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

By Sidharth Dhawan

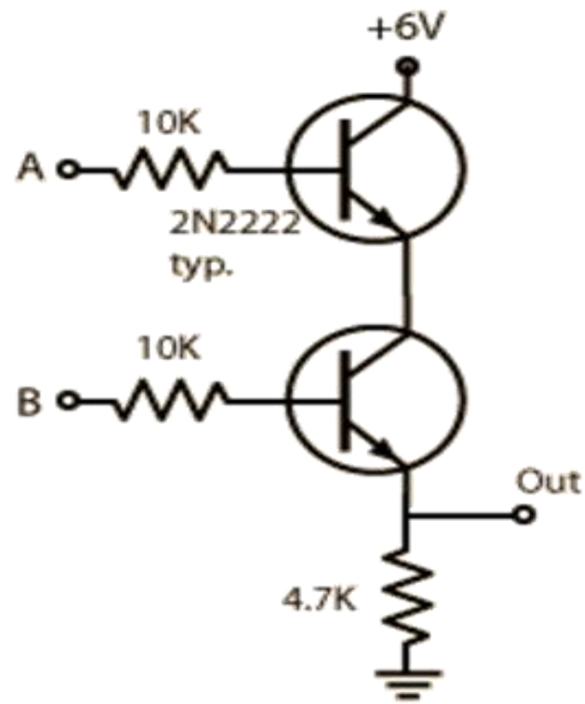
GROVER ALGORITHM

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.



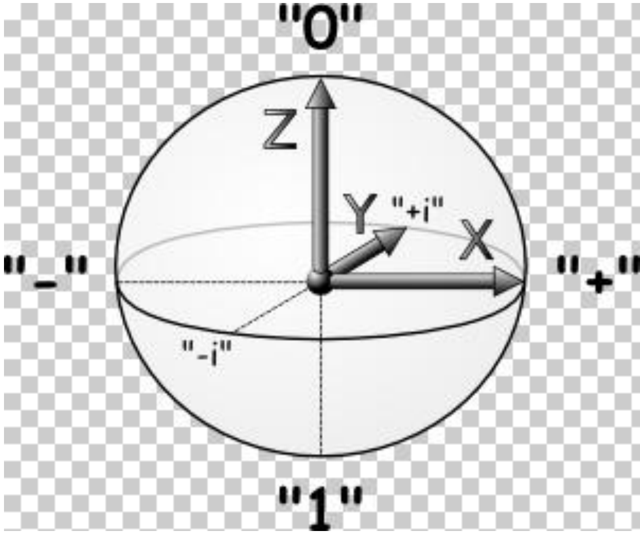
Superposition and Quantum Computers

- 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"



The Bloch Sphere represents all possible quantum states.

Superposition and Search

- We can use the **Hadamard transform** to create an even superposition between the |0> and |1> 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>, |01>, |10>, and |11>.

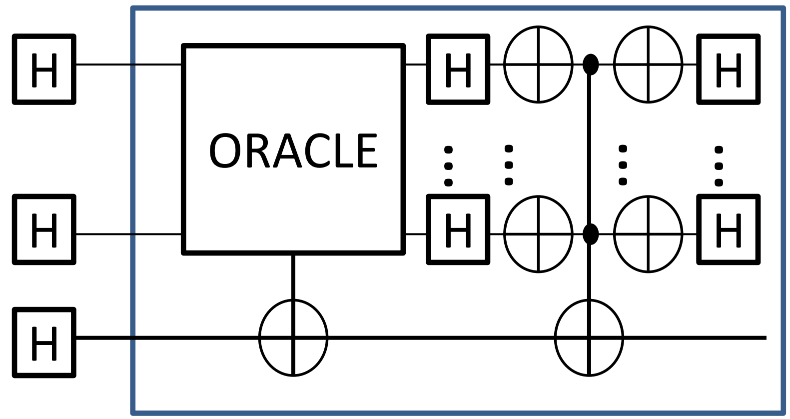
- For this reason, quantum bits store more information than classical bits – I can represent 2^n classical states with n qubits

Quantum Unstructured Search ("Grover")

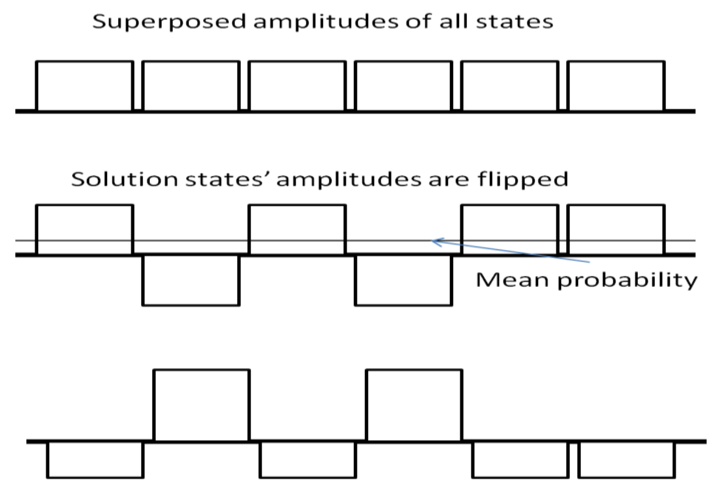
- **Grover's Algorithm** uses this property of quantum information to perform an unstructured search more quickly

- 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 √N times (as opposed to N iterations in classical logic) to maximally amplify the states.

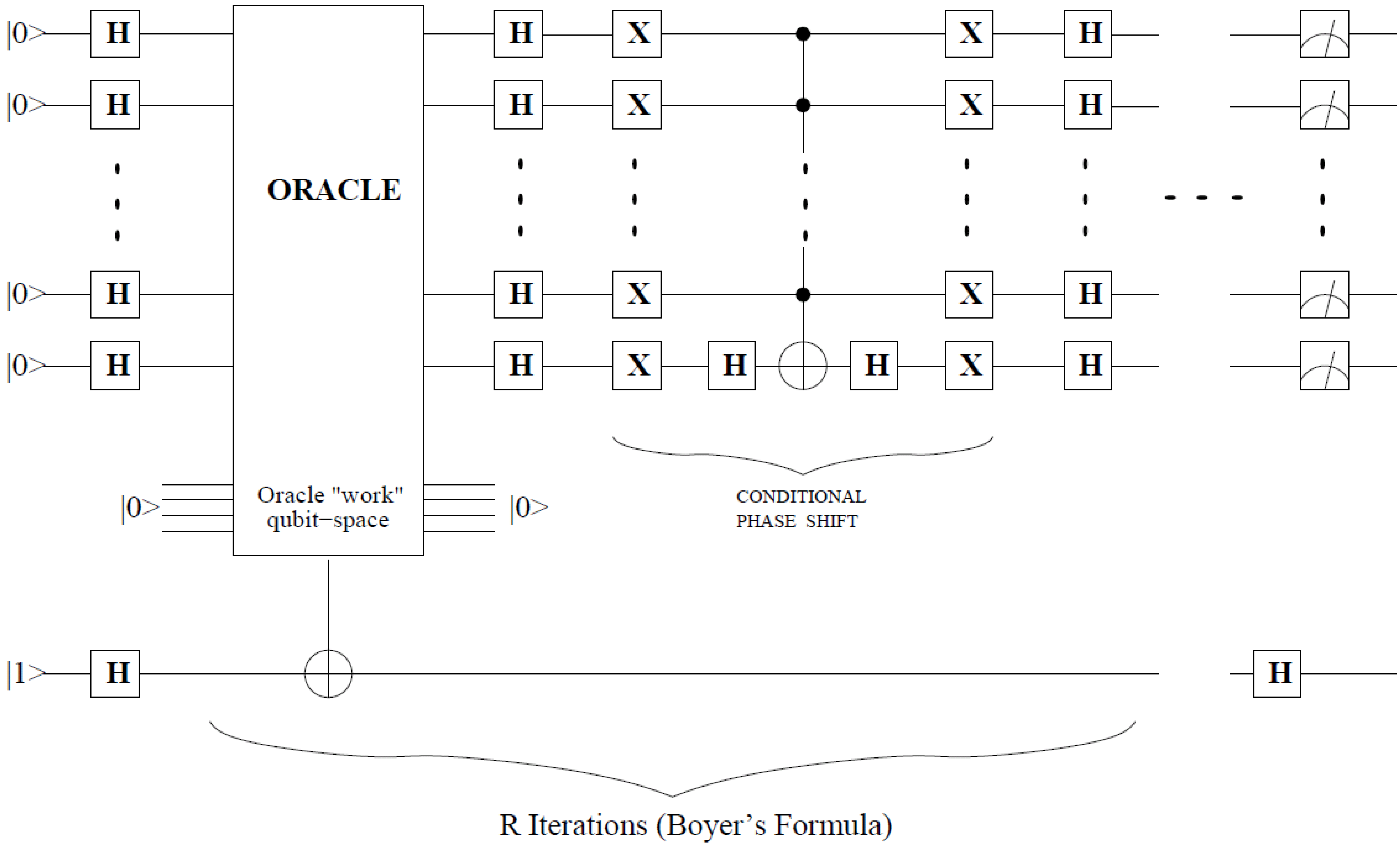


Grover Loop (* √N)



Solution states are amplified, while non-solutions are less probable

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 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.

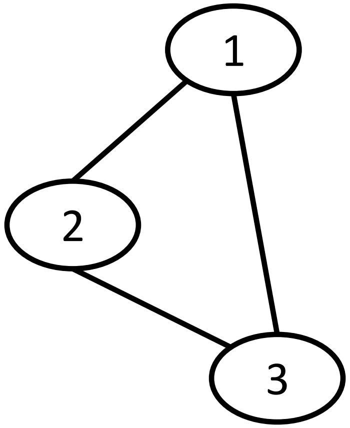
- For example, the Grover Circuit for SEND MORE MONEY costs 17 thousand trillion trillion more basic gates with the less efficient method

Graph Coloring

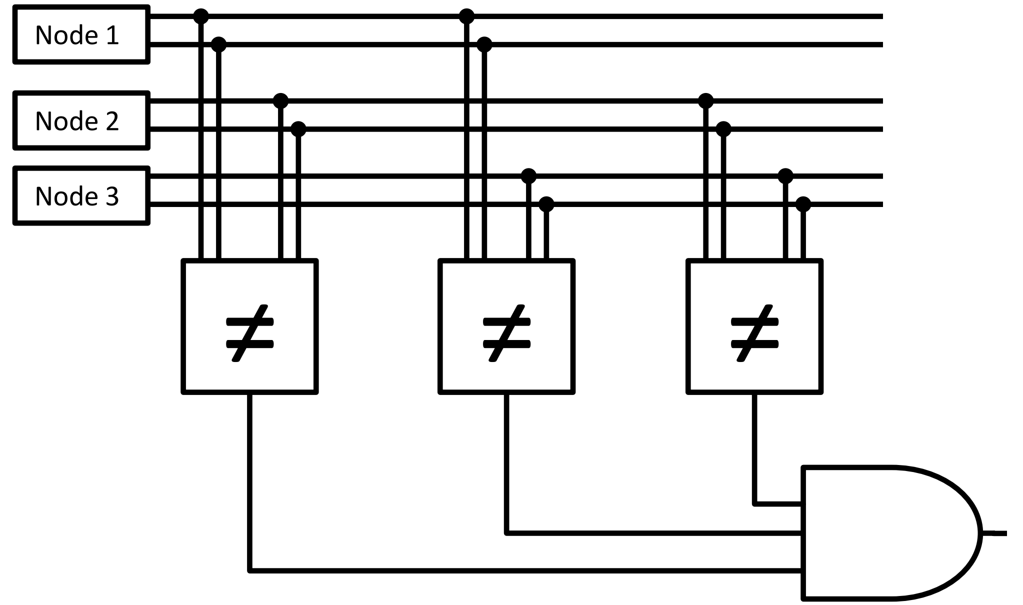
- Graph coloring is an NP complete problem

- 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.

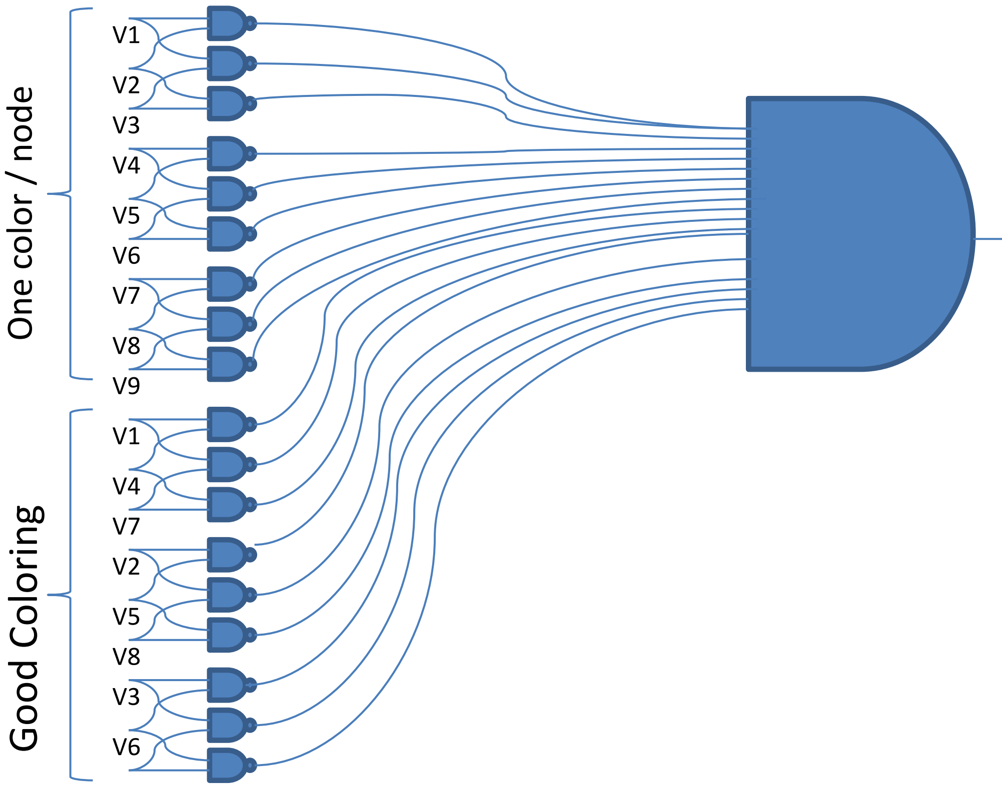
- For example, the states|00> and |01> represent different colors.

- 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

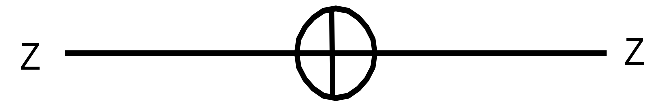
- In this method, each assignment of a color to a node is represented by a qubit, v1-v9.
- No two elements of a row can coexist, because only one color can be assigned to a node. I use NAND gates to ensure this
- If, for example, nodes one and two are connected, we must do: v1 NAND v4, v2 NAND v5, and v3 NAND v6.

