Study of Existing Quantum Search Algorithms and Problem Formulations to Determine the Most Efficient Method to Solve Constraint Satisfaction Problems

By Sidharth Dhawan
GROVER ALGORITHM
Applications of Quantum Computers

• Classical computers today are fast.

• However, in some cases, quantum computers are significantly faster.
  – For example, Shor’s algorithm can solve semiprime factorization problems with exponential speedups over classical computers.
  – Grover’s algorithm can achieve polynomial speedups in large, non-polynomial problems using unstructured search.
Superposition and Quantum Computers

• A classical bit is represented by a classical entity, like a current of electrons.

• Thus, it is confined to two discrete states, “0” and “1”.

• It is relatively easy to determine its state.
Superposition and Quantum Computers

• A quantum bit represents information in a quantum mechanical entity, like an electron’s spin axis

• Our tools cannot determine the precise state of a quantum entity

• Thus, a qubit can exist in any combination of the states “0” and “1” (in other words, any point on the bloch sphere).

• A classical bit is confined to the “poles”
Superposition and Search

• We can use the **Hadamard transform** to create an even superposition between the \( |0\rangle \) and \( |1\rangle \) state in one qubit.

• If I perform a collective Hadamard transform to a system of two qubits, the system *as a whole* can represent \( |00\rangle, |01\rangle, |10\rangle, \) and \( |11\rangle \).

• For this reason, quantum bits store more information than classical bits – I can represent \( 2^n \) classical states with \( n \) qubits.
Quantum Unstructured Search ("Grover")

- **Grover’s Algorithm** uses this property of quantum information to perform an unstructured search more quickly.

- The initial input qubits are superposed to represent all possible solutions.

- The **Oracle** operation tags the phase of the solution states in this superposition.

- Another circuit then changes the phase information (which is hidden) into amplitude information (which we can detect).

- This process is iterated $\sqrt{N}$ times (as opposed to $N$ iterations in classical logic) to maximally amplify the states.
Circuit level representation of Grover
Applications of Grover Algorithm

- Grover’s algorithm can provide quadratic speedup in NP Complete problems:

- Examples of NP Complete problems are:
  - Graph Coloring
  - Maximum Clique
  - Satisfiability
  - Travelling Salesman
  - DNA Sequencing
  - Scheduling
  - Sudoku
The Oracle

• The most important part of the Grover Circuit is the oracle.

• An oracle is essentially a classical circuit that can recognize a state (combination of inputs) that is a solution.

• In the Grover loop, the oracle searches through every solution simultaneously and “tags” the solution state with a phase change.
The Oracle

• Since the oracle operation is iterated $\sqrt{N}$ times, a decrease in cost of one basic gate for the oracle would decrease the cost of the entire Grover loop by many more gates.

• The number of input qubits is also important, because the Grover Circuit must be iterated $2^{(n/2)}$ times.

• For example, the Grover Circuit for SEND MORE MONEY costs 17 thousand trillion trillion more basic gates with the less efficient method.
Graph Coloring

- Graph coloring is an NP complete problem
- It involves finding a “good” coloration for a system of n nodes connected by e edges
- No two nodes connected by an edge can have the same color
Perkowski’s Method

• Since there are three nodes, each can take on up to three colors.

• Each node is represented with $|\log_2(3)| = 2$ qubits so that the total number of possible collective states per node is more than three.

• For example, the states $|00\rangle$ and $|01\rangle$ represent different colors.

• I have to make sure no two nodes that are connected are assigned the same color using bit-by-bit inequality gates.
Hogg’s Method

• In this method, each assignment of a color to a node is represented by a qubit, v1-v9.

• No two elements of a row can coexist, because only one color can be assigned to a node. I use NAND gates to ensure this.

• If, for example, nodes one and two are connected, we must do: v1 NAND v4, v2 NAND v5, and v3 NAND v6.
Reversible Logic and Quantum Cost

• An alternative classical logic implementation is called AND–EXOR logic.

• It is reversible because you can determine inputs from the outputs.

• This kind of logic is easier to simulate with most quantum technologies.
Reversible and Non Reversible Circuits

\[ \text{sum} = a \oplus b \oplus c \]

\[ \text{carry} = ab \oplus c(a \oplus b) \]

\[ \text{sum} = a \oplus b \oplus c \]

\[ \text{carry} = ab \oplus ac \oplus bc \]
# Quantum Cost

<table>
<thead>
<tr>
<th>Gates</th>
<th>Cost in Basic Gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantum NOT</td>
<td>1</td>
</tr>
<tr>
<td>Hadamard Gate</td>
<td>1</td>
</tr>
<tr>
<td>CNOT Gate</td>
<td>1</td>
</tr>
<tr>
<td>3 input Toffoli Gate</td>
<td>5</td>
</tr>
<tr>
<td>N input Toffoli Gate</td>
<td>Best Case: 32n-96</td>
</tr>
<tr>
<td></td>
<td>Worst Case: ((2^{n+1}) - 3)</td>
</tr>
</tbody>
</table>
N-Bit Toffoli Cost

• I have used two estimates to calculate the cost of a toffoli gate.
  – $32m - 96$, plus one garbage bit, for $m > 5$
  – $2^{(m+1)} - 3$, where $m$ is the number of controlling bits

• The difference between these costs should be underscored.

