2011 41st IEEE International Symposium on Multiple-Valued Logic

Comparison of influence of two data-encoding methods for Grover Algorithm on quantum costs

Sidharth Dhawan
Westview High School, Portland, Oregon USA,
siddhawan@yahoo.com

Marek Perkowski
Dept. of Electrical and Computer Engineering,
Portland State University, Portland, Oregon,
mperkows@ee.pdx.edu,

Abstract

It is important to be able to calculate realistic estimates of quantum costs for real oracles used in quantum algorithms. In this paper, we compare Perkowski's[1] oracle data encoding method with Hogg's[2] encoding method for Grover algorithm[3], to examine the decrease in Oracle gate cost, if any, for four common constraint satisfaction problems: Graph coloring, Satisfiability, Send-More-Money and Max Clique.

1. Introduction

Grover's quantum algorithm [3] gives a polynomial speedup over classical unstructured search. In the rapid search for new methods, one of the least recognized ways to save time in constraint satisfaction problems is to create a more efficient Grover oracle. So far, not much was published on practical design of Grover oracles for various constraint satisfaction problems (or any other problems), with the exception of [1]. An oracle that has smaller cost and is faster can be obtained by good quantum logic synthesis and also by appropriate encoding of input data to it. In this paper we compare Hogg's [2] and Perkowski's [1] encoding methods for four different constraint satisfaction problems. We found that Perkowski's encoding is usually more efficient than Hogg's method.

2. Quantum Cost

The effect of a single-qubit gate is to change the quantum state of a qubit ($\alpha|0\rangle + \beta|1\rangle$), or equivalently, change its position on the Bloch sphere. A quantum gate can also be represented by a unitary matrix, which can be multiplied by the vector $\langle\alpha,\beta\rangle$ to determine the new quantum state. Unlike the matrices of classical reversible gates, which are necessarily permutative and operate on three inputs, the matrices of truly quantum gates operate on two inputs, and are not permutative. Some examples of truly quantum gates are the phase gate, which performs a rotation about the z-axis, a Hadamard Gate, a NOT gate (which switches between zero and one), and the controlled NOT gate (equivalent to the classical EXOR gate).

Various costs have been defined for quantum gates. Costs in terms of electromagnetic pulses for NMR technology were given in [10]. According to this paper, the Hadamard gate costs 2, the inverter gate costs 1, the Feynman costs 5, etc. Here we define cost as approximately the count of basic (one or two qubit) gates in a circuit [7]. For example, a Toffoli, or double controlled NOT gate (which is not considered a basic quantum gate) can be reduced to the Barenco circuit [6], which consists of five basic quantum gates. Thus, the cost of the Toffoli gate in this cost counting method would be five.

One of the most useful (and costly) reversible gates is the n-qubit Toffoli gate. As shown by Maslov and Dueck [7,8], for large number of qubits ($n > 6$), an n-input Toffoli gate is $32n-96$ basic gates, plus one extra qubit. We use this as the best case cost of a Toffoli gate. The alternative, worst case, cost formula, is defined by Maslov & Dueck [7,8] as $(2^{n+1})-3$ basic gates. Notice that the worst case formula grows exponentially, while the best case formula is linear. There is a very large disparity between these two formulas – for example, a 9-qubit Toffoli gate has a worst case cost of $2^{10}-3 = 1021$ basic gates and a best case cost of $32(9)-96 = 192$ basic gates. Since there is a large disparity between the best and worst case estimates, we have calculated both these costs for all four problems.

The cost of a circuit is simply calculated by the count of each gate in the circuit multiplied by the cost of that gate. For example, an arithmetic inequality gate for 2-qubit encoding of data consists of 4 Feynman gates and one Toffoli gate. The cost is thus $4(1)+1(5) = 9$ basic gates.

3. Quantum Search

In Grover's algorithm[3], one must create an oracle which, given an input state, determines if the state is a possible solution. Grover's algorithm works by superposing all possible starting states in its qubits and then applying the classical oracle operation to "tag" the solution states using phase change. Since a change in phase cannot be measured a collective Hadamard gate breaks the superposition and translates this phase change into an amplitude change, which is amplified in the "I(s)" operation using a Toffoli gate and inverters. Grover algorithm repeats this process $\sqrt{N}$ times to maximally amplify the solution states and then performs quantum measurement [11, 14].

To increase the efficiency, it is very important to find a cost effective oracle. There are two known ways to encode data for the oracle, which significantly affect the method to synthesize the oracle's circuit and thus the quantum cost. The first is the subset selection (Hogg's) method [2], which creates one variable for every mapping and selects a certain subset. The second is the mapping (Perkowski's) method [1], where more than one qubit is used to dictate a value for each variable. To illustrate, for the SEND MORE MONEY problem, the cost difference between the two encoding methods is about 30

0195-623X/11 $26.00 © 2011 IEEE
DOI 10.1109/ISMVL.2011.29

million billion basic gates. This is explained more in the following sections.