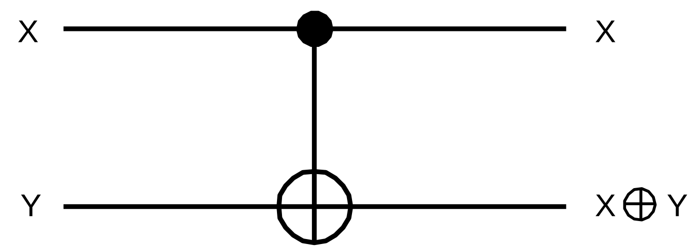
Colors → **Nodes**	**A**	**B**	**C**
1	V1	V2	V3
2	V4	V5	V6
3	V7	V8	V9

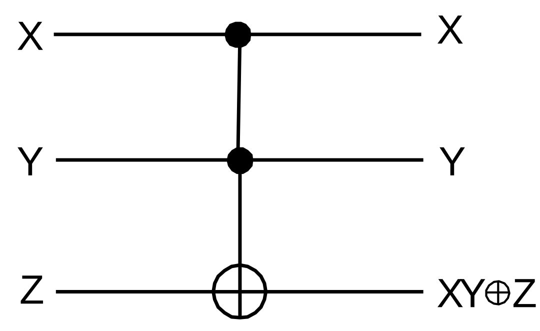


Reversible Logic and Quantum Cost

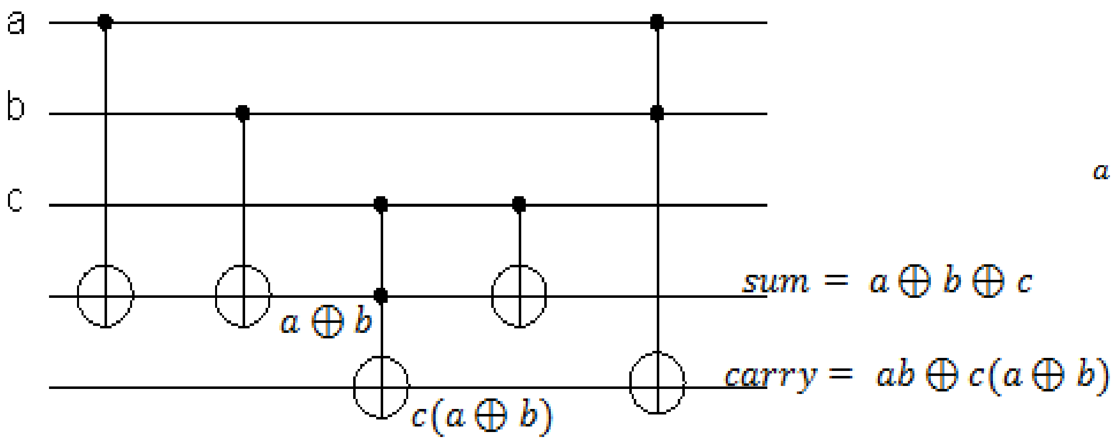
- An alternative classical logic implementation is called AND – EXOR logic.
- It is reversible because you can determine inputs from the outputs.
- This kind of logic is easier to simulate with most quantum technologies.

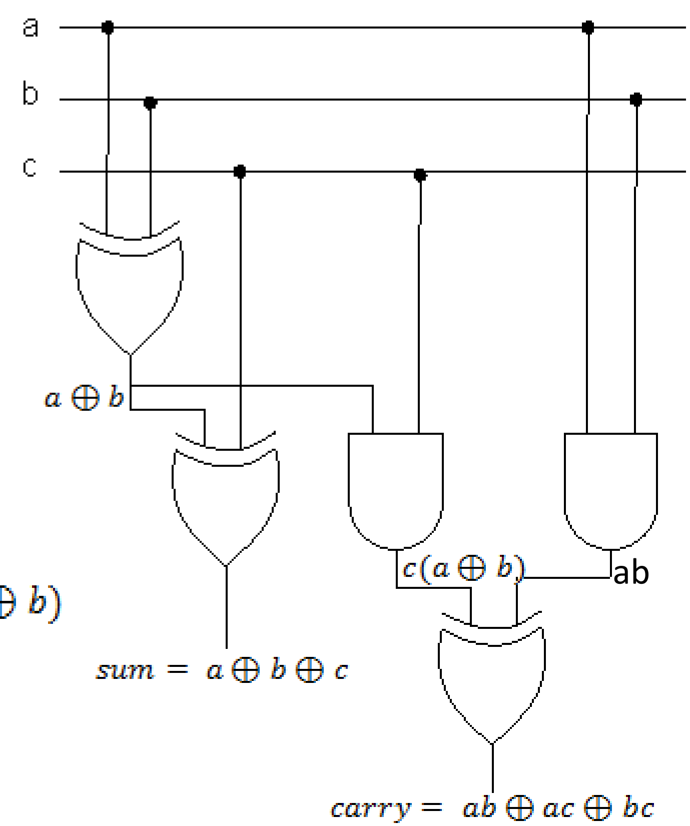






Reversible and Non Reversible Circuits





Quantum Cost

Gates	**Cost in Basic Gates**
Quantum NOT	1
Hadamard Gate	1
CNOT Gate	1
3 input Toffoli Gate	5
N input Toffoli Gate	Best Case: 32n-96 Worst Case: (2^{n+1}) -3

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

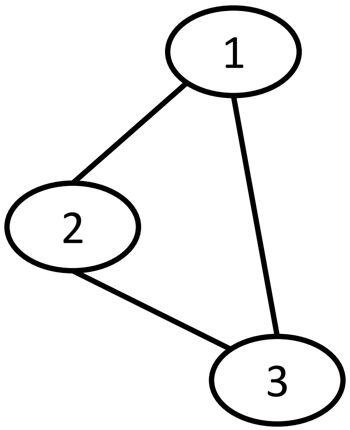
In this paper, the costs of these two data-encoding methods for building Grover Oracles are compared by testing both for four problems:

- Satisfiability (SAT),
- Maximum Clique,
- SEND MORE MONEY
- Graph Coloring

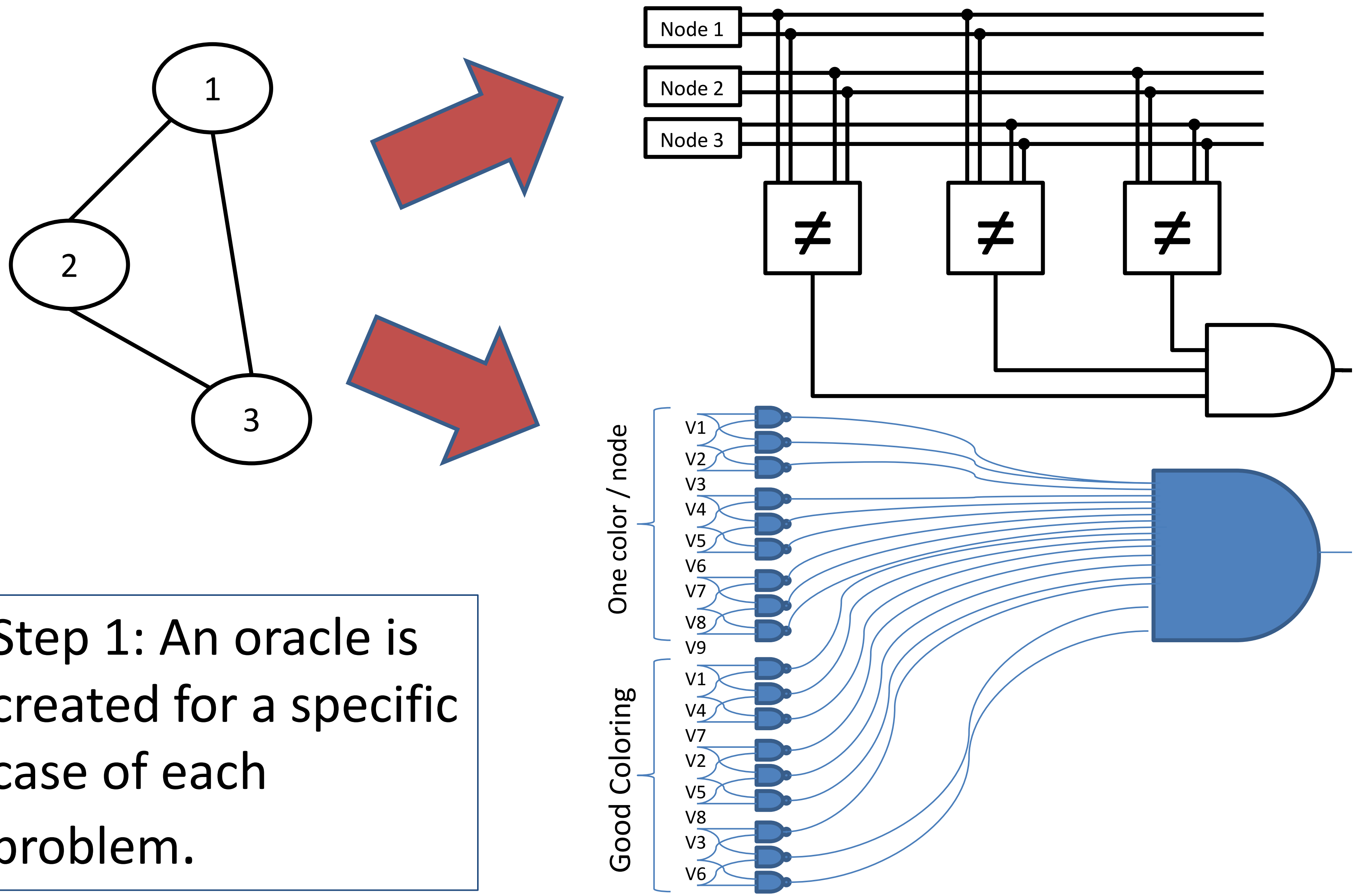
COST DERIVATION FOR GRAPH COLORING

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



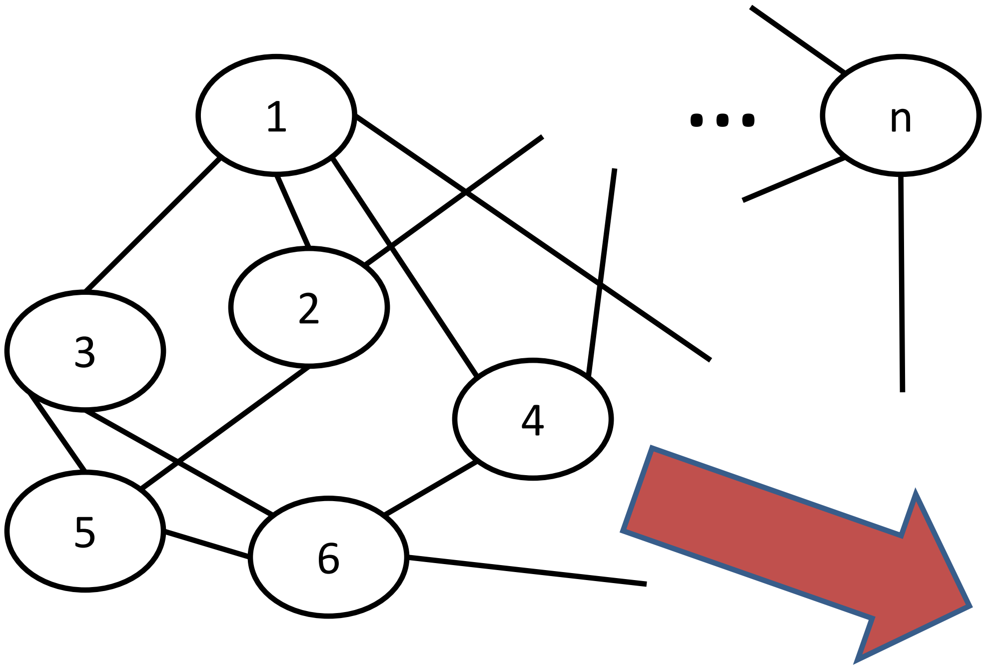
Cost Derivation for Graph Coloring

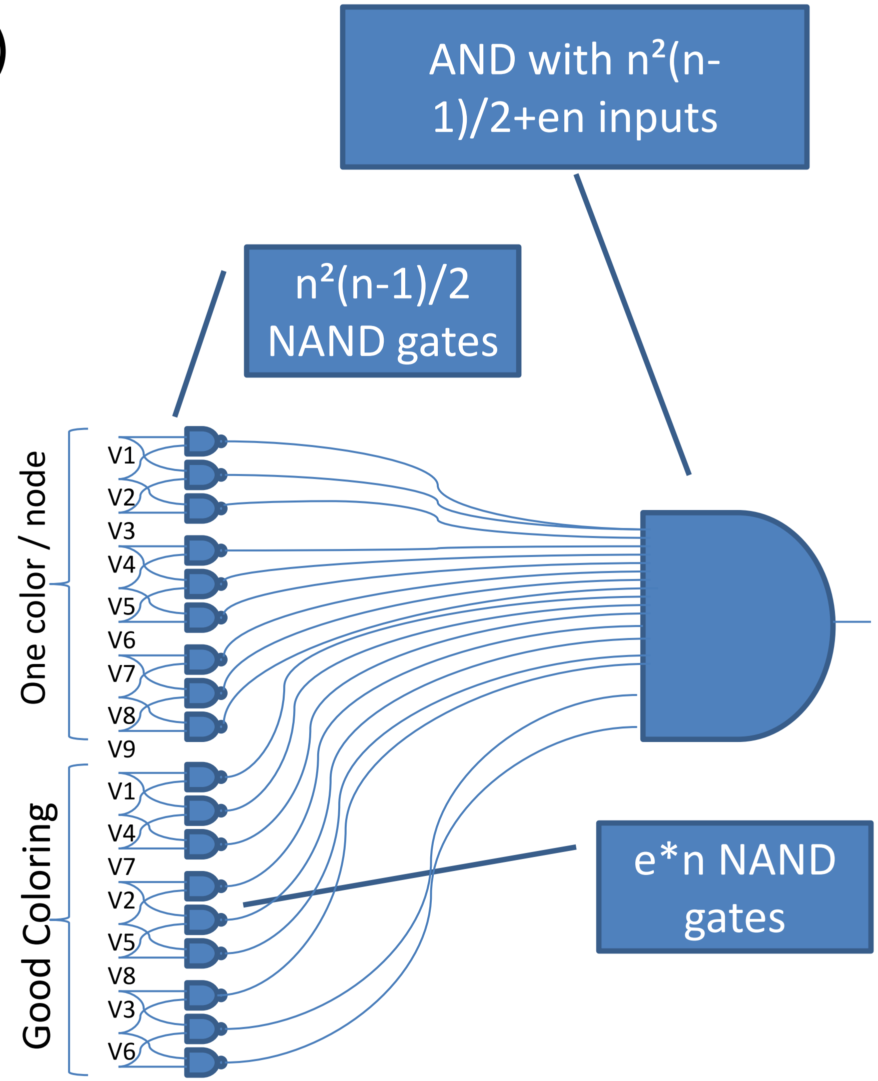


Step 1: An oracle is created for a specific case of each problem.

