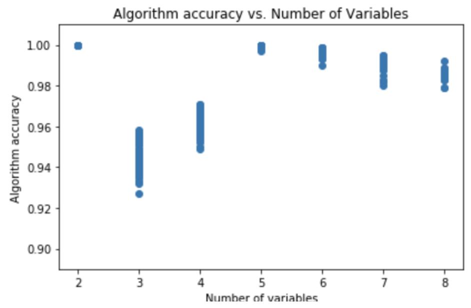


Figure 1: Probability of recovering a satisfying assignment, over all sampled CNFs and oracle constructions, for instances requiring up to 25 qubits in total.

Next, we note that the Fredkin reduction produces an unacceptable scaling of the number of ancillary qubits required to the number of actual variables:

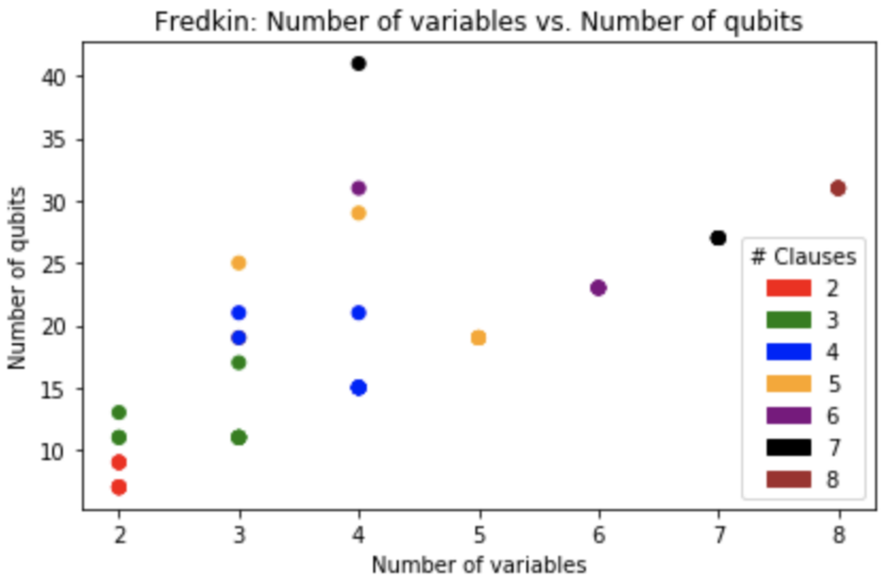
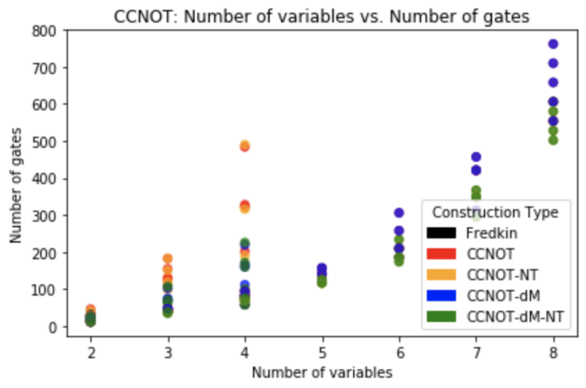
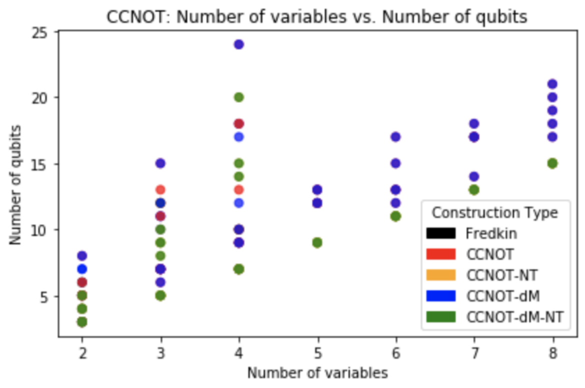


Figure 2: Total number of qubits in the circuit using the Fredkin reduction.

A 2-variable formula instance requiring 14 qubits in total, and even a bitstring formula over 8 variables requiring over 30 qubits is very wasteful. Furthermore, it takes far too long to simulate more than 25 qubits on the QVM, and quantum lattices with large numbers of qubits are expensive. Hence, the Fredkin reduction is not well-suited to practical uses. We thus shift our attention to the CCNOT reduction:



(a) Clausal de Morgan results in a significant reduction in the number of gates



(b) Negation tracking results in a lower number of qubits required

Figure 3: Plots

From the above results, we can see that the number of qubits scales linearly as clauses increase and as variables increase, which is expected. Qubits (and corresponding ancilla bits needed to model the circuit or to represent negations) model the variables of the CNF. We can also see that the number of gates scale exponentially as more qubits are added, which is also expected since exploring possible combinations of solutions through gate combinations still requires exponential time. However, the number of gates is also highly dependent on the number of positive or negated variables in the CNF, as well as the number of variables that appear in each clause of the CNF. Clausal de Morgan and negation tracking also result in improvements in the number of ancillary qubits required as well as the total number of gates in the circuit.

6 Code and Instructions

Download the code from the following link. Running `grover_3sat.py` makes the tests we did to produce our results.

If you would like to input your own CNFs, it may be a little experimental (since you should only add ones that you know to only have one solution), you can add them in the bottom runnable section of the python file, where there is a variable designed to add custom CNF strings. Uppercase letters are not negated, while lowercase numbers are negated, and a space marks the end of a clause.

Thank you to the teaching team for a great quarter.

References

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

[2] Lov K Grover. A fast quantum mechanical algorithm for database search. *arXiv preprint quant-ph/9605043*, 1996.

[3] Robert S Smith, Michael J Curtis, and William J Zeng. A practical quantum instruction set architecture, 2016.