CSCI 4140
Satisfiability

Mark Green

Faculty of Science

Ontario Tech

Introduction

- This is one of the classical problems in mathematical logic
- It is based on logical expressions without quantifiers, that is no existential or universal quantifiers
- It is possible to solve this problem classically, but the obvious algorithm is exponential
- We can solve this problem efficiently using Grover’s algorithm, changing it from a logic problem to a search problem

Basics

- Logical expressions are built from a small number of items
- Logical variables, x_i can have the value TRUE or FALSE, they are binary
- There are three logical operators:
  - $\neg$ - not
  - $\vee$ - or
  - $\wedge$ - and
- We use the or operator to build clauses, one or more variables or'ed together

Basics

- A logical expression consists of a number of clauses that are linked together by the and operator
- The particular logical expression that we will be interested in is:

$$f(v_1, v_2, v_3) = (\neg v_1 \vee \neg v_2 \vee \neg v_3) \wedge (v_1 \vee \neg v_2 \vee v_3) \wedge (v_1 \vee v_2 \vee \neg v_3) \wedge (v_1 \vee \neg v_2 \vee \neg v_3) \wedge (\neg v_1 \vee v_2 \vee v_3)$$

- This is called conjunctive normal form, we can put any logical expression into this form

Satisfiability

- The satisfiability problem is stated in the following way:
  - Find an assignment of TRUE or FALSE to the x_i such that the logical expression is true
  - If this is possible we say that the expression is satisfiable, otherwise it is not satisfiable
- There may be multiple combinations of values that satisfy the expression, we only need to find one, there could also be none
- The classical algorithm is based on constructing a truth table, which only works for a small number of variables

Satisfiability

- The particular version of the problem we are looking at is 3-sat, since there are three variables in each clause, the general problem is k-sat
- Our approach to solving this problem is to use Grover's algorithm
- Recall: Grover's algorithm searchers unstructured data looking for an item that satisfies an oracle
- All we need to do is convert our logical expression into an oracle and our problem is solves
- Fortunately, Qiskit provides a function for doing this

Satisfiability

- We need to convert our logical expression into a format that this function can use
- Our logical expression becomes the following in DIMACS CNF

```
c example DIMACS CNF 3-SAT
p cnf 3 5
-1 -2 -3 0
1 -2 3 0
1 2 -3 0
1 -2 -3 0
-1 2 3 0
```

- The first line is a comment, the first non-comment line must start with p, cnf states this is conjunctive normal form, the next number is the number of variables followed by the number of clauses

Satisfiability

- Each clause is one a separate line, terminated by a zero
- A positive number indicates a variable number, a negative number is the not of that variable
- Now let’s start converting this to Python code

```
import numpy as np
from qiskit import BasicAer
from qiskit.visualization import plot_histogram
from qiskit.aqua import QuantumInstance
from qiskit.aqua.algorithms import Grover
from qiskit.aqua.components.oracles import LogicalExpressionOracle, TruthTableOracle
```

Satisfiability

- We convert our logical expression to DIMACS CNF

```
input_3sat = '''
c example DIMACS-CNF 3-SAT
p cnf 3 5
-1 -2 -3 0
1 -2 3 0
1 2 -3 0
1 -2 -3 0
-1 2 3 0
'''
```

- The LogicalExpressionOracle function converts this expression into an oracle for Grover’s algorithm

Satisfiability

- Qiskit provides a Grover's algorithm function that implements the algorithm, so we don't have to cut and paste or own implementation
- This gives