Colors → **Nodes**	**A**	**B**	**C**
1	V1	V2	V3
2	V4	V5	V6
3	V7	V8	V9

Cost Derivation for Graph Coloring





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

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

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

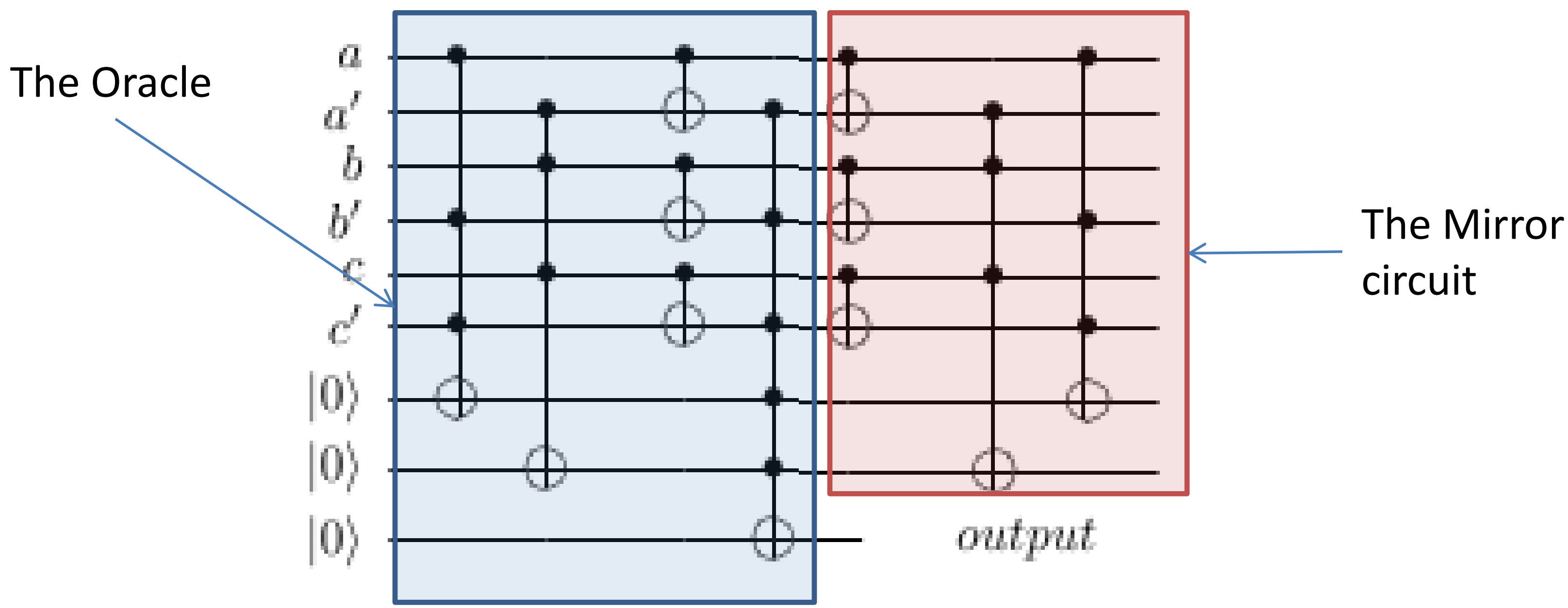
Total Cost:
(2*STEP 2+ STEP 4)=
$42(en+n^2(n-1)/2) – 96$
Basic gates

Step 3: Convert gates to cost with cost formulas

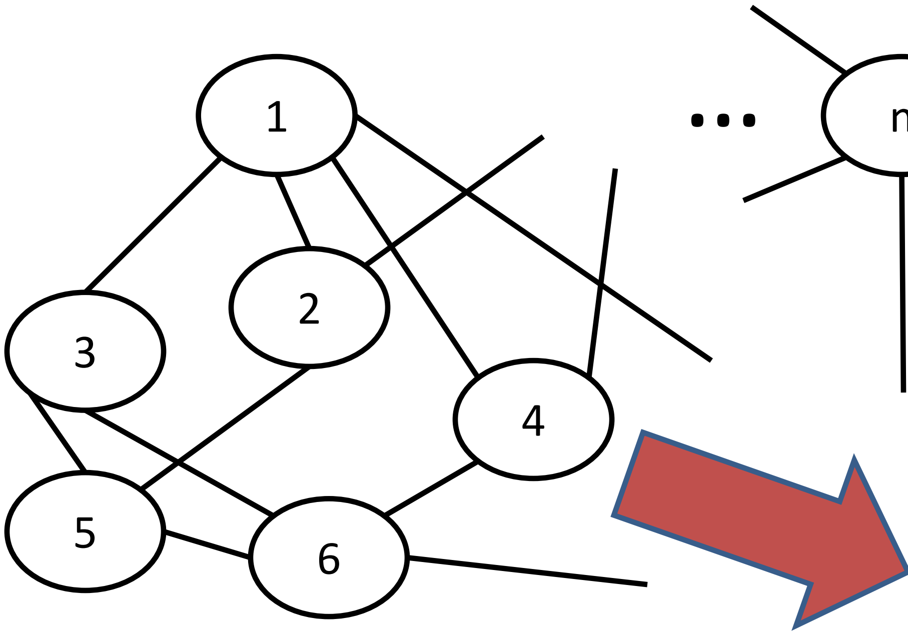
The Mirror Circuit

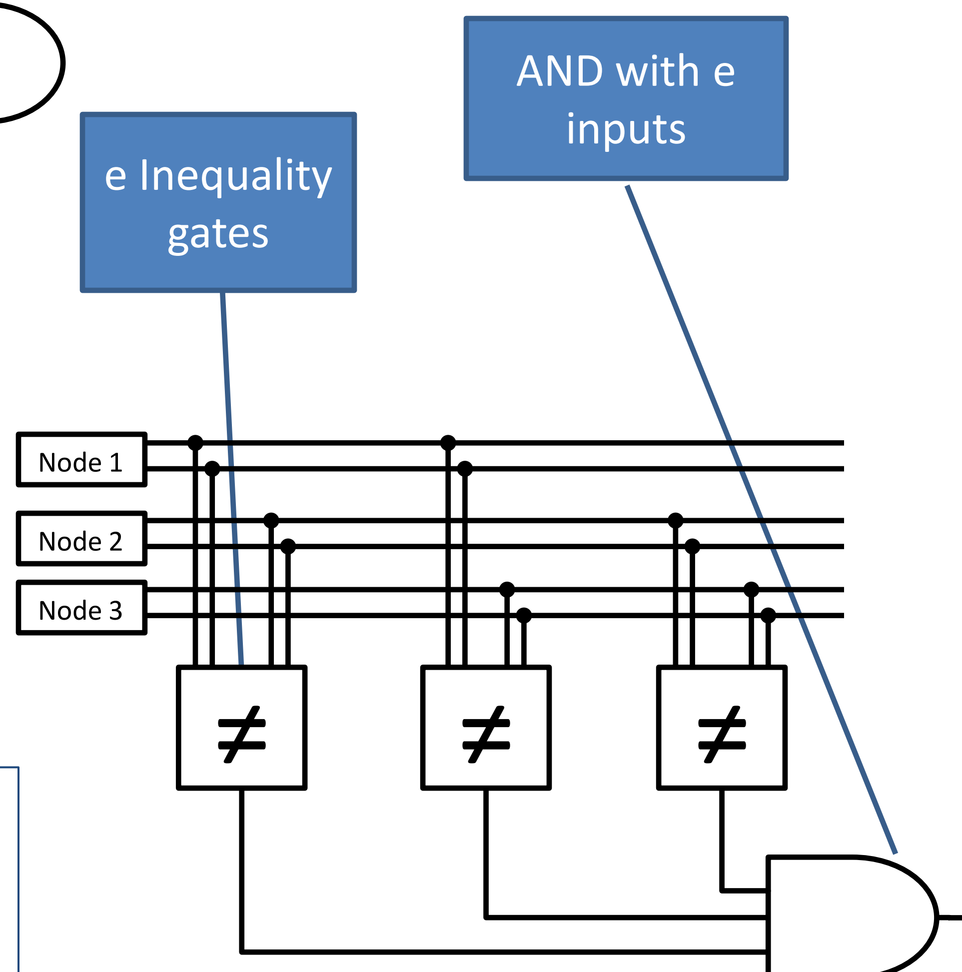
- At the end of the oracle, the qubits in the "Work space" must be returned to their original states for the next oracle operation.

- 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



Cost Derivation – Perkowski's Method

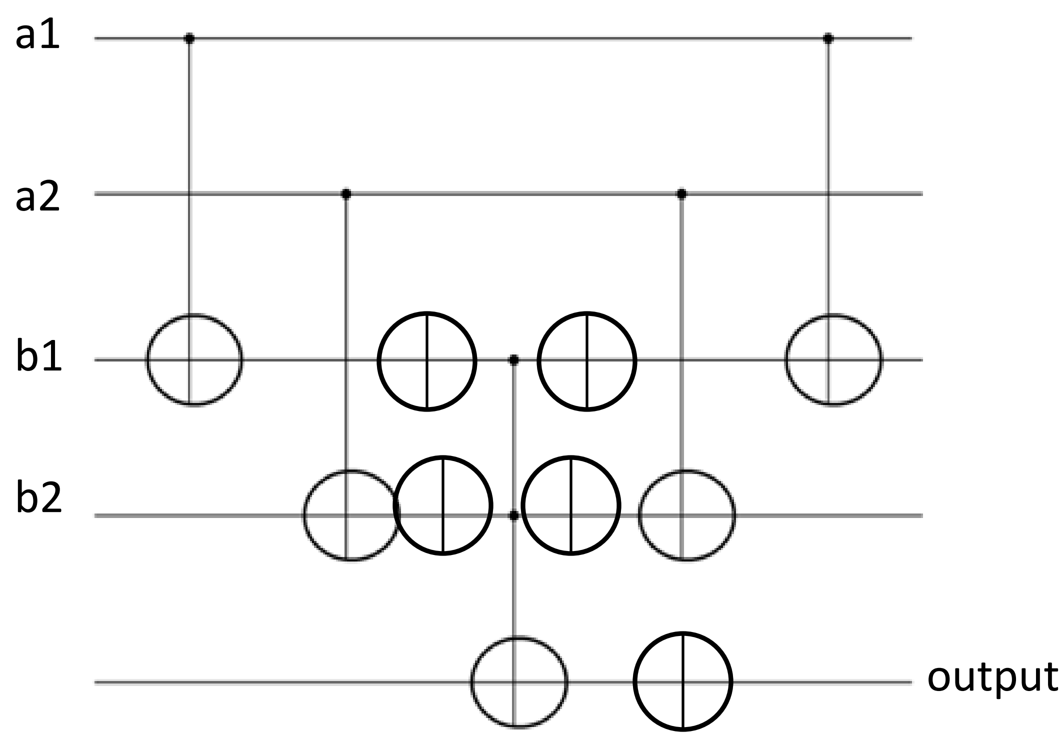




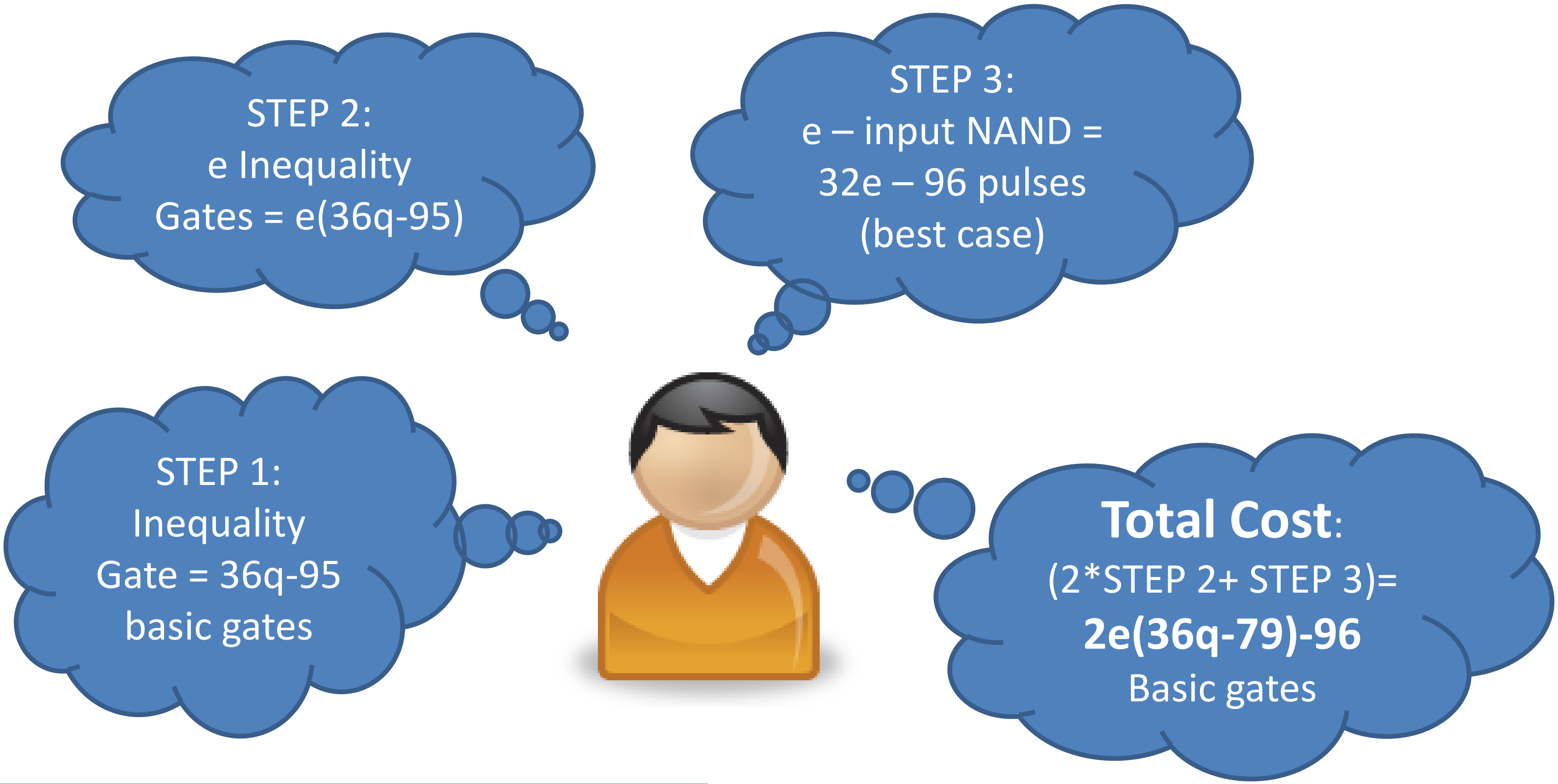
Step 2: How many gates are needed in a generic case with n nodes and e edges?

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**



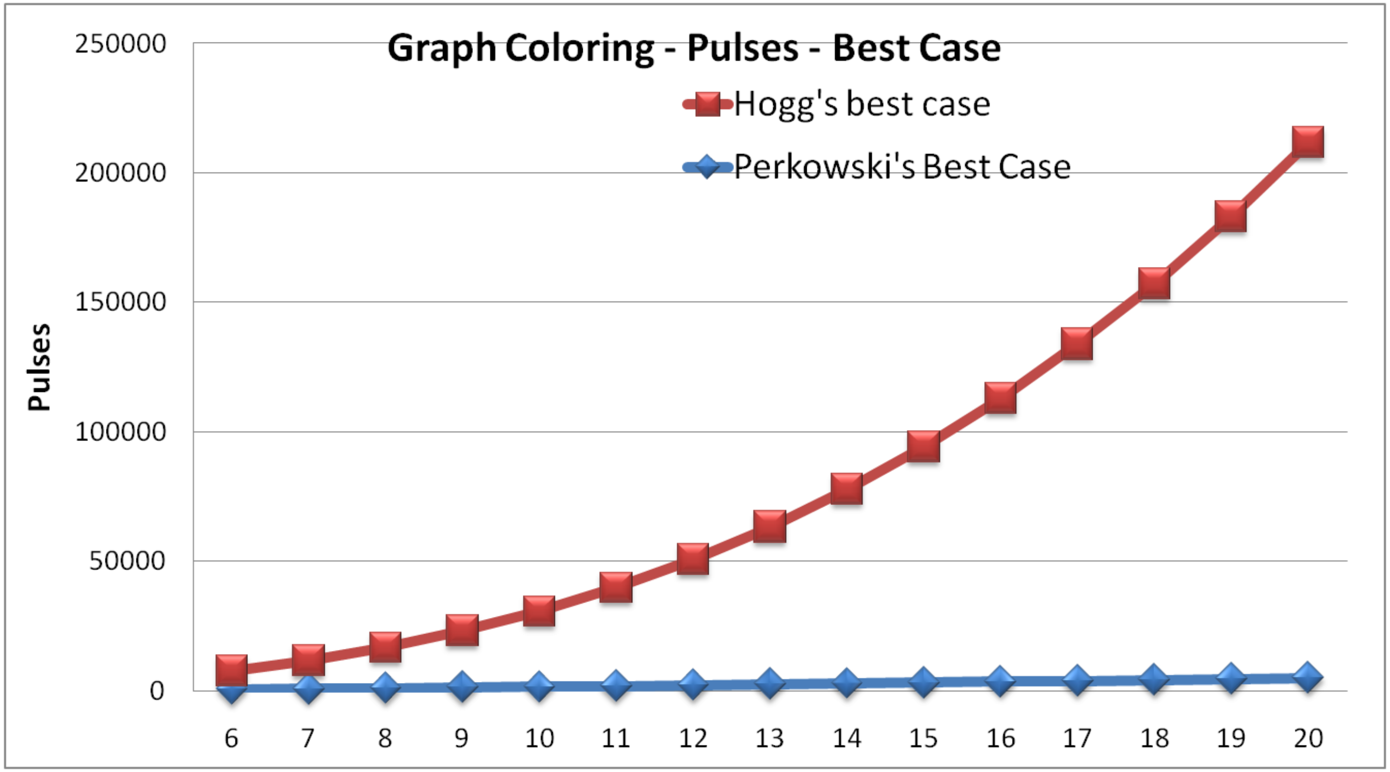
Cost Derivation – Perkowski's Method



Step 3: Convert gates to cost with cost formulas

Cost Derivation for Graph Coloring

$$42\left(en+\frac{n^2(n-1)}{2}\right)-96$$



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

$$\begin{array}{r} S\ E\ N\ D \\ +\ M\ O\ R\ E \\ \hline M\ O\ N\ E\ Y \end{array}$$