• For example, one technique for building the SEND MORE MONEY oracle costs about 100,000 basic gates with the best case method and over a googol with the worst case method.
Goal

In this paper, the costs of these two data-encoding methods for building Grover Oracles are compared by testing both for four problems:

- Satisfiability (SAT),
- Maximum Clique,
- SEND MORE MONEY
- Graph Coloring
COST DERIVATION FOR GRAPH COLORING
Graph Coloring

- Graph coloring is an NP complete problem

- It involves finding a “good” coloration for a system of $n$ nodes connected by $e$ edges

- No two nodes connected by an edge can have the same color
Step 1: An oracle is created for a specific case of each problem.
<table>
<thead>
<tr>
<th>Colors ➔</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>V1</td>
<td>V2</td>
<td>V3</td>
</tr>
<tr>
<td>2</td>
<td>V4</td>
<td>V5</td>
<td>V6</td>
</tr>
<tr>
<td>3</td>
<td>V7</td>
<td>V8</td>
<td>V9</td>
</tr>
</tbody>
</table>
Step 2: How many gates are needed in a generic case with n nodes and e edges?
Cost Derivation for Graph Coloring

**STEP 1:**
2-input NAND = Toffoli = 5 basic gates

**STEP 2:**
e*n + n²(n-1)/2 NANDs = 5(en+n²(n-1)/2) pulses

**STEP 3:**
m – input NAND = 32m – 96 pulses (best case)

**STEP 4:**
Final NAND = 32(en+n²(n-1)/2) - 96 basic gates

**Total Cost:**
(2*STEP 2 + STEP 4) = 42(en+n²(n-1)/2) – 96 Basic gates

**Step 3:** Convert gates to cost with cost formulas
The Mirror Circuit

• At the end of the oracle, the qubits in the “Work space” must be returned to their original states for the next oracle operation.

• Because this is reversible logic, we can simply use the reverse of the gates originally applied – this is called a mirror circuit, and it must be factored into cost estimates.
Step 2: How many gates are needed in a generic case with n nodes and e edges?
The Inequality Gate

- The Feynman gate outputs “1” when both inputs are not equal.

- Thus, we create Feynman gates between corresponding qubits of different nodes.

- All the results are ORed with a final Toffoli gate – the output will be one if ANY inputs are one.

- Thus, because there are $2q+1$ NOT gates, $2q$ Feynman gates, and a $q$-input Toffoli gate, the cost is $(2q+1)+(2q)+(32q-96) = 36q-95$. 
Cost Derivation – Perkowski’s Method

**STEP 1:**
Inequality Gate = 36q - 95 basic gates

**STEP 2:**
e Inequality Gates = e(36q - 95)

**STEP 3:**
e – input NAND = 32e – 96 pulses (best case)

**Total Cost:**
(2*STEP 2 + STEP 3) = 2e(36q - 95) - 96 basic gates

**Step 3:** Convert gates to cost with cost formulas
Step 4: Once I figured out my formulas for all problems, I graphed them and compared the results to see which method works better.
SEND MORE MONEY
SEND MORE MONEY

- The goal of this problem is to find a correct integer assignment to each of the letters so that the equation above is satisfied.

- We can tackle this problem better by reducing it to smaller equations, as shown:
  
  \[
  \begin{align*}
  D + E &= 10 \times C_1 + Y \\
  C_1 + N + R &= 10 \times C_2 + E \\
  C_2 + E + O &= 10 \times C_3 + N \\
  C_3 + S + M &= 10 \times M + O \\
  \end{align*}
  \]
SEND MORE MONEY - Perkowski

Perkowski’s method:

• The oracle on the right uses Perkowski’s method to solve the problem

• Each letter is represented by four qubits and can take on ten values.

• This emulates the formulas with gates
  – for example, the conditions $D + E$ and $10*c1+Y$ are plugged into the bottom equality gate

• Best case cost:
  5,186 basic gates and 126 Qubits
Inequality Gates

• Every combination of two letters must be inputs to an inequality gate

• Thus, we need $n(n-1)/2$ inequality gates.

• Recall that the cost of an inequality gate is $4n+1$ in Feynman's and inverters, plus (in this case) a four-controlled toffoli gate (29). Thus, our gate costs 46 basic gates.

• Since $n=8$ (letters) and $q=4$, the total cost of this step is $8(7)/2 \times (46) = 1288$ basic gates.
<10 Block

• Each letter should have a value of less than ten

• A simple “<10” block could be created by the operation \((a_1*(a_2+a_3))’\),
  – Since 10 is 1010 in binary, both the most significant bit and either the second or third must be one if a number is greater than ten.