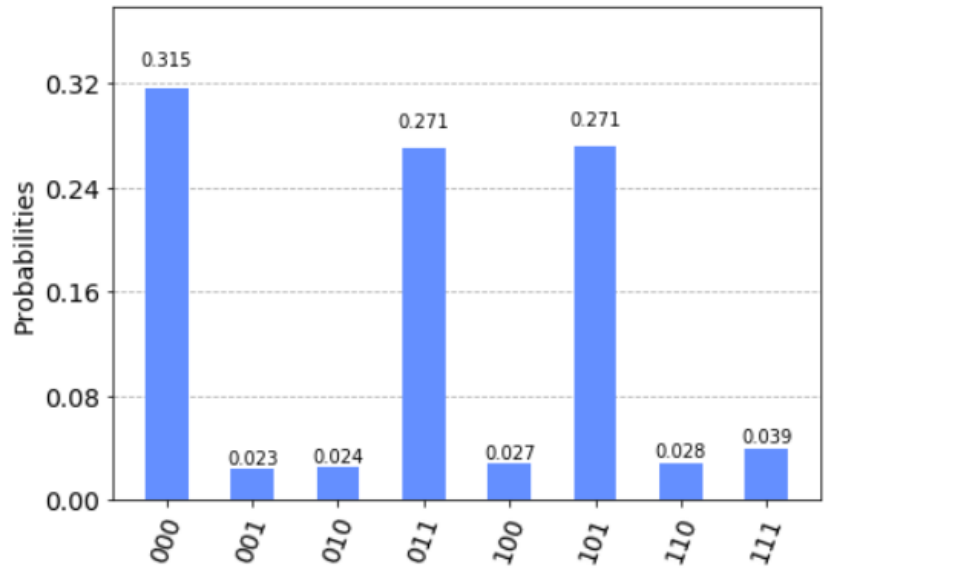
```
oracle = LogicalExpressionOracle(input_3sat)
grover = Grover(oracle)
backend = BasicAer.get_backend('qasm_simulator')
quantum_instance = QuantumInstance(backend, shots=1024)
result = grover.run(quantum_instance)
print(result['result'])
```

```
[-1, -2, -3]
```

Satisfiability

- This gives us one solution, there are two other solutions
- A histogram shows the two other solutions

```
: plot_histogram(result['measurement'])
:
```



Satisfiability

- Let’s try a slightly different example:

```
input_3sat = '''
c example DIMACS-CNF 3-SAT
p cnf 3 2
-1 -2 -3 0
1 2 3 0
'''
```

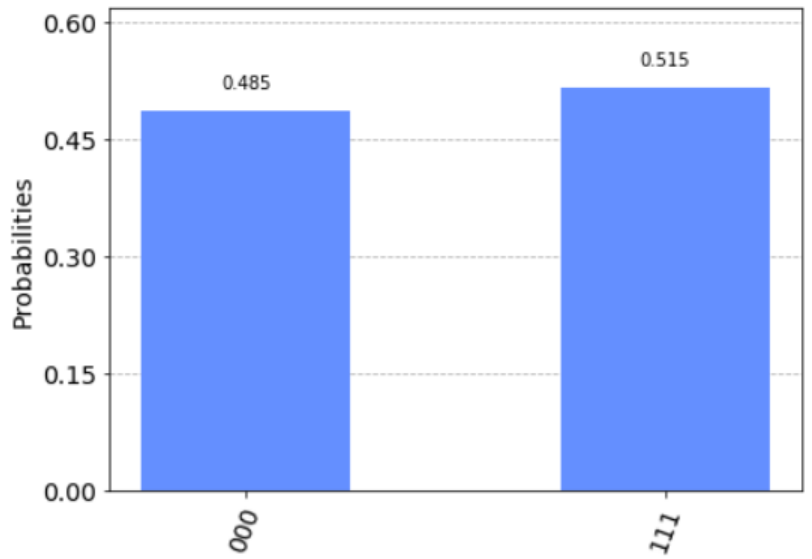
- This particular example is not satisfiable, so let’s see what happens with the Grover algorithm

Satisfiability

```
oracle = LogicalExpressionOracle(input_3sat)
grover = Grover(oracle)
backend = BasicAer.get_backend('qasm_simulator')
quantum_instance = QuantumInstance(backend, shots=1024)
result = grover.run(quantum_instance)
print(result['result'])
```

```
[1, 2, 3]
```

```
plot_histogram(result['measurement'])
```



Satisfiability

- What happened? It appears to have a solution, but it’s wrong
- This is what happens when you don’t check all the output

```
oracle = LogicalExpressionOracle(input_3sat)
grover = Grover(oracle)
backend = BasicAer.get_backend('qasm_simulator')
quantum_instance = QuantumInstance(backend, shots=1024)
result = grover.run(quantum_instance)
if result['oracle_evaluation']:
    print(result['result'])
else:
    print("No Result")
```

```
No Result
```

Summary

- Showed how a logic problem can be converted into a search problem
- Lesson: need to be able to see problems in a different light, may already have an algorithm to solve the problem
- Also need to be careful to check all of the output
- Problem: the resulting circuit is too large to run on existing quantum computers
- An area that will hopefully become more practical over time