D + E = 10*C1 + Y

C1 + N + R = 10*C2 + E

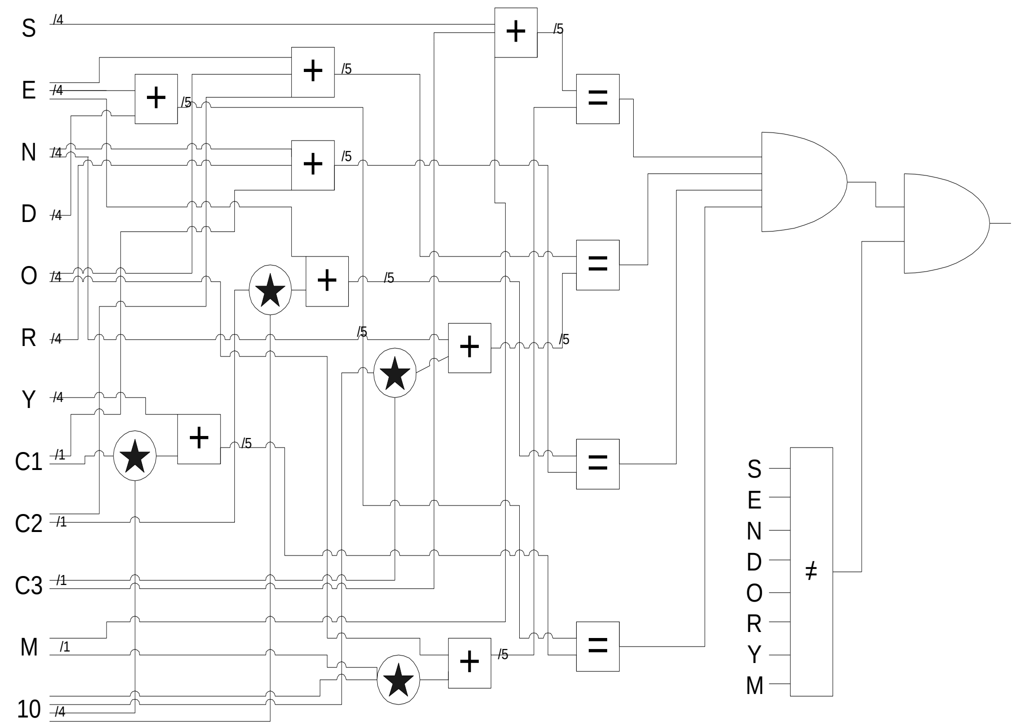
C2 + E + O = 10*C3 + N

C3 + S + M =10*M + O

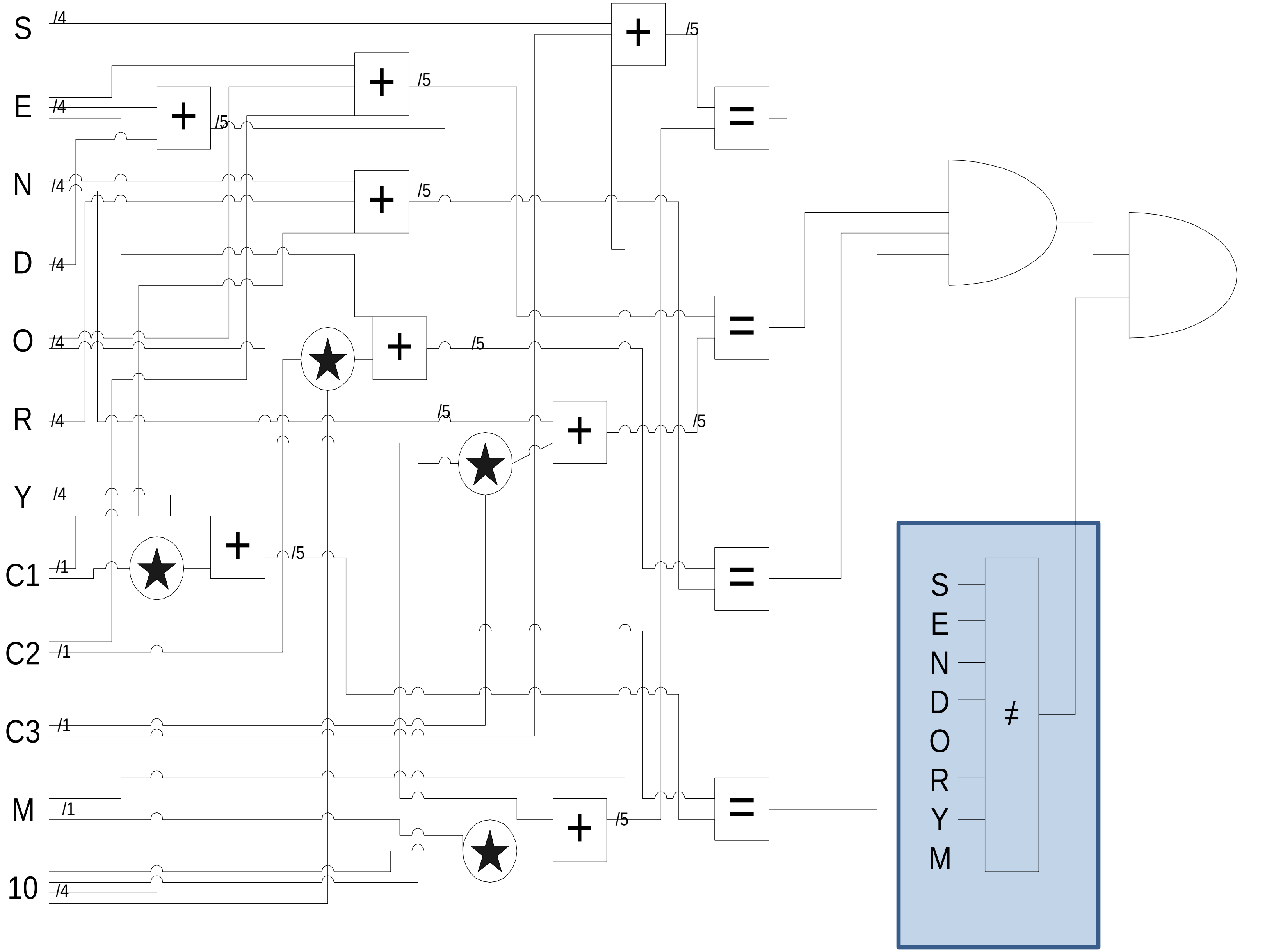
SEND MORE MONEY - Perkowski

<u>Perkowski's method:</u>

- The oracle on the right uses Perkowski's method to solve the problem

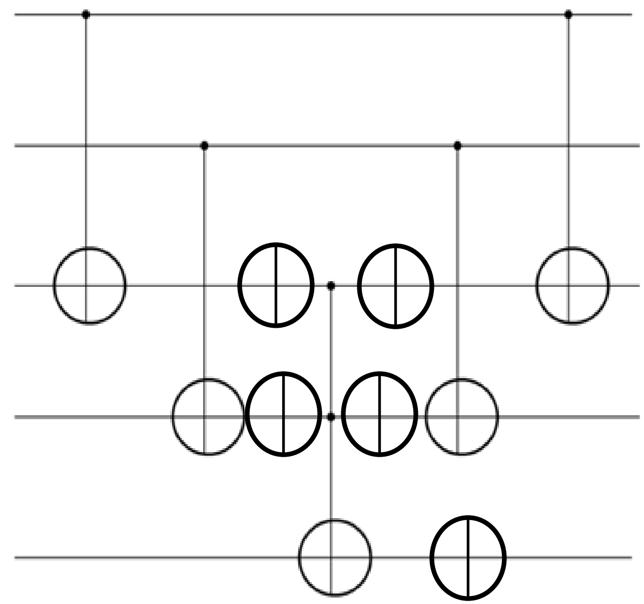


- Each letter is represented by four qubits and can take on ten values.
- This emulates the formulas with gates
  - for example, the conditions D + E and 10*c1+Y are plugged into the bottom equality gate
- Best case cost:
  5,186 basic gates and 126 Qubits


S
E
N
D
O
R
Y
C1
C2
C3
M
10
/4
/1
/5
+
=
S
E
N
D
O
R
Y
M
≠

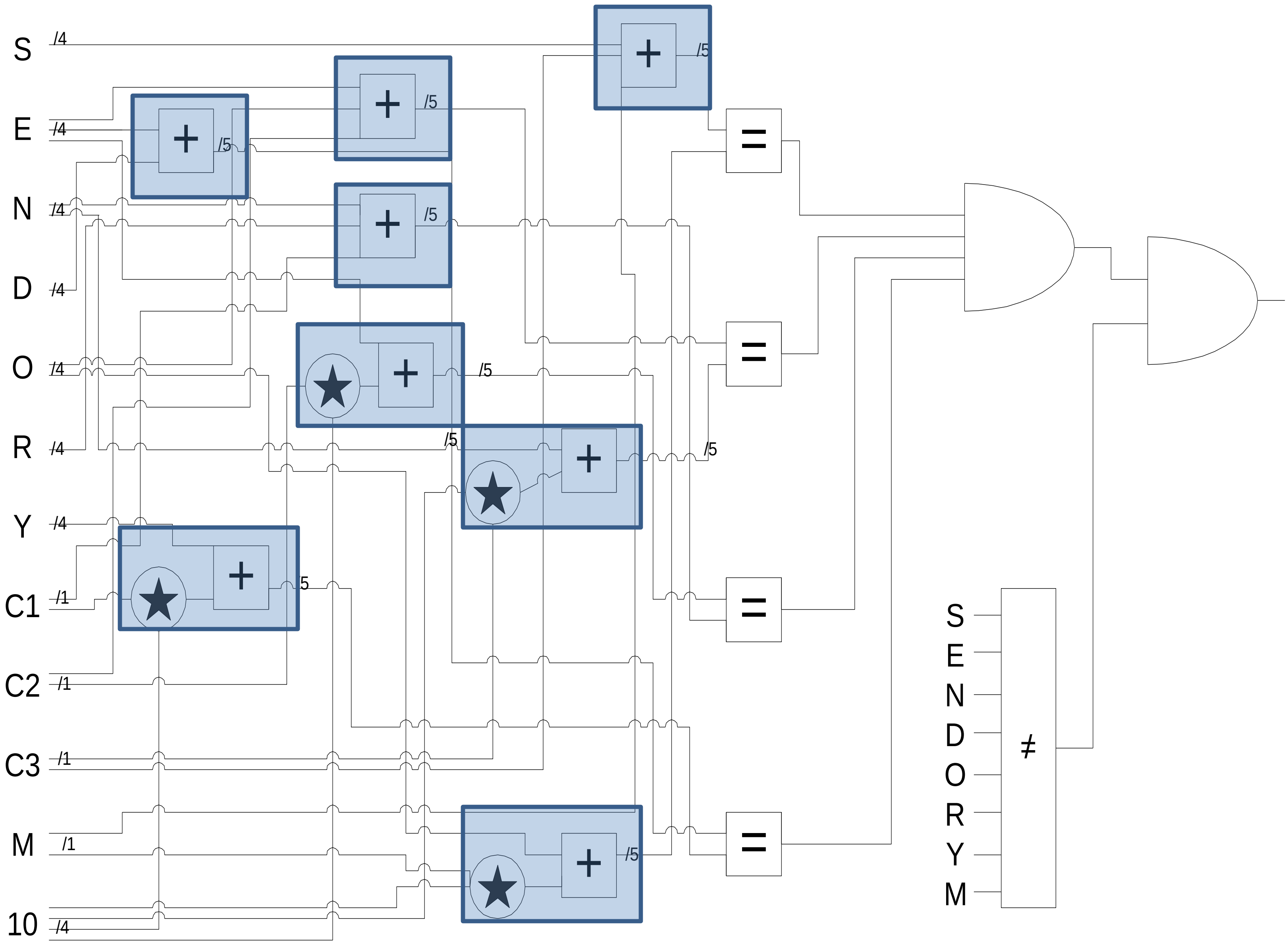
Inequality Gates

- 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 Feynmans 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 * (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 (a1*(a2+a3))',
  - 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.**



S
E
N
D
O
R
Y
C1
C2
C3
M
10
/4
/1
/5
+
=
≠

$$D + E = 10*C1 + Y$$

$$C1 + N + R = 10*C2 + E$$

$$C2 + E + O = 10*C3 + N$$

$$C3 + S + M = 10*M + O$$

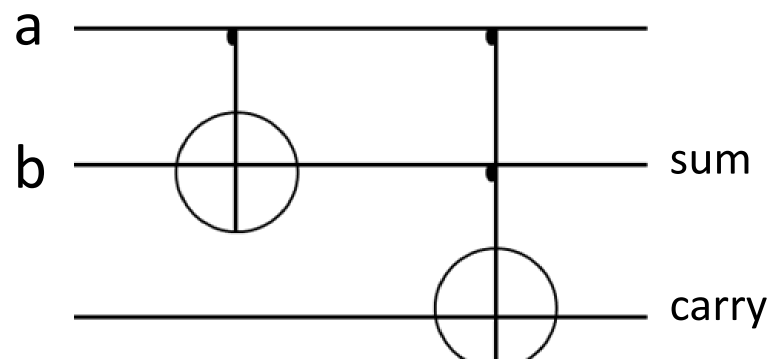
There are two types of equations that must be described with adder circuits.

For example, there is c1+ N+ R, which requires full adders.

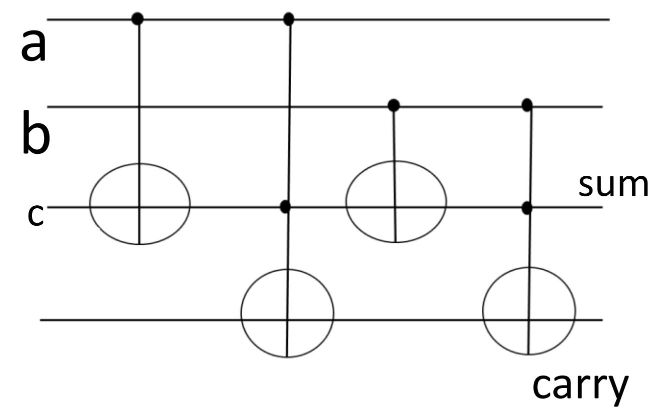
Also, there is 10*c1 + Y, which must be described using half-adders

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.