• This circuit will consist of a toffoli gate, an OR gate (which is a Toffoli gate plus four inverters), and a final inverter. It will cost 16 basic gates

• Seven of these are required, one per letter. Thus the cost of this step is 112 basic gates.
\[
\begin{align*}
D + E &= 10 \times C_1 + Y \\
C_1 + N + R &= 10 \times C_2 + E \\
C_2 + E + O &= 10 \times C_3 + N \\
C_3 + S + M &= 10 \times M + O
\end{align*}
\]

There are two types of equations that must be described with adder circuits.

For example, there is \(C_1 + N + R\), which requires full adders.

Also, there is \(10 \times C_1 + Y\), which must be described using half-adders.
Adder Gates

- A half adder circuit is shown on the right
  - This circuit adds two inputs
  - As shown in the truth tables below, sum is represented by EXOR, while carry is represented by AND.
- The cost of this gate is 6 basic gates.

- A full adder gate is shown on the bottom right:
  - In a full adder gate, the sum is represented by: a EXOR b EXOR c.
  - The carry is represented by ab EXOR ac EXOR bc.
- The cost is 12 basic gates.

<table>
<thead>
<tr>
<th>a\b</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a\bc</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>10</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>
Adder Tree

C1+N+R:

- A full adder tree can be used to add two letters, as shown.

- The “carry” from the first full adder is carried into the second.

- The cost of one of these trees is four full adder gates, plus one feynman gate to bring the “carry” gate down to one ancilla bit. Thus, it costs 4(12) +1 = 49 basic gates.
Adders for $10 \times c_1 + Y$

- In binary, $10 = 1010$.
- Since all the carries are one qubit, $10 \times c_1 = c_1, 0, c_1, 0$.
- Thus, $S_1$ will equal $Y_1$.
- $C_1 + Y_2$ will be a half-adder, because only two qubits are being added.
- $C_1$ only needs to be added to $Y_2$ and $Y_4$, thus, the only Full adder will come at the end, because this is the only time three qubits are being added.
- Thus, the cost is 2 feynmans, two half-adders, and a full adder $= 26$ basic gates.
Special Case: E+D

- In this case, only the first adder is a half adder, and the rest are full adders, because there is no carry bit.

- Thus, the cost is 43 basic gates
Total Cost of adders

• There are three equations of type: $c_1 + N + R$

• There are four equations of type: $10 \times c_1 + Y$

• There is one equation of type: $E + D$

• Thus, total cost of all adders is:

$$3(49) + 4(26) + (43) = 294 \text{ basic gates.}$$
Equality Gates

- This circuit will have four equality gates. They will each operate on two five-qubit inputs (the outputs of the adder gates).

- An equality gate costs $4n$, plus a (in this case) five-controlled Toffoli gate, which costs 52 qubits, best case.

- Thus, the total cost will be 72 per circuit, and 288 basic gates for all four circuits.
The Letter M

• The letter M is represented by four qubits, so that we can use it in inequality gates. However, it can only be zero or one, because it is a carry.

• Thus, we need a three-input NOR gate to make sure that three of its qubits all equal zero.

• This will cost 6 inverters and a 3-controlled Toffoli (13), or **19 basic gates** total.
Final Toffoli Gate

• There are $8(8-1)/2 = 28$ outputs from the first inequality section.

• There are 4 outputs from the equations, and 1 output from the letter M.

• There are 7 outputs from the “<10” block.

• Thus, there are 40 inputs, so the cost is $32(40)-96 = 1184$ basic gates.
Hogg’s Method

Since

- each one of 8 letters can take 9 values (1 to 9), and
- each one of 3 carry’s can take 2 values (0, 1),
- this oracle requires 78 variables
SEND MORE MONEY – Hogg’s Method

Hogg’s oracle will require us to figure out all possible assignments to the four formulas:

\[
\begin{align*}
D + E &= 10*C1 + Y \\
C1 + N + R &= 10*C2 + E \\
C2 + E + O &= 10*C3 + N \\
C3 + S + M &= 10*M + O
\end{align*}
\]

• For example, the assignments 1,2,0,3 and 1,3,0,4 for the variables D, E, c1, and Y, (respectively) satisfy the first equation.

• We then use AND and OR gates to make sure that at least one of these assignments are satisfied.

• There are about 160 such equations; each of these will require a five or six input Toffoli gate, and then a massive OR gate at the end.

• The total cost of all these gates is 35213 basic gates, and four output qubits.
SEND MORE MONEY – Hogg’s Method

No two letters can be assigned the same value

• This will require $n(n-1)/2$ NAND gates, times the number of values(10) = $10(8)(7)/2 = 280$ NAND gates

• The total cost of this step is thus 1400 basic gates

• 280 output qubits are involved
SEND MORE MONEY – Hogg’s Method

Next, no two values can be assigned to the same letter

• This will require $n(n-1)/2$ NAND gates, where $n = 10$, and this will need to be repeated 8 times, for a total of 360 NAND gates

• Thus, the total cost is **1800 basic gates**

• 360 output qubits are involved
SEND MORE MONEY – Hogg’s Method

• For the final Toffoli gate, there are 4+360+280 = 644 controlling qubits

• Thus, the best case cost is 32(644)-96 = 20512 basic gates
  – The worst case cost is $2^{645} - 3$, which is over a googol.

