Solving 3-SAT Expressions Using Grover's Algorithm

Michelle Ding and Letizia Fazzini

November 28, 2022

1 Introduction

A 3-SAT boolean formula is made up of a number of conjunctions joined by logical "AND" gates, within which there are exactly three literals joined by "OR" gates and possibly negated. Determining whether there exists a set of truth values that makes this expression evaluate to true is an NP-complete problem, but finding these solutions can be done much faster than classically using Grover's Algorithm. In our project, we implemented a function that computes the quantum oracle of arbitrary 3-SAT expressions, then used a classical algorithm to find the number of solutions the expressions has, and finally ran Grover's Algorithm to identify these solutions with high probability.

2 Generating the Expressions

Our 3-SAT generator produces a pseudo-random Boolean formula of varying complexity based on an input number of literals and number of conjunctions in the expression. Namely for each conjunction, we choose three random literals each denoted by a number from 1 to `num_literals` and append a random signum to the number to represent the truth value (+ positive, - negative). The end representation is a list of conjunctions, each containing a list of three signed literals.

3 Building the Quantum Oracle

The Quantum Oracle was built in a way that minimized the number of auxilliary bits needed. First, the individual conjunctions were evaluated and the result of each was placed in auxilliary bits 0 to $n-1$, where n was the number of clauses. To achieve this, the result of the first "OR" in the clause was placed in auxilliary bit n, and the second "OR" was computed between this value and the third input in the clause. Next, to be able to reuse auxilliary bit n, the first "OR" was "uncomputed", which was easily achieved since both X and CNOT gates are reversible. Lastly, a multi-CX gate was used to combine all the clauses' results into the output qubit. Thus, the total number of auxilliary bits used was only $n+1$.

Normally, "AND" gates as well as "OR" gates were implemented as shown in Lesson 7, however we also had to handle the special case of computing the "OR" of a certain input with itself. In the case of either "x or x" and "!x or !x", we could simply use a CX (rather than CCX) gate, possibly negating the inputs. Meanwhile, "x or !x" and "!x or x" always evaluate to true, so we simply added an X gate to the auxilliary qubit without needing to control on the input.

Additionally, since the Oracle sometimes needed to be run multiple times (when $R > 1$) we also added code to uncompute all the gates applied to the auxilliary qubits so they could be reused in later iterations of Grover's Algorithm.

4 Finding the Number of Solutions

Before counting the number of solutions, we first determine if the 3-SAT formula is satisfiable. To do this, we implement a basic version of the DPLL Algorithm. We first take a literal (we use the first one in the expression for simplicity but the choice of literal can be arbitrary), and use propositional logic rules to simplify the rest of the formula after assigning the truth value of the given literal on a case-by-case basis. Once we have reached a state where either the formula is null or there is a null clause, we know the interpretation is valid or invalid respectively, and we recurse up the tree to either return the expression as Satisfiable or try another combination of literals. This is due to the short-circuiting behavior of the algorithm, as an empty formula means all clauses have short-circuited to True based on some choice of literals and any null clause indicates all variables in the clause have been eliminated as False one by one, resulting in an invalid interpretation.

After determining the satisfiability of the 3-SAT formula, we either terminate early if the expression is Unsatisfiable, as there would be no point in running Grover's algorithm with $R = 0$ iterations, or run a brute force algorithm to check for the number of solutions. The latter is done by assigning a True or False interpretation to each of n literals on a case-by-case basis. This classical approach has a $O(2^n)$ runtime.

5 Using Grover's Algorithm

We implement Grover's iterative search algorithm to find the best bitstrings. A bitstring consists of a sequence of literals, and those of high probability likely correspond to a valid interpretation of the 3-SAT formula. First, we set up our quantum registers. We compose our circuit with our oracle function as defined in section 3 of the report and then with the W gate as defined in Grover's algorithm. We initialize a new circuit $qc2$ to run over all R iterations of Grover's algorithm, then display a histogram to view the resulting bitstrings. Iterating through all the bitstrings, we keep track of the first n bistrings of highest occurrence, which we check correspond to the n classical solutions to the 3-SAT.

6 Experimental Results

Running our algorithm on several different types of boolean expressions we observed the following interesting results:

- The algorithm is much more accurate when the expression has relatively few solutions. In particular, when more than half of the possible inputs led to a solution, we achieved much better results by negating the expression and running Grover's Algorithm to find the set of non-solutions (from which we could then derive the real solutions).
- Resetting the auxilliary bits added some noise to the results, likely because it doubled the number of gates applied to each input and thus increased the chance of an error occurring. Thus, there is a tradeoff between the depth of the circuit and the number of qubits needed.

7 Sources

Boolean Satisfiability Problems: https://en.wikipedia.org/wiki/Boolean_satisfiability_problem