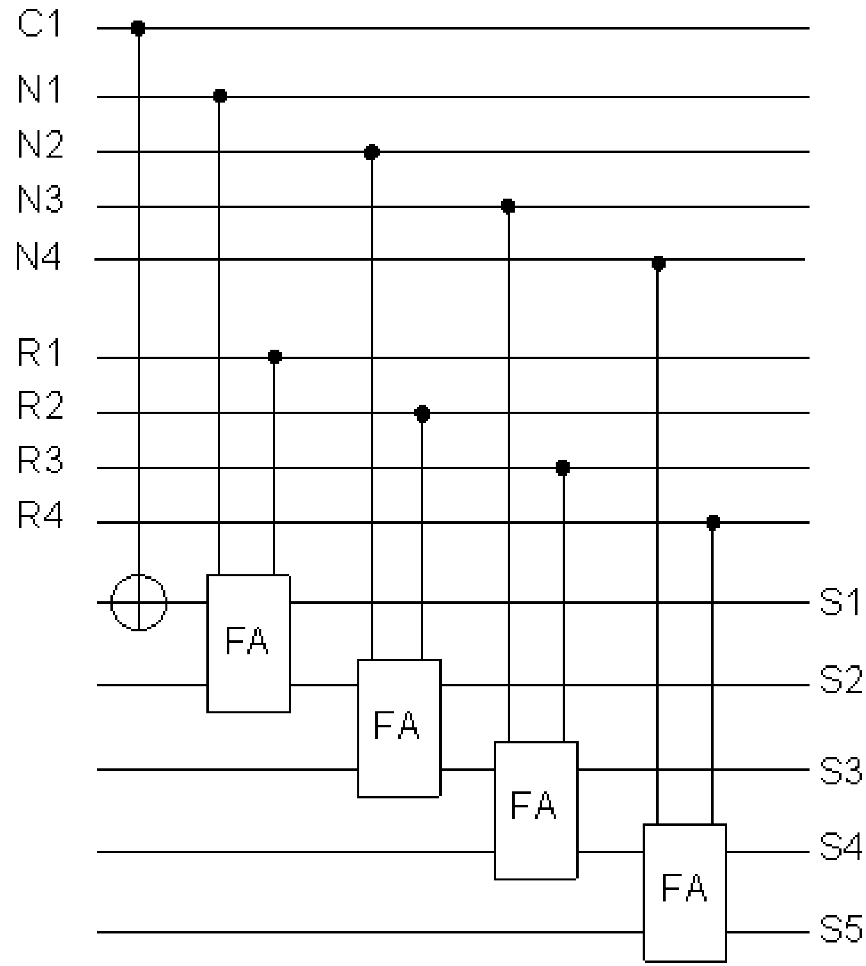


a\b	0	1
0	00	01
1	01	11

a\bc	00	01	11	10
0	00	01	10	01
1	01	10	11	10

Adder Tree

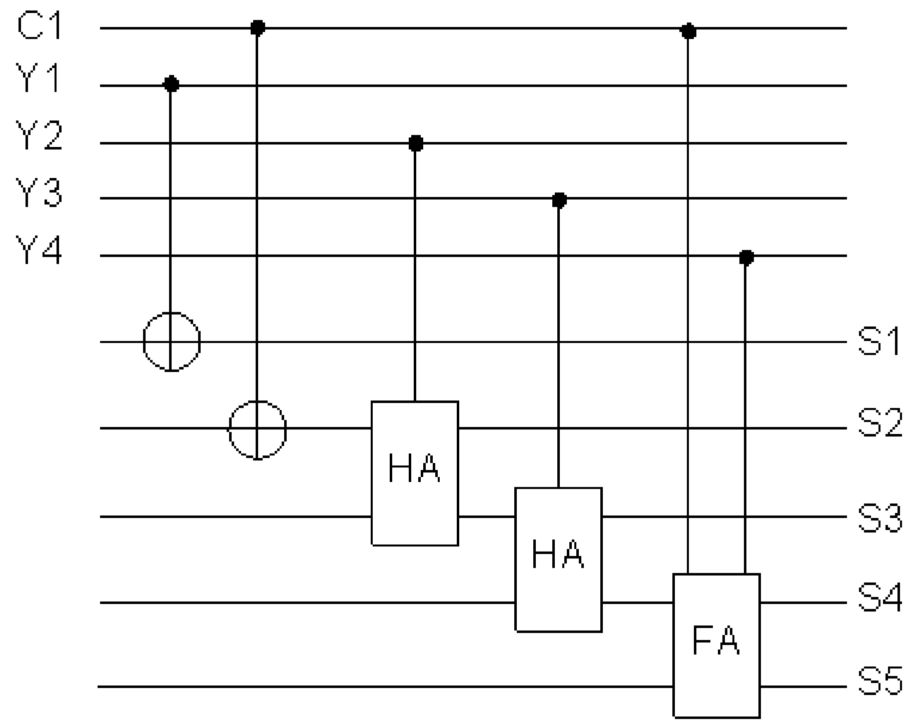
C1+N+R:



- 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*c1 + Y

- In binary, 10 = 1010.
- Since all the carries are one qubit, 10*c1 = c1, 0, c1, 0.

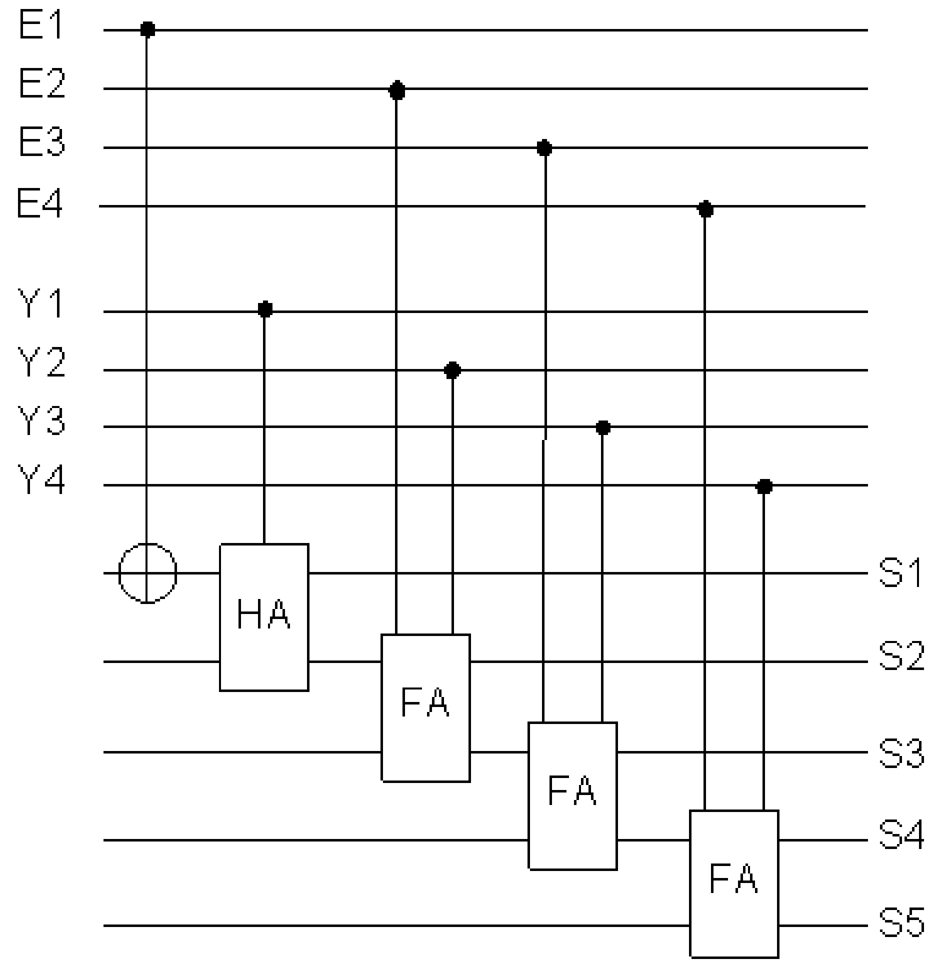


- Thus, S1 will equal Y1
- C1+Y2 will be a half-adder, because only two qubits are being added
- C1 only needs to be added to Y2 and Y4, thus, the only Full adder will come at the end, because this is the only time three qubits are being added.
- Thus, the cost is 2 feynmans, two half-adders, and a full 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.

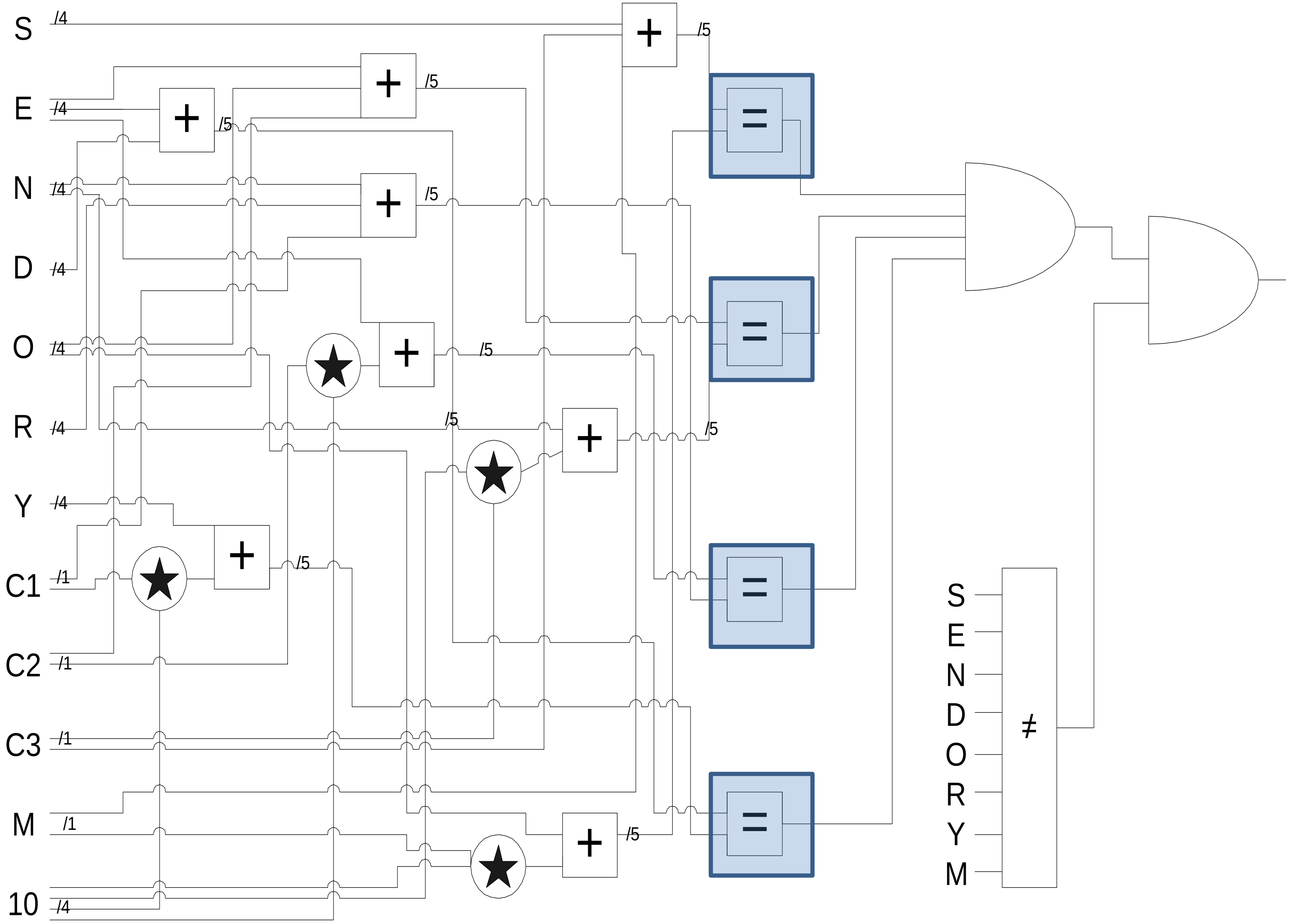
- Thus, the cost is **43 basic gates**



Total Cost of adders

- There are three equations of type: c1+N+R
- There are four equations of type: 10*c1+Y
- There is one equation of type: E+D
- Thus, total cost of all adders is:

  3(49) + 4(26) + (43) = **294 basic gates.**



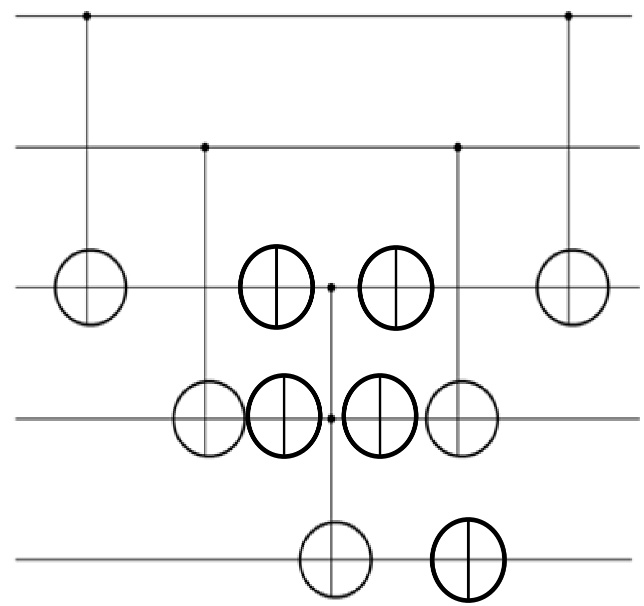
S
E
N
D
O
R
Y
C1
C2
C3
M
10
/4
/1
/5
+
=
★
≠

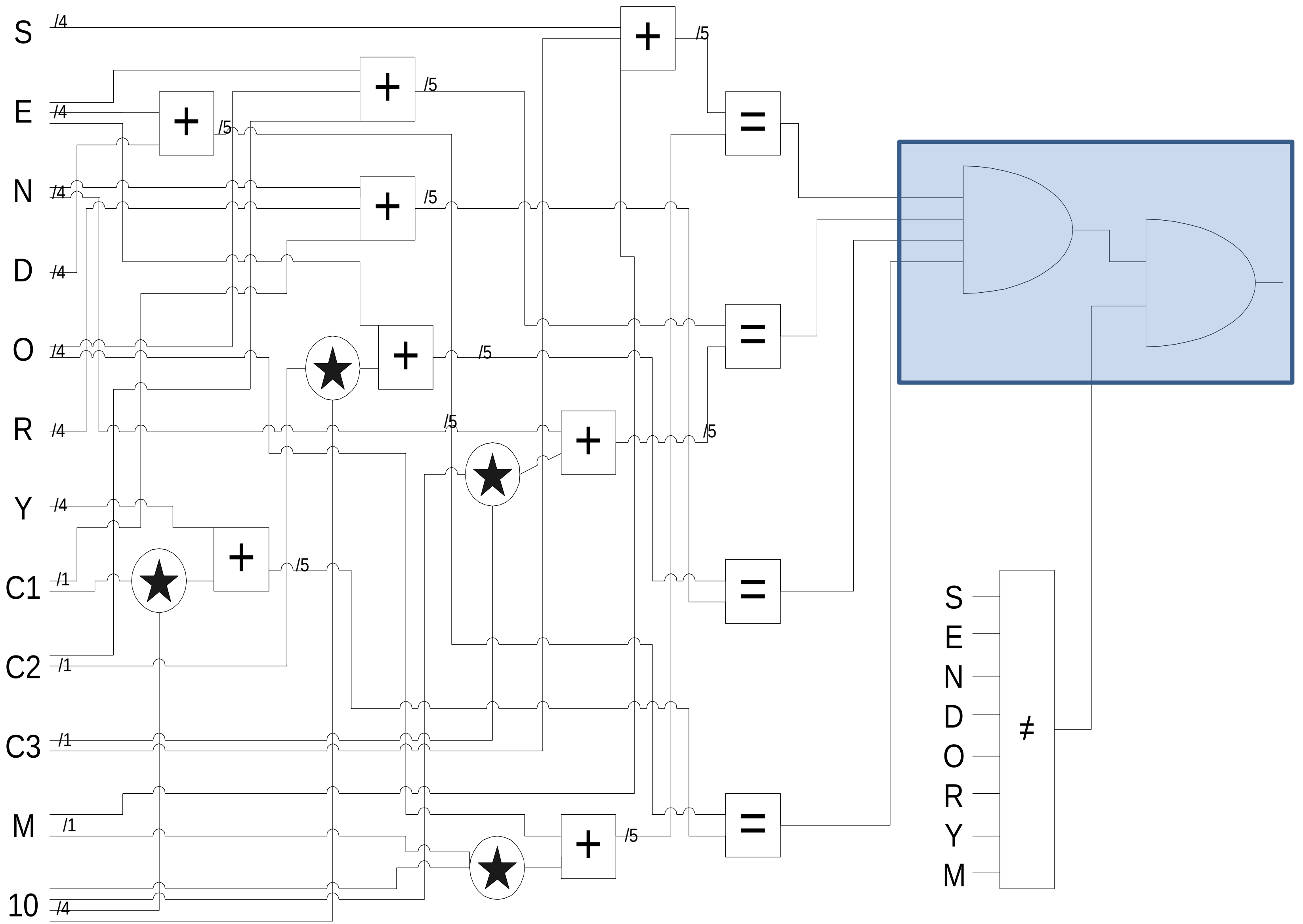
Equality Gates

- 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.