• The overall cost amounts to $2(1800+1400+35213) + 20512 = 97216$ basic gates, best case, including mirror circuits.
SATISFIABILITY
**The Problem:** Find a Boolean (1 or 0) assignment of variables that gives an output of one in a formula such as the one shown on the top right.

**Perkowski’s method:**
- This formula is emulated by gates. In other words, I simply do: \( \{a \text{ NOT}\) OR b OR c\} AND \{a OR (b NOT) OR (c NOT)\} with quantum gates. NOT gates are represented by X’s in the figure.

**Hogg’s method**
- This is exactly the same, except that no NOT gates are needed; instead, inputs are taken from different qubits. However, we also need inequality (CNOT) gates to make sure no two representative qubits have the same value.

For a large number of AND-ed terms and NOT-ed terms, and few variables, it is possible that Hogg’s method could be more efficient than Perkowski’s method because Perkowski’s method requires more NOT gates.
MAXIMUM CLIQUE
Maximum Clique

**The Problem:** Find the largest number of interconnected nodes in a graph

Perkowski’s Method:

1. I created the oracle for the 4 node graph as shown on the right.
2. I created this oracle by representing each node with a 1 if it was “activated”, i.e. part of the proposed clique, and with a 0 if it wasn’t.
3. I then NAND-ed all nodes that were not connected, because the maximum clique must have all nodes interconnected, so no two non-connected nodes can be part of the clique.
Maximum Clique – Hogg’s method

1. The oracle I created using Hogg’s method for this problem is very similar to the previous oracle. It creates two qubits per node: one representing the deactivated node, and the other representing an activated node.

2. First, I create similar gates to Perkowski’s method to check that the given assignment actually is a clique.

3. In addition, I have to create NAND gates between each node’s representative qubits to make sure that no qubit is simultaneously activated and deactivated.
# Results – Costs of Oracles

<table>
<thead>
<tr>
<th>Problems and Methods</th>
<th>Cost in Basic Gates</th>
<th>Cost in Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graph Coloring</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hogg’s</td>
<td>$\text{Best: } 42\left(\frac{en + n^2(n-1)}{2}\right) - 96$</td>
<td>$2 + en + \frac{n^2(n-1)}{2} + n^2$</td>
</tr>
<tr>
<td>Perkowski’s</td>
<td>$\text{Best: } 2e(36</td>
<td></td>
</tr>
<tr>
<td><strong>Maximum Clique</strong></td>
<td>Perkowski’s method + 42n</td>
<td>Perkowski’s + 2n</td>
</tr>
<tr>
<td>Hogg’s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perkowski’s</td>
<td>$\text{Best: } 42\left(\frac{n(n-1)}{2} - e\right) - 96$</td>
<td>$\frac{n(n-1)}{2} - e + 2 + n$</td>
</tr>
<tr>
<td><strong>Satisfiability</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hogg’s</td>
<td>$\text{Best: } 4v+2m*(2^{n+1}-3)+32(m+v)-96$</td>
<td>$v+m+2$</td>
</tr>
<tr>
<td>Perkowski’s</td>
<td>$\text{Best: } 2m*(2^{n+1}-3)+2t+32m-96$</td>
<td>$3v+m+2$</td>
</tr>
<tr>
<td><strong>SEND MORE MONEY</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hogg’s</td>
<td>$\text{Best: } 93,138$</td>
<td>$949$</td>
</tr>
<tr>
<td>Perkowski’s</td>
<td>$\text{Best: } 3952$</td>
<td>$2.2 \times 10^{12}$</td>
</tr>
</tbody>
</table>
# Results: Max Clique and Graph Coloring

## Max Clique

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hogg's best case</strong></td>
<td>Pulses</td>
<td>282</td>
<td>492</td>
<td>744</td>
<td>1038</td>
<td>1374</td>
<td>1752</td>
<td>2172</td>
<td>2634</td>
<td>3138</td>
<td>3684</td>
<td>4272</td>
<td>4902</td>
<td>5574</td>
<td>6288</td>
<td>7044</td>
</tr>
<tr>
<td></td>
<td>Qubits</td>
<td>23</td>
<td>30</td>
<td>38</td>
<td>47</td>
<td>57</td>
<td>68</td>
<td>80</td>
<td>93</td>
<td>107</td>
<td>122</td>
<td>138</td>
<td>155</td>
<td>173</td>
<td>192</td>
<td>212</td>
</tr>
<tr>
<td></td>
<td>Qubits</td>
<td>30</td>
<td>198</td>
<td>408</td>
<td>660</td>
<td>954</td>
<td>1290</td>
<td>1668</td>
<td>2088</td>
<td>2550</td>
<td>3054</td>
<td>3600</td>
<td>4188</td>
<td>4818</td>
<td>5490</td>
<td>6204</td>
</tr>
</tbody>
</table>