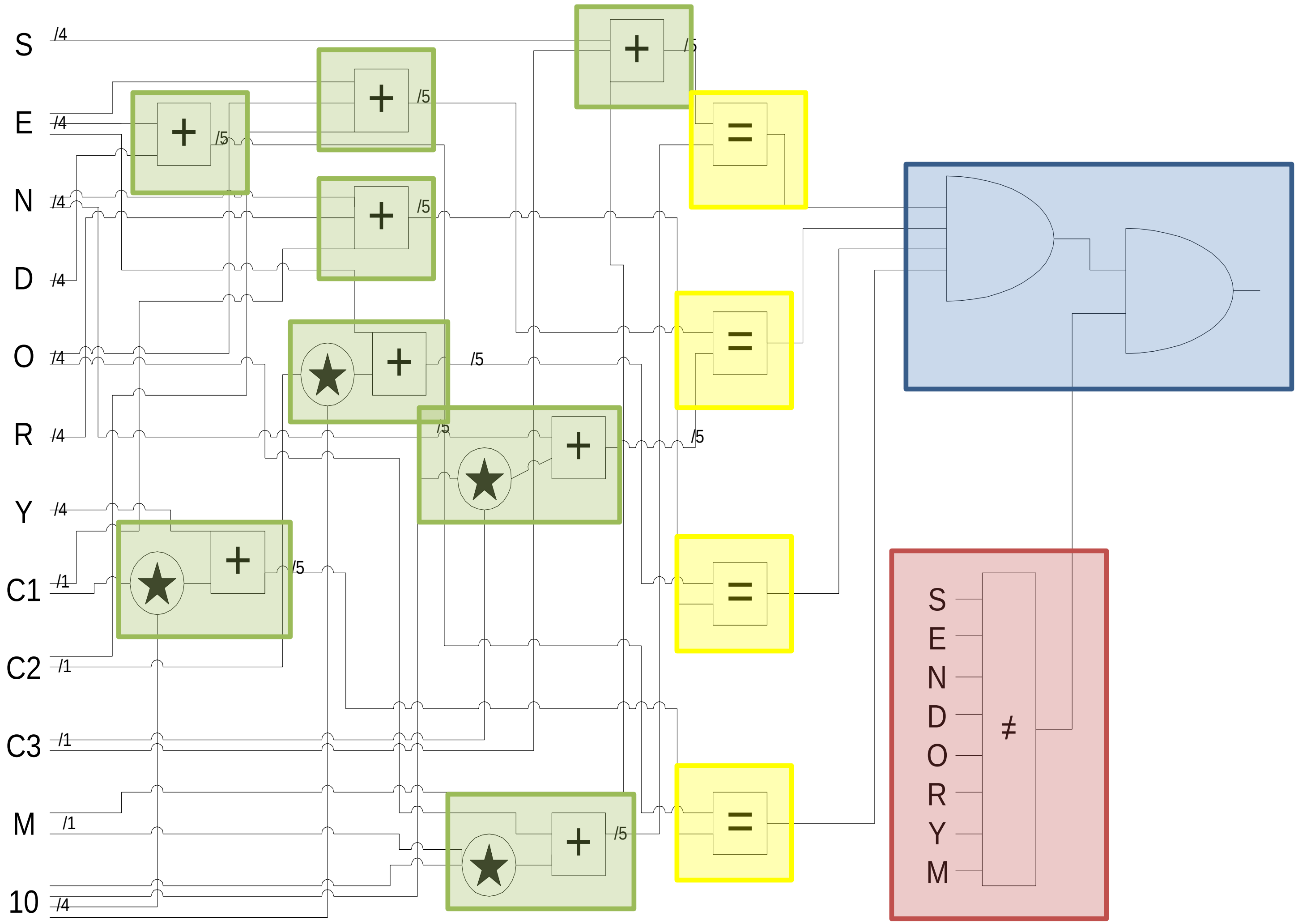
S
/4
E
/4
N
/4
D
/4
O
/4
R
/4
Y
/4
C1
/1
C2
/1
C3
/1
M
/1
10
/4
+
/5
=
≠
S
E
N
D
O
R
Y
M

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**.



Total cost: 2(1288 + 112+ 294 + 288 + 19)+1184 = 5186 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

$$D + E = 10*C1 + Y$$
$$C1 + N + R = 10*C2 + E$$
$$C2 + E + O = 10*C3 + N$$
$$C3 + S + M = 10*M + O$$

- 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

Satisfiability

$(a'+b+c)(a+b'+c')$

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.

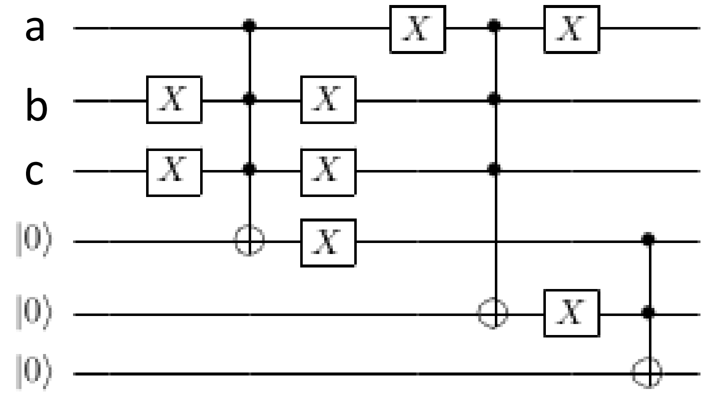
Perkowski's method:

- This formula is emulated by gates. In other words, I simply do: {(a 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.

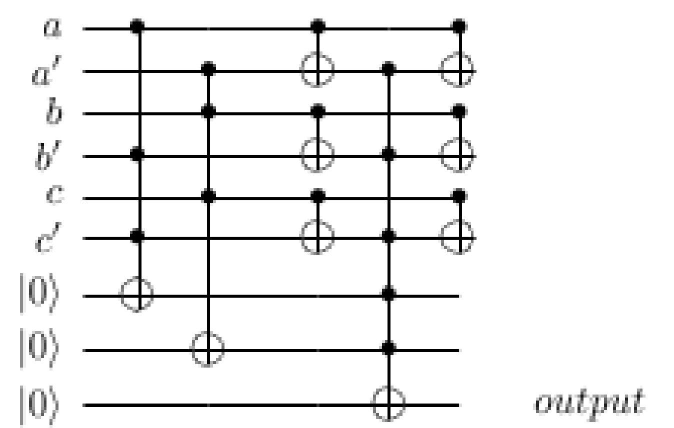
Hogg's method

- 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.



Perkowski's method

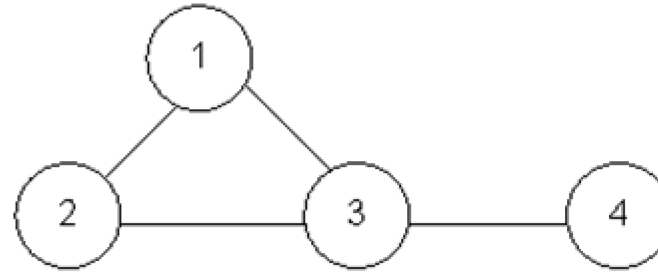


Hogg's Method – costs more qubits

MAXIMUM CLIQUE

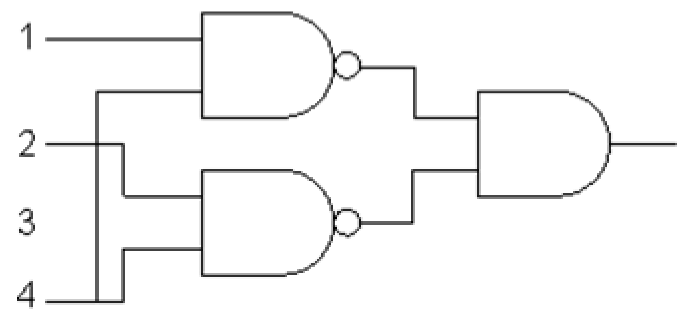
Maximum Clique

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



Perkowski's Method:

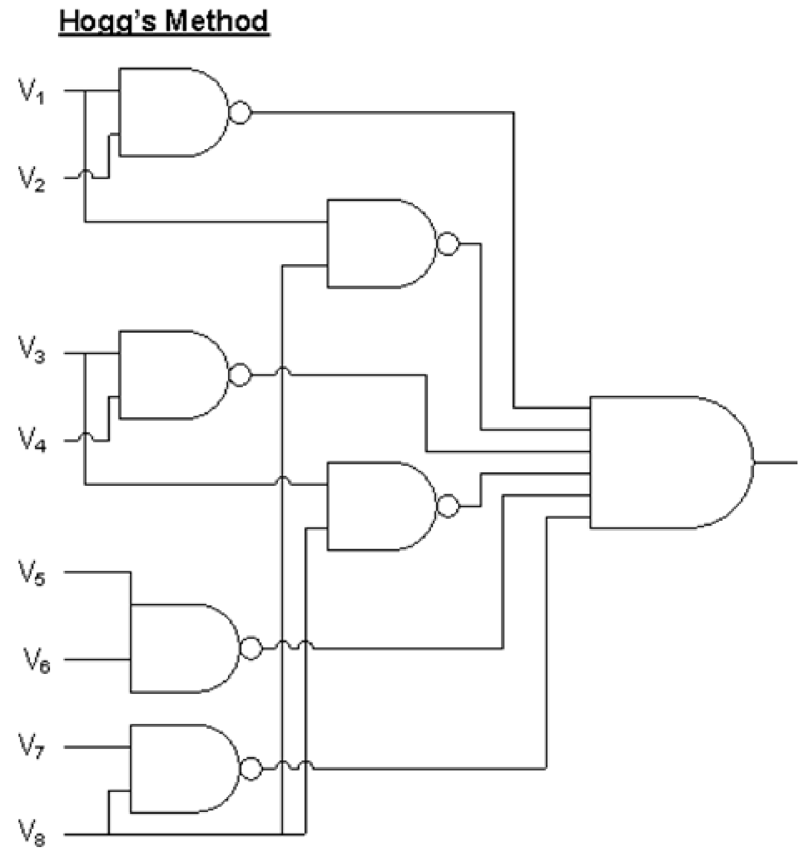
1. I created the oracle for the 4 node graph as shown on the right.
2. I created this oracle by representing each node with a 1 if it was "activated", i.e. part of the proposed clique, and with a 0 if it wasn't.
3. I then NAND-ed all nodes that were not connected, because the maximum clique must have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



Perkowski's method

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

Problems and Methods		Cost in Basic Gates	Cost in Qubits
Graph Coloring	*Hogg's*	Best: $42\left(en + \frac{n^2(n-1)}{2}\right) - 96$ Worst: $10en + 10\frac{n^2(n-1)}{2} + 2^{en+\frac{n^2(n-1)}{2}+1} - 3$	$2 + en + \frac{n^2(n-1)}{2} + n^2$
	Perkowski's	Best: $2e(36\lVert\log_2 n\rVert - 63) - 96$ Worst: $2e(4\lVert\log_2 n\rVert + 2^{q+1} - 2) + 2^{e+1} - 3$	$2e + 2 + n\lVert\log_2 n\rVert$
Maximum Clique	*Hogg's*	Perkowski's method + 42n	Perkowski's + 2n
	Perkowski's	Best: $42\left(\frac{n(n-1)}{2} - e\right) - 96$ Worst: $10\left(\frac{n(n-1)}{2} - e\right) + 2^{\frac{n(n-1)}{2}-e+1} - 3$	$\frac{n(n-1)}{2} - e + 2 + n$
Satisfiability	*Hogg's*	Best: $4v+2m*(2^{n+1}-3)+32(m+v)-96$ Worst: 4v+2m(2n+1-3)+2m+v+1-3	v+m+2
	Perkowski's	Best: $2m*(2^{n+1}-3)+2t+32m-96$ Worst: 2m*(2n+1-3)+2t+2m+1-3	3v+m+2
SEND MORE MONEY	*Hogg's*	Best: 93,138 Worst: 10^{165}	949
	Perkowski's	Best: 3952 Worst: $2.2*10^{12}$	126

Results: Max Clique and Graph Coloring

Max Clique

	N	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Hogg's best case	**Pulses**	282	492	744	1038	1374	1752	2172	2634	3138	3684	4272	4902	5574	6288	7044
e=2n	**Qubits**	23	30	38	47	57	68	80	93	107	122	138	155	173	192	212
Hogg's Worst case	**Pulses**	1.E+03	3.E+04	2.E+06	3.E+08	7.E+10	4.E+13	4.E+16	7.E+19	3.E+23	2.E+27	4.E+31	1.E+36	9.E+40	1.E+46	3.E+51
Perkowski's Best Case	**N**	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
e=2n	**Pulses**	30	198	408	660	954	1290	1668	2088	2550	3054	3600	4188	4818	5490	6204
	Qubits	11	16	22	29	37	46	56	67	79	92	106	121	137	154	172
Perkowski's Worst Case	**Pulses**	4.E+01	3.E+02	8.E+03	5.E+05	7.E+07	2.E+10	9.E+12	9.E+15	2.E+19	8.E+22	6.E+26	1.E+31	3.E+35	2.E+40	3.E+45

Graph Coloring

	N	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Hogg's best case	**Pulses**	6708	10194	14688	20316	27204	35478	45264	56688	69876	84954	102048	121284	142788	166686	193104
e=2n	**Qubits**	200	296	418	569	752	970	1226	1523	1864	2252	2690	3181	3728	4334	5002
Hogg's Worst case (e=2n)	**Pulses**	1.E+49	1.E+74	2.E+106	4.E+146	9.E+195	2.E+255	Excel can not calculate any more								
Hogg with e = 3n	**Pulses**	8220	12252	17376	23718	31404	40560	51312	63786	78108	94404	112800	133422	156396	181848	209904
Complete Hogg	**Pulses**	7464	12252	18720	27120	37704	50724	66432	85080	106920	132204	161184	194112	231240	272820	319104
Perkowski's Best Case	**Pulses**	625	970	1344	1744	2168	2612	3075	3555	4052	4563	5088	5626	6176	6738	7311
e=2n	**Qubits**	42	50	58	67	75	84	93	102	111	121	130	139	149	159	168
Perkowski's Worst Case	**Pulses**	9.E+21	6.E+29	7.E+38	1.E+49	3.E+60	1.E+73	1.E+87	1.E+102	2.E+118	6.E+135	3.E+154	2.E+174	2.E+195	4.E+217	1.E+241
Perkowski with e = 3n	**Pulses**	986	1503	2064	2664	3299	3966	4660	5381	6125	6892	7680	8487	9313	10155	11015
Complete Perkowski	**Pulses**	355	703	1164	1744	2451	3289	4264	5381	6644	8057	9624	11348	13233	15281	17496

Results - SAT

t = mn/2

Perkowski's Method

n\m	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	124	168	212	256	300	344	388	432	476	520	564	608	652	696	740	784
3	209	270	331	392	453	514	575	636	697	758	819	880	941	1002	1063	1124
4	374	468	562	656	750	844	938	1032	1126	1220	1314	1408	1502	1596	1690	1784
5	699	858	1017	1176	1335	1494	1653	1812	1971	2130	2289	2448	2607	2766	2925	3084

Hogg's Method (v = n)

2	186	228	270	312	354	396	438	480	522	564	606	648	690	732	774	816
3	302	360	418	476	534	592	650	708	766	824	882	940	998	1056	1114	1172
4	498	588	678	768	858	948	1038	1128	1218	1308	1398	1488	1578	1668	1758	1848
5	854	1008	1162	1316	1470	1624	1778	1932	2086	2240	2394	2548	2702	2856	3010	3164

t = mn/3

Perkowski's Method

n\m																
2	120.6667	164	207.3333	250.6667	294	337.3333	380.6667	424	467.3333	510.6667	554	597.3333	640.6667	684	727.3333	770.6667
3	204	264	324	384	444	504	564	624	684	744	804	864	924	984	1044	1104
4	367.3333	460	552.6667	645.3333	738	830.6667	923.3333	1016	1108.667	1201.333	1294	1386.667	1479.333	1572	1664.667	1757.333
5	690.6667	848	1005.333	1162.667	1320	1477.333	1634.667	1792	1949.333	2106.667	2264	2421.333	2578.667	2736	2893.333	3050.667

Hogg's Method(v = n)

2	186	228	270	312	354	396	438	480	522	564	606	648	690	732	774	816
3	302	360	418	476	534	592	650	708	766	824	882	940	998	1056	1114	1172
4	498	588	678	768	858	948	1038	1128	1218	1308	1398	1488	1578	1668	1758	1848
5	854	1008	1162	1316	1470	1624	1778	1932	2086	2240	2394	2548	2702	2856	3010	3164

t = 2mn/3

Perkowski's Method

n\m																
2	127.3333	172	216.6667	261.3333	306	350.6667	395.3333	440	484.6667	529.3333	574	618.6667	663.3333	708	752.6667	797.3333
3	214	276	338	400	462	524	586	648	710	772	834	896	958	1020	1082	1144
4	380.6667	476	571.3333	666.6667	762	857.3333	952.6667	1048	1143.333	1238.667	1334	1429.333	1524.667	1620	1715.333	1810.667
5	707.3333	868	1028.667	1189.333	1350	1510.667	1671.333	1832	1992.667	2153.333	2314	2474.667	2635.333	2796	2956.667	3117.333

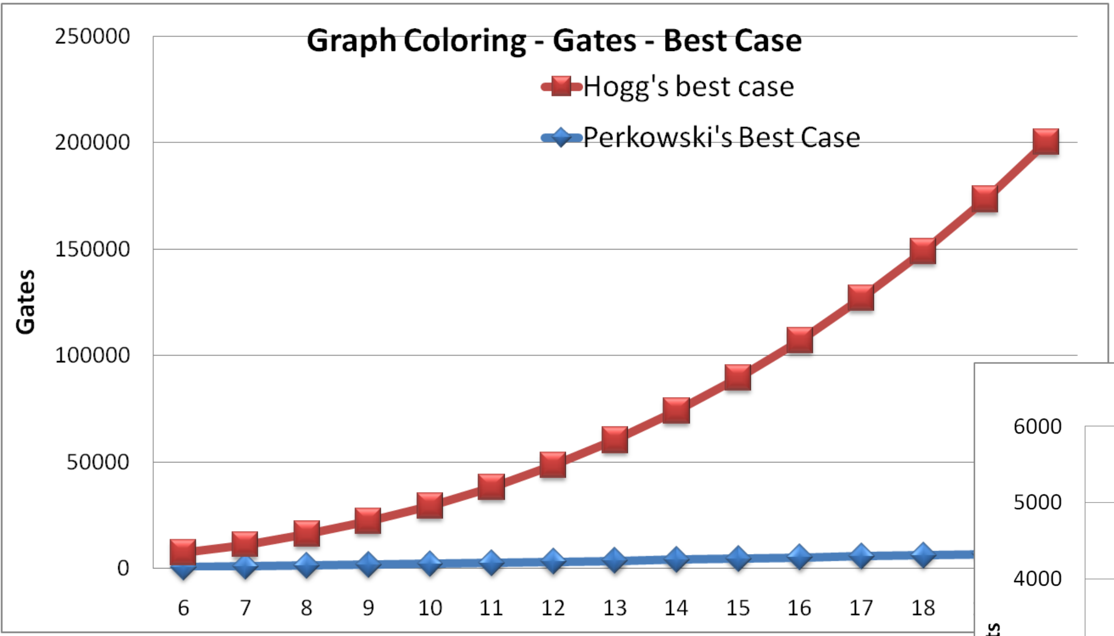
Hogg's Method

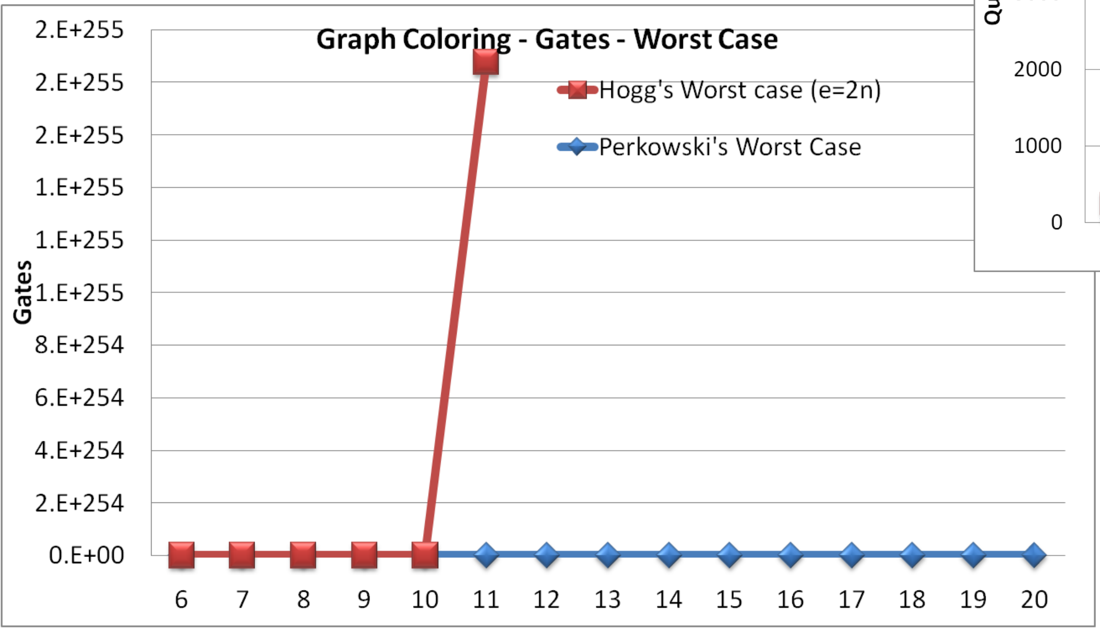
n\m																
2	186	228	270	312	354	396	438	480	522	564	606	648	690	732	774	816
3	302	360	418	476	534	592	650	708	766	824	882	940	998	1056	1114	1172
4	498	588	678	768	858	948	1038	1128	1218	1308	1398	1488	1578	1668	1758	1848
5	854	1008	1162	1316	1470	1624	1778	1932	2086	2240	2394	2548	2702	2856	3010	3164

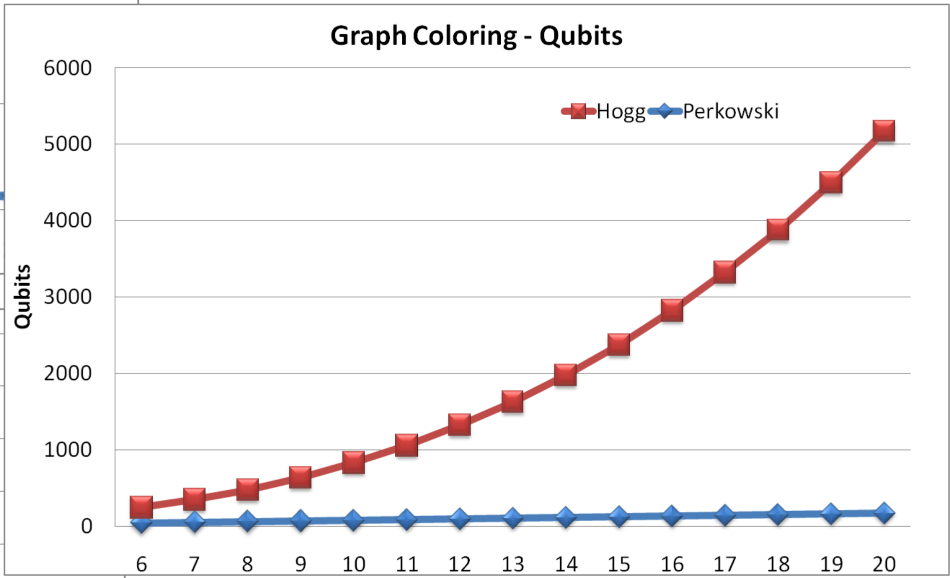
Results – SAT Contd..

Worst Case: 3-SAT																
Hogg	651	1189	2239	4313	8435	16653	33063	65857	131419	262517	524687	1049001	2097603	4194781	8389111	16777745
Pk: (t=mn/3)	201	293	449	733	1273	2325	4401	8525	16745	33157	65953	131517	262617	524789	1049105	2097709
Pk: (t=mn/2)	206	299	456	741	1282	2335	4412	8537	16758	33171	65968	131533	262634	524807	1049124	2097729
Pk: (t=2mn/3)	211	305	463	749	1291	2345	4423	8549	16771	33185	65983	131549	262651	524825	1049143	2097749
Qubits: 3-SAT																
Hogg	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Perkowski	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