## Perkowski's Best Case

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>e=2n</strong></td>
<td>Pulses</td>
<td>30</td>
<td>198</td>
<td>408</td>
<td>660</td>
<td>954</td>
<td>1290</td>
<td>1668</td>
<td>2088</td>
<td>2550</td>
<td>3054</td>
<td>3600</td>
<td>4188</td>
<td>4818</td>
<td>5490</td>
<td>6204</td>
</tr>
<tr>
<td></td>
<td>Qubits</td>
<td>11</td>
<td>16</td>
<td>22</td>
<td>29</td>
<td>37</td>
<td>46</td>
<td>56</td>
<td>67</td>
<td>79</td>
<td>92</td>
<td>106</td>
<td>121</td>
<td>137</td>
<td>154</td>
<td>172</td>
</tr>
</tbody>
</table>

## Perkowski's Worst Case

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Qubits</td>
<td>42</td>
<td>50</td>
<td>58</td>
<td>67</td>
<td>75</td>
<td>84</td>
<td>93</td>
<td>102</td>
<td>111</td>
<td>121</td>
<td>130</td>
<td>139</td>
</tr>
</tbody>
</table>

## Graph Coloring

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hogg's best case</strong></td>
<td>Pulses</td>
<td>6708</td>
<td>10194</td>
<td>14688</td>
<td>20316</td>
<td>27204</td>
<td>35478</td>
<td>45264</td>
<td>56688</td>
<td>69876</td>
<td>84954</td>
<td>102048</td>
<td>121284</td>
<td>142788</td>
<td>166686</td>
<td>193104</td>
</tr>
<tr>
<td></td>
<td>Qubits</td>
<td>200</td>
<td>296</td>
<td>418</td>
<td>569</td>
<td>752</td>
<td>970</td>
<td>1226</td>
<td>1523</td>
<td>1864</td>
<td>2252</td>
<td>2690</td>
<td>3181</td>
<td>3728</td>
<td>4334</td>
<td>5002</td>
</tr>
<tr>
<td></td>
<td>Qubits</td>
<td>8220</td>
<td>12252</td>
<td>17376</td>
<td>23718</td>
<td>31404</td>
<td>40560</td>
<td>51312</td>
<td>63786</td>
<td>78108</td>
<td>94404</td>
<td>112800</td>
<td>133422</td>
<td>156396</td>
<td>181848</td>
<td>209904</td>
</tr>
</tbody>
</table>

## Complete Hogg

<table>
<thead>
<tr>
<th></th>
<th>Pulses</th>
<th>7464</th>
<th>12252</th>
<th>18720</th>
<th>27120</th>
<th>37704</th>
<th>50724</th>
<th>66432</th>
<th>85080</th>
<th>106920</th>
<th>132204</th>
<th>161184</th>
<th>194112</th>
<th>231240</th>
<th>272820</th>
<th>319104</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Qubits</td>
<td>125</td>
<td>203</td>
<td>285</td>
<td>379</td>
<td>483</td>
<td>597</td>
<td>721</td>
<td>855</td>
<td>1000</td>
<td>1155</td>
<td>1310</td>
<td>1475</td>
<td>1640</td>
<td>1805</td>
<td>2070</td>
</tr>
</tbody>
</table>

## Complete Perkowski

<table>
<thead>
<tr>
<th></th>
<th>Pulses</th>
<th>625</th>
<th>970</th>
<th>1344</th>
<th>1744</th>
<th>2168</th>
<th>2612</th>
<th>3075</th>
<th>3555</th>
<th>4052</th>
<th>4563</th>
<th>5088</th>
<th>5626</th>
<th>6176</th>
<th>6738</th>
<th>7311</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Qubits</td>
<td>42</td>
<td>50</td>
<td>58</td>
<td>67</td>
<td>75</td>
<td>84</td>
<td>93</td>
<td>102</td>
<td>111</td>
<td>121</td>
<td>130</td>
<td>139</td>
<td>149</td>
<td>159</td>
<td>168</td>
</tr>
</tbody>
</table>

## Perkowski's Worst Case

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Qubits</td>
<td>355</td>
<td>703</td>
<td>1164</td>
<td>1744</td>
<td>2451</td>
<td>3289</td>
<td>4264</td>
<td>5381</td>
<td>6644</td>
<td>8057</td>
<td>9624</td>
<td>11348</td>
<td>13233</td>
<td>15281</td>
<td>17496</td>
</tr>
</tbody>
</table>
# Results - SAT

\[ t = \frac{mn}{2} \]

<table>
<thead>
<tr>
<th>m/n</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>124</td>
<td>168</td>
<td>212</td>
<td>256</td>
<td>300</td>
<td>344</td>
<td>388</td>
<td>432</td>
<td>476</td>
<td>520</td>
<td>564</td>
<td>608</td>
<td>652</td>
<td>692</td>
<td>740</td>
<td>784</td>
</tr>
<tr>
<td>3</td>
<td>209</td>
<td>270</td>
<td>331</td>
<td>392</td>
<td>453</td>
<td>514</td>
<td>575</td>
<td>636</td>
<td>697</td>
<td>758</td>
<td>819</td>
<td>880</td>
<td>941</td>
<td>1002</td>
<td>1063</td>
<td>1124</td>
</tr>
<tr>
<td>4</td>
<td>374</td>
<td>468</td>
<td>562</td>
<td>656</td>
<td>750</td>
<td>844</td>
<td>938</td>
<td>1032</td>
<td>1126</td>
<td>1220</td>
<td>1314</td>
<td>1408</td>
<td>1502</td>
<td>1596</td>
<td>1690</td>
<td>1784</td>
</tr>
<tr>
<td>5</td>
<td>699</td>
<td>858</td>
<td>1017</td>
<td>1176</td>
<td>1335</td>
<td>1494</td>
<td>1653</td>
<td>1812</td>
<td>1971</td>
<td>2130</td>
<td>2289</td>
<td>2448</td>
<td>2607</td>
<td>2766</td>
<td>2925</td>
<td>3084</td>
</tr>
</tbody>
</table>

Hogg's Method \((v = n)\)

\[ t = \frac{mn}{3} \]

<table>
<thead>
<tr>
<th>m/n</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>204</td>
<td>264</td>
<td>324</td>
<td>384</td>
<td>444</td>
<td>504</td>
<td>564</td>
<td>624</td>
<td>684</td>
<td>744</td>
<td>804</td>
<td>864</td>
<td>924</td>
<td>984</td>
<td>1044</td>
<td>1104</td>
</tr>
</tbody>
</table>

Hogg's Method \((v = n)\)

\[ t = 2mn/3 \]

<table>
<thead>
<tr>
<th>m/n</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>214</td>
<td>276</td>
<td>338</td>
<td>400</td>
<td>462</td>
<td>524</td>
<td>586</td>
<td>648</td>
<td>710</td>
<td>772</td>
<td>834</td>
<td>896</td>
<td>958</td>
<td>1020</td>
<td>1082</td>
<td>1144</td>
</tr>
<tr>
<td>4</td>
<td>380.667</td>
<td>476</td>
<td>571.333</td>
<td>666.667</td>
<td>762</td>
<td>857.333</td>
<td>952.667</td>
<td>1048</td>
<td>1143.333</td>
<td>1238.667</td>
<td>1334</td>
<td>1429.333</td>
<td>1524.667</td>
<td>1620</td>
<td>1715.333</td>
<td>1810.667</td>
</tr>
</tbody>
</table>

Hogg's Method \((n = m)\)
Results – SAT Contd..

Worst Case: 3-SAT

<table>
<thead>
<tr>
<th>Hogg</th>
<th>651</th>
<th>1189</th>
<th>2239</th>
<th>4313</th>
<th>8435</th>
<th>16653</th>
<th>33063</th>
<th>65857</th>
<th>131419</th>
<th>262517</th>
<th>524687</th>
<th>1049001</th>
<th>2097603</th>
<th>4194781</th>
<th>8389111</th>
<th>1677774</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pk: (t=mn/3)</td>
<td>201</td>
<td>293</td>
<td>449</td>
<td>733</td>
<td>1273</td>
<td>2325</td>
<td>4401</td>
<td>8525</td>
<td>16745</td>
<td>33157</td>
<td>65953</td>
<td>131517</td>
<td>262617</td>
<td>524789</td>
<td>1049105</td>
<td>2097709</td>
</tr>
<tr>
<td>Pk: (t=mn/2)</td>
<td>206</td>
<td>299</td>
<td>456</td>
<td>741</td>
<td>1282</td>
<td>2335</td>
<td>4412</td>
<td>8537</td>
<td>16758</td>
<td>33171</td>
<td>65968</td>
<td>131533</td>
<td>262634</td>
<td>524807</td>
<td>1049124</td>
<td>2097729</td>
</tr>
<tr>
<td>Pk: (t=2mn/3)</td>
<td>211</td>
<td>305</td>
<td>463</td>
<td>749</td>
<td>1291</td>
<td>2345</td>
<td>4423</td>
<td>8549</td>
<td>16771</td>
<td>33185</td>
<td>65983</td>
<td>131549</td>
<td>262651</td>
<td>524825</td>
<td>1049143</td>
<td>2097749</td>
</tr>
</tbody>
</table>

Qubits: 3-SAT

<table>
<thead>
<tr>
<th>Hogg</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perkowski</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
</tbody>
</table>
Results: Graph Coloring

Graph Coloring - Gates - Best Case
- Hogg's best case
- Perkowski's Best Case

Graph Coloring - Gates - Worst Case
- Hogg's Worst case (e=2n)
- Perkowski's Worst Case

Graph Coloring - Qubits
- Hogg
- Perkowski
Results: Max Clique

Max Clique - Gates - Best Case
- Hogg's best case
- Perkowski's Best Case

Max Clique - Qubits
- Hogg
- Perkowski

Max Clique - Gates - Worst Case
- Hogg's Worst case
- Perkowski's Worst Case
### RESULTS: SEND MORE MONEY