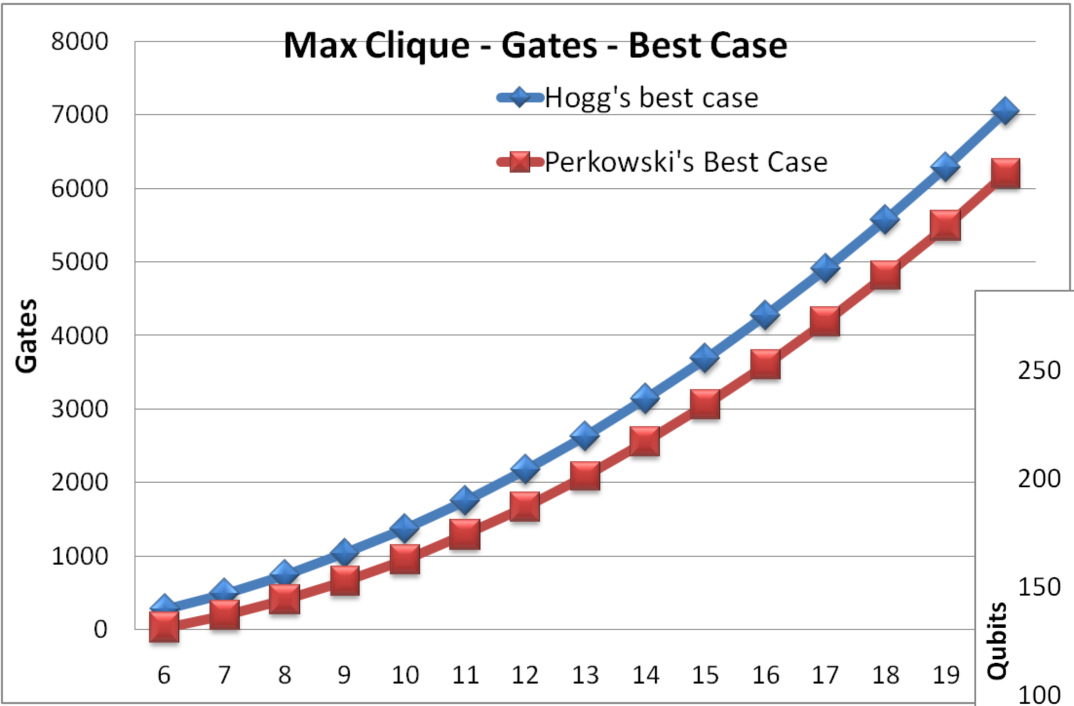
Results: Graph Coloring

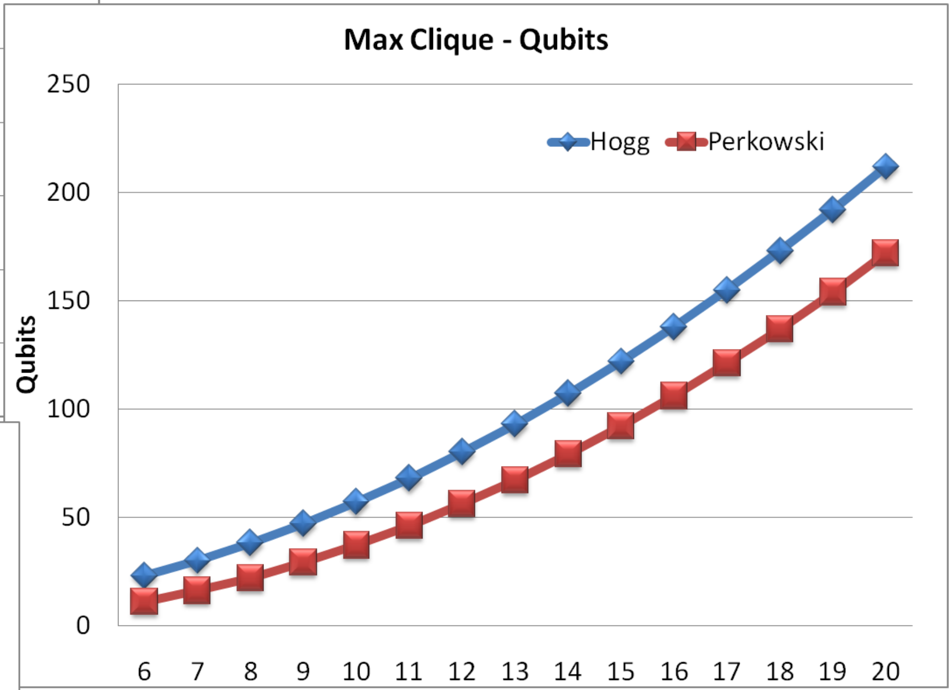


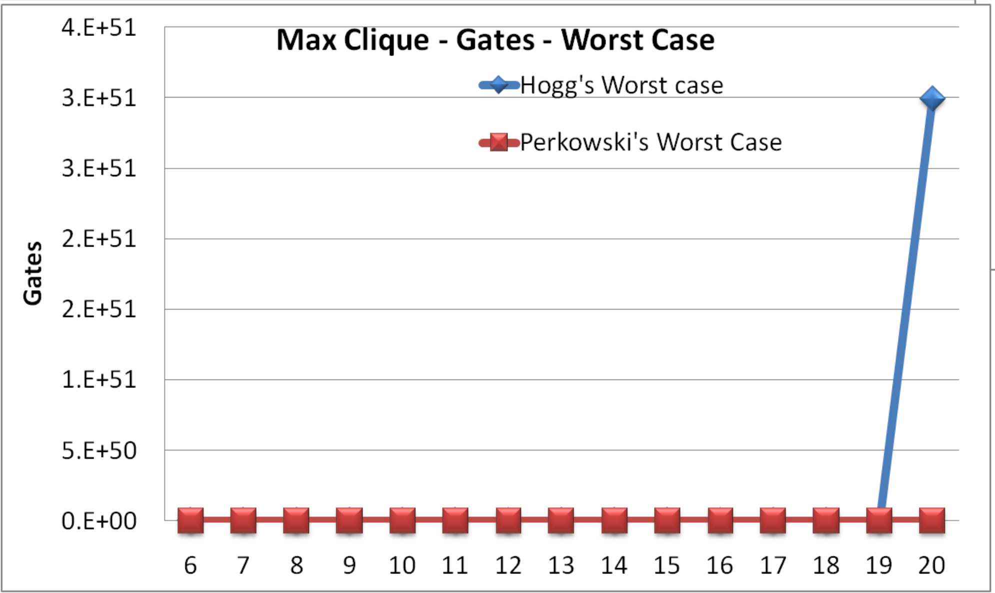




Results: Max Clique

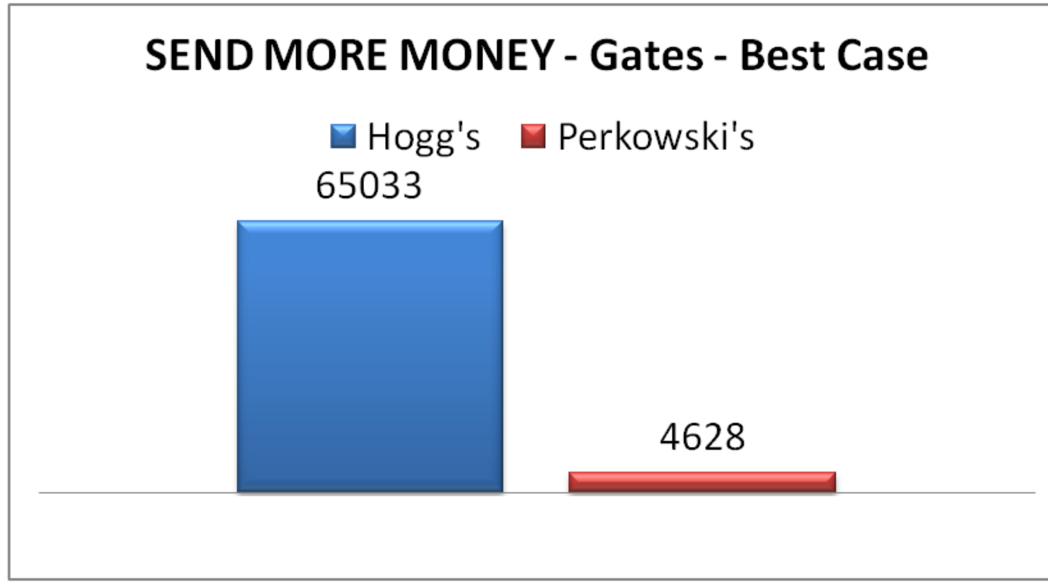


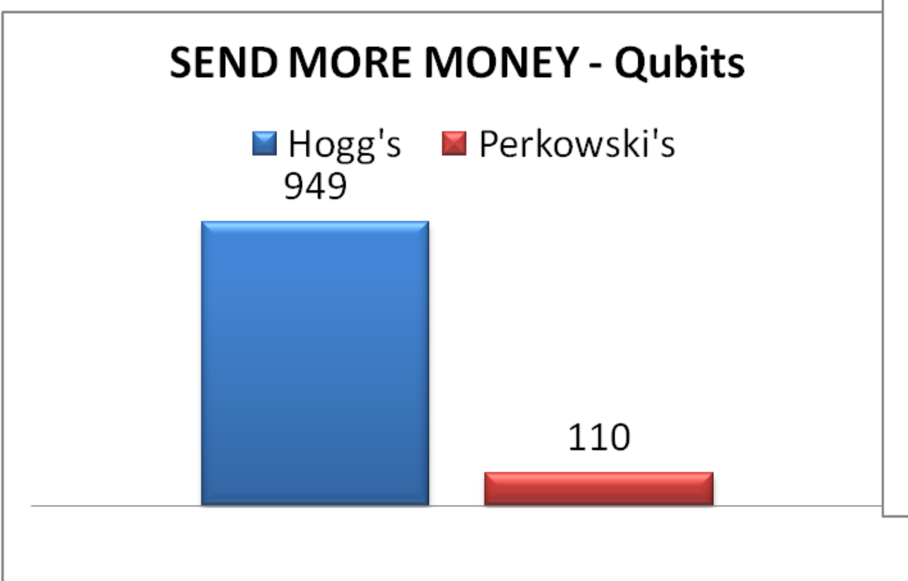


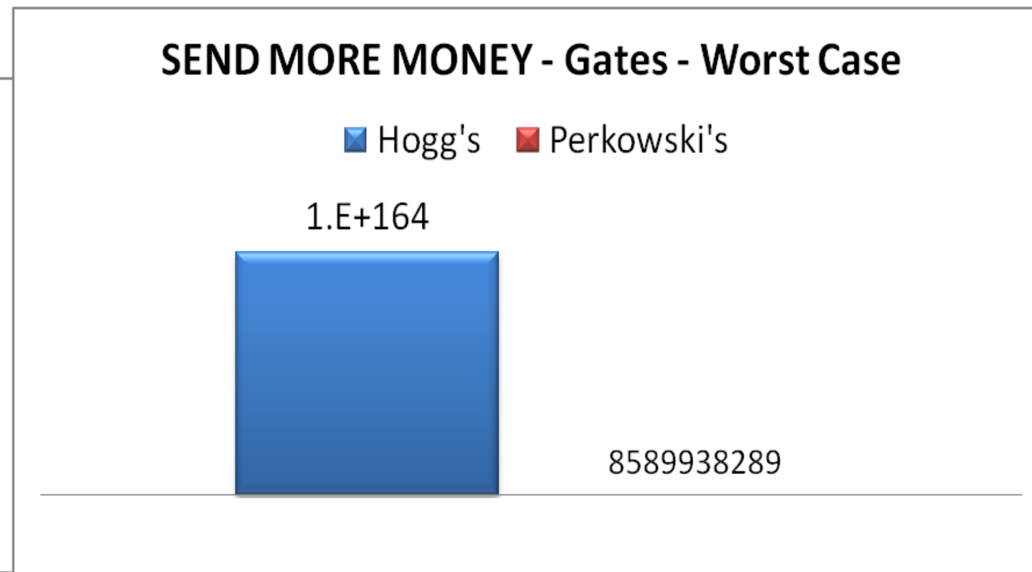


RESULTS: SEND MORE MONEY

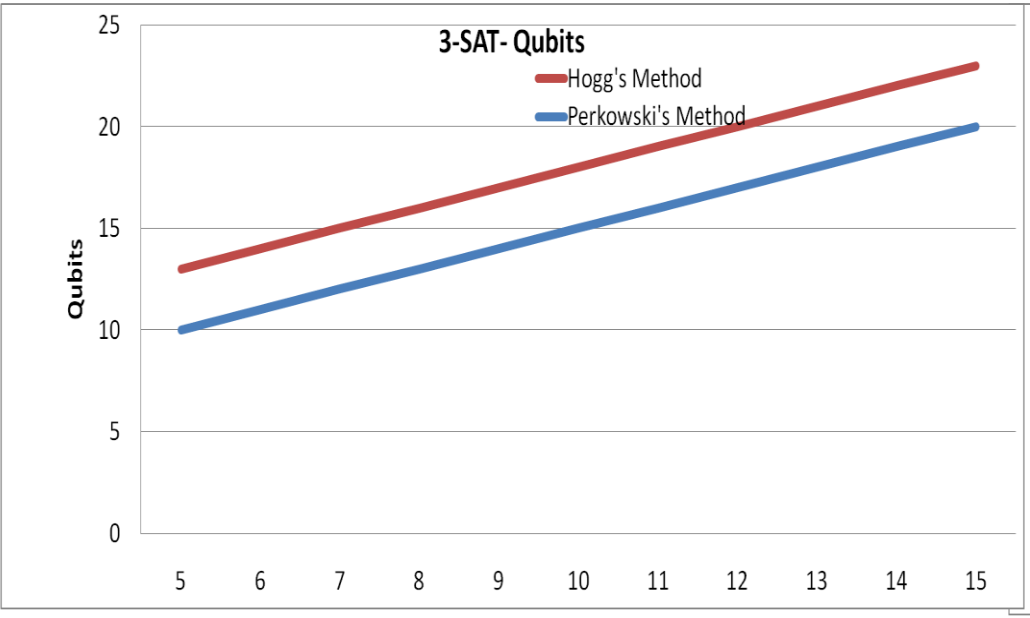
SEND MORE MONEY					
Hogg's	Best Case			Worst Case	
	Gates	65033		1.E+164	
	Qbits	949			
Perkowski's	Best Case			Worst Case	
	Gates	4628		2.2E12	
	Qbits	110			

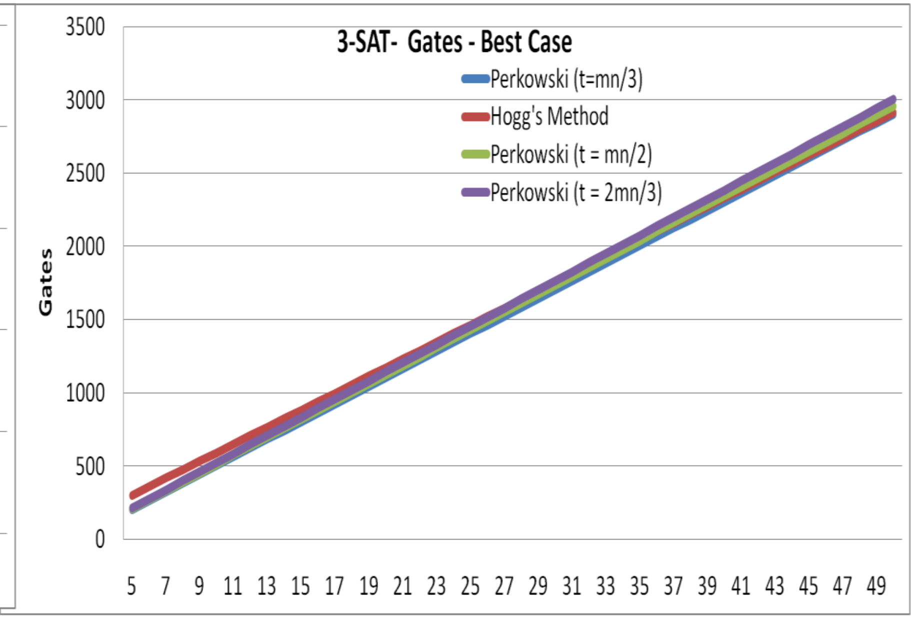


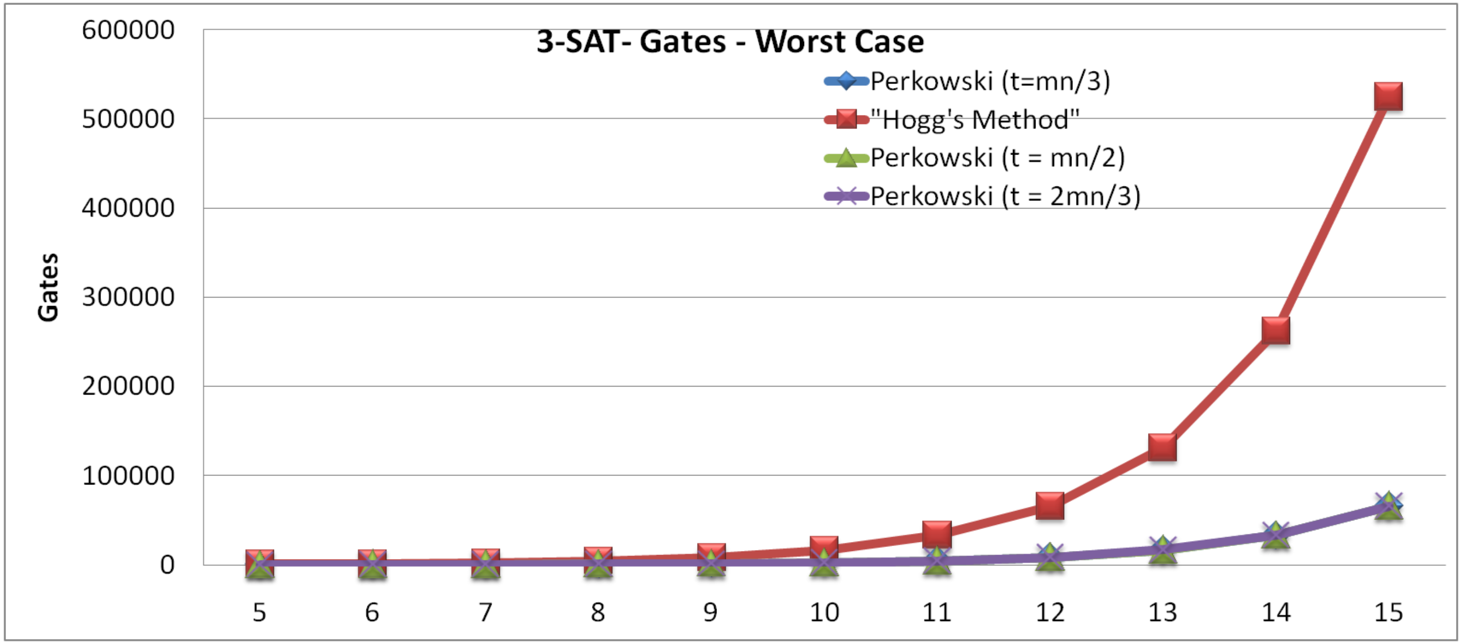




Results: SAT


3-SAT- Qubits
Hogg's Method
Perkowski's Method
Qubits
25
20
15
10
5
0
5
6
7
8
9
10
11
12
13
14
15


3-SAT- Gates - Best Case
Perkowski (t=mn/3)
Hogg's Method
Perkowski (t = mn/2)
Perkowski (t = 2mn/3)
Gates
3500
3000
2500
2000
1500
1000
500
0
5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49


3-SAT- Gates - Worst Case
Perkowski (t=mn/3)
"Hogg's Method"
Perkowski (t = mn/2)
Perkowski (t = 2mn/3)
Gates
600000
500000
400000
300000
200000
100000
0
5
6
7
8
9
10
11
12
13
14
15

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

1. Cerf, N. J., Grover, L. K., & Williams, C. P. (1999, December 1). Nested Quantum Search and NP Hard Problems. *Applicable Algebra in Engineering, Communication, and Computing*, *10*, 311-388
2. Perry, R. T. (April 29,2006). *Temple of Quantum Computing*
3. Hogg, T. (1996, March). Quantum Computing and Phase Transitions in Combinatorial Search. *Journal of Artificial Intelligence*, *4*, 91-128
4. Maslov, D., & Dueck, W. R. (2004, March 5). Improved Quantum Cost for n-Bit Toffoli Gates
5. Lee, S., Lee, J., & Kim, T. (2003, July 7). Cost of Basic Gates in Quantum Computation. *Department of Physics, Korea Advanced Institute of Science and Technology,*
6. Perkowski, M. A. (2009) *Quantum Robotics*
7. Nielsen, M. A., & Chuang, I. L. (2000). *Quantum Computing and Quantum Information*. Cambridge, UK: Cambridge University Press
8. Grover, L. K. (1997). A Fast Quantum Mechanical Algorithm for Database Search. *28th Annual Symposium on the Theory of Computing*, 212 – 219

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 $2^{\frac{n(n-1)}{2}-e+1}-3$ 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.

SEND MORE MONEY - Perkowski

- 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 = 8.59x10^9 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.

SEND MORE MONEY - Hogg

- 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 there 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.