<table>
<thead>
<tr>
<th>SEND MORE MONEY</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hogg's</td>
<td>Perkowski's</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Best Case</td>
<td>Best Case</td>
<td>Worst Case</td>
<td>Worst Case</td>
<td></td>
</tr>
<tr>
<td>Gates</td>
<td>65033</td>
<td>4628</td>
<td>1.0E+164</td>
<td>2.2E+12</td>
<td></td>
</tr>
<tr>
<td>Qbits</td>
<td>949</td>
<td>110</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SEND MORE MONEY - Gates - Best Case

- **Hogg's**: 65033
- **Perkowski's**: 4628

### SEND MORE MONEY - Gates - Worst Case

- **Hogg's**: 1.0E+164
- **Perkowski's**: 8.589938289E+31
Results: SAT
Conclusion

• I found that Perkowski’s method was much better than Hogg’s method when each variable can take on a large range of values.

• However, in cases where each variable could only take on a few values, like in Max Clique and SAT, they were closer together.

• In special cases of SAT, Hogg’s method was more efficient (in best case pulses only) than Perkowski’s method because Perkowski’s method requires more NOT gates.
Resources & Works Cited


Future Work

• Perform simulations of Grover Algorithm to test whether or not my mathematical formulas are correct

• Invent a new, and even more efficient oracle building method

• Further investigate of Nested Quantum Search (recursive application of Grover Algorithm)

• Investigate how the Shor Algorithm or the Bernstein – Vazirani Algorithm can be used for other problems
PRINT ONLY FIRST 25 SLIDES, NOT THE BACKUP

BACKUP
Max Clique - Perkowski

• Ensure that disconnected nodes are not activated
  – This will require a NAND (toffoli) gate, because no two unconnected nodes can both be one.
  – The total number of toffoli gates needed is the number of possible connections minus the number of actual connections.
  – The total possible connections is \( n(n-1)/2 \). We denote the total number of actual connections with \( e \). Thus we need a total of \( n(n-1)/2 - e \) toffoli gates, i.e., \( 13(n(n-1)/2 - e) \) pulses, and \( n(n-1)/2 - e \) ancilla qubits.

• Perform a global AND of the results
  – Since we need to verify \( n(n-1)/2 - e \) outputs of toffoli gates, we will need an \( (n(n-1)/2 - e) \)-bit toffoli gate at the end. This will cost \( 32(n(n-1)/2 - e) - 96 \) pulses and two qubits.
  – In the worst case, it will cost \( \frac{n(n-1)}{2} - e + 1 - 3 \) pulses and 1 qubit.
Max Clique - Hogg

• Hogg’s method is almost exactly like Perkowski’s method, except that it creates two qubits per node—one of these represents the activated node, and the other represents the inactivated node.
• If the “inactivated node” qubit of a certain node is zero, the node is activated. If it is one, the node is inactivated. The “activated node” qubit for a certain node is analogous to Perkowski’s qubits: it is one if the node is activated and zero if the node isn’t.
• Thus, we can simply duplicate Perkowski’s method for this oracle using the “activated node” qubits instead of Perkowski’s qubits.
• There is only one important difference, and that is that Hogg’s method needs to make sure a qubit isn’t both activated and inactivated.
• This will cost n more NAND gates and n extra inputs into the final toffoli gate, or $13n + 32n = 45n$ more pulses, and $2n$ extra qubits (n for each additional NAND gate and n for each additional starting qubit). Thus, for this problem, Hogg’s method is virtually obsolete.
Satisfiability - Perkowski

• For Perkowski’s method the oracle would be very simple: for each term of the equation you would have an OR gate for the n variables, and then a global AND for all the terms- just like in the formulation of the problem.

• A large scale OR gate can be created in quantum technology by using a toffoli gate with all the inputs NOTed before and after the gate (to restore original values).
  – Note that for the term a’, we would not have to include an extra NOT before the Toffoli, because we already are NOT-ing that input.

• The cost of this circuit would be m n-bit toffoli gates, plus 2t NOT gates (t is the total of un NOT-ed terms), and then a final m-bit toffoli at the end. Since n is usually small, I will be using the $2^{m+1}$ estimate for OR gates. Thus the best case cost is $m*(2^{n+1}-3)+2t+32m-96$ pulses and $v+m+2$ qubits.

• The worst case cost is $m*(2^{n+1}-3)+2t+2^{m+1}-3$ pulses
Satisfiability - Hogg

• For Hogg’s method, the circuit will be similar to Perkowski’s method, except that no NOT gates would be needed.

• Since we create a 1 and 0 (regular and NOT-ed) qubit for every variable, the un-NOT-ed variable can be represented by the 1 qubit, and the NOT-ed variable can be represented by the 0 qubit.

• However, there will be $2v$ qubits instead of $v$ ($v$ is the total number of variables) qubits in Perkowski’s method.

• Thus the cost is $10v + m^*(2^{n+1}-3) + 32(m+v) - 96$ pulses or $10v + m(2^{n+1}-3) + 2^{m+v+1}-3$ pulses and $3v + m + 2$ qubits.

• The most efficient method in this case largely depends on $t$ and $v$. 
• I will make sure that no two letters are the same value, which will require inequality gates
  • Since there are 9 possible values for the 8 letters, each letter will get four bits. There will be 32 qubits for the letters.
  • Since there are 8 letters, there will have to by $8(8-1)/2 = 28$ inequality gates.
  • Since there are four bits for each letter, the inequality gates will cost four Feynman gates, four NOT gates, a four-bit toffoli (which, when costing no garbage bits, costs 61 pulses), and an ancilla bit.
  • Thus the cost is $28(4*5+4*1+61)$ = **2380 pulses** and **28 ancilla bits**
• I will test the four equations with quantum gates. This will require quantum adder gates, and it will require us to multiply carry gates by 10.
  • In binary, 10 is 1010. With this in mind, $c*1010 = c0c0$, given that $c$ is a one digit binary number (which, incidentally, all the carries are). Thus, all we will have to do is create four ancilla bits, and transfer our preferred “carry” value to two of them using a two Feynman gates. Thus multiplying a carry by 10 will cost 4 qubits and 10 pulses each.
  • We will also have to use a network of quantum adder gates to perform each of our additions. Each network of adder gates, including the multipliers, costs 330 pulses and 12 qubits.
  • Since this must be repeated for four equations, the overall cost will be **1320 pulses** and **48 qubits**. Four of these will be output qubits.
• I will create a global AND at the end
  – We will need to verify a total of $28+4 = 32$ outputs. Thus, the global AND will cost $32(32)-96 = 928$ pulses and **2 bits**.
  – Calculating the worst case cost, we have $2^{32+1}-3 = 8.59 \times 10^9$ pulses
• The overall cost of the circuit is **4628 pulses** and **110 qubits**.
• If we use the worst case estimate, we will have **8589938289** pulses.
Hogg’s method for this problem is very, very inefficient. Since Hogg’s variables can only take on values of one and zero, and not the value of the actual assignment that was made to the letter, one cannot use quantum adder gates. Instead, one has to come up with all possible solutions to the equations above by themselves, and then input these solutions into one large master circuit.

For example, the equation \( D+E = 10^*c1+Y \) can be solved by the assignments (for \( D, E, Y, c1 \) respectively) 1,2,3,0; 1,3,4,0; 1,4,5,0; etc. In fact, there are 80 such sets for the first equation. Since there is an additional carry in the next terms, there are 160 such sets for the second and third equations. For the last equation, there is only one set of assignments that satisfies the equation, so there is only one set. For Hogg’s method we need the following:

- We need gates to make sure that one variable does not have two values, and that no two variables are given the same value
  - This is very similar to what was done in graph coloring. It will take \( n(n-1)/2 \) NAND gates, where \( n \) is one of nine values. Thus, there are \( 9(8)/2 = 36 \) NAND gates. This procedure is repeated 8 times (one per letter) for a total of 288 toffoli gates, or 3744 pulses and 288 ancilla bits.
  - To test that no two letters are assigned the same number, we need \( n(n-1)/2 \) NAND gates again, except that this time, we have \( n = 8 \), because there is one of each kind of assignment per letter. Since this procedure is repeated 9 times (once per number), we have \( 9*8(7)/2 = 252 \) toffoli gates, or 3276 pulses and 252 ancilla bits.

- We also need gates to make sure that at least one of the possible conditions discussed above is met
  - For the first equation of those discussed above, there are 80 possible conditions. Since there are four constraints in each of these possible solutions, we will need 80 4-control toffoli gates, one for each of the possible sets. This will cost 61*80 = 4880 pulses and 80 ancilla bits.
  - For the next two sets of terms, we have 5 constraints and 160 possible solution sets. Thus, we will need 160 5-control toffoli gates, or 125 * 160 = 20,000 pulses and 160 ancilla bits. Because this is being repeated twice, we have 40,000 pulses and 320 ancilla bits total.
  - Since there is only one possible solution to the third equation, we only need one 5-control toffoli gate, or 125 pulses and 1 ancilla bit.
  - To make sure that at least one of the possible conditions are satisfied after the Toffoli gates, we need to use a large quantum OR gate. As discussed earlier, an quantum OR gate costs \( n+32n – 96 = 33n – 96 \) pulses and two qubits. Since we have to apply a quantum OR gate to 3 sets of inputs, we have \( 33(80) – 96 + 2( 33(160) – 96 ) = 2640 + 10368 = 13008 \) pulses and 6 qubits. Three of these are output qubits.
  - A final AND gate. This will have \( 4+288+252 = 544 \) inputs. Thus it will cost 32(544) – 96 = 17312 pulses and 2 qubits.

- Overall, we calculate 65033 pulses and 949 qubits for best case.
- In the worst case, we will have \( 10^{165} + 47718 \) pulses.