4. Graph Coloring Problem

Hogg's encoding:

The oracle in Fig 4 shows the subset selection (Hogg's method) to a simple 3-node graph coloring problem. This problem involves finding a good coloring for a graph, such that no two nodes connected by an edge have the same color. Since only one color per node can be selected, a set of "NAND" gates dictate that no two elements of the same row

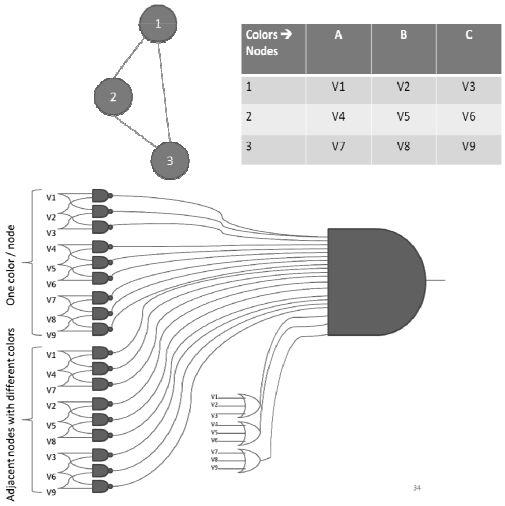

Fig 4: Oracle for Hogg's method

(in the chart) can be selected. Since no two adjacent nodes can have the same color, another set of "NAND" gates dictate that no two elements in the same column can be selected.

Next we extend the solution for a graph of ***n*** nodes connected by ***e*** edges, where each node can have one of ***c*** colors. The following steps are required for this:

Step 1: We create one qubit for every assignment of one color to one node, resulting in c*n qubits. In the worst case, if all the nodes are interconnected, each node will need its own separate color; thus n=c. In order to accommodate for the worst case, we need c*n = n*n = **n^2 qubits**.

Step 2: We make sure that only one color is assigned to each node. If a color is assigned to a node then the qubit corresponding to color c and node n is one. Since no two color-node assignments for one node (n colors) can be one, we need a network of NAND gates such that every color assignment for one node is NANDed with every other color assignment for that node. We need n*(n-1)/2 NANDs for each of n nodes. Thus, we have *n^2(n-1)/2 NAND gates*. The quantum equivalent of a NAND gate is a Toffoli gate, which costs 5 basic gates and one ancilla qubit. Thus, the overall cost of this step is **$5n^2$(n-1)/2 basic gates** and **n^2(n-1)/2 ancilla bits**.

Step 3: We make sure that no two adjacent nodes are given the same color assignment. For this we need c=n NAND gates. This condition depends on the number of edges (e) of the graph, so we need *n*e* Toffoli gates. Thus, the overall cost of this step is **5(e)(n) basic gates** and **(e)(n) ancilla qubits**.

Step 4: In order to make sure that all of these conditions are satisfied, we need a global AND gate at the end and include the outputs of all ancilla bits as controls. As mentioned before we will use two estimates for the cost of a Toffoli gate: Best Case = 32m-96 basic gates + 1 garbage bit, and Worst Case = $2^{m+1} - 3$ basic gates, where m is the number of controls, and m>5. For the AND gate, m= en+n^2(n-1)/2, which means that the best case cost of this gate is **32(en+n^2(n-1)/2)-96 basic gates** and **2 qubits.** In the worst case the cost is: $\mathbf{2^{en+\frac{n^2(n-1)}{2}+1} - 3}$ basic gates.

$$\boldsymbol{Overall\ Best\ Case\ Cost} = \mathbf{32\left(en + \frac{n^2(n-1)}{2}\right) - 96 + 5en + 5\frac{n^2(n-1)}{2} = 37\left(en + \frac{n^2(n-1)}{2}\right) - 96}$$

$$\boldsymbol{Overall\ qubits} = \boldsymbol{ancilla\ bits} + \boldsymbol{qubits} = \mathbf{2} + \boldsymbol{en} + \frac{\boldsymbol{n^2(n-1)}}{\mathbf{2}} + \boldsymbol{n^2}$$

$$\boldsymbol{Worst\ case\ Cost} = \mathbf{5}\boldsymbol{en} + \mathbf{5}\frac{\boldsymbol{n^2(n-1)}}{\mathbf{2}} + \mathbf{2}^{\boldsymbol{en}+\frac{\boldsymbol{n^2(n-1)}}{\mathbf{2}}+\mathbf{1}} - \mathbf{3}$$

Perkowski's Encoding:

This oracle in Fig 5 shows another way to solve the same problem using Perkowski's method. Each node is represented by two qubits that have binary values which represent the color of the node in the graph. Since only one color can be selected at a time in this oracle, we need inequality ("≠") gates[1,14] to make sure that no two adjacent nodes have the same color.

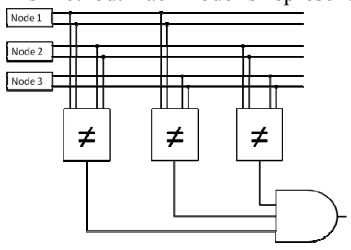

Fig 5: Oracle for Perkowski's method

For this method, the cost of the circuit is calculated by the following steps:

Step 1: We represent each qubit by its binary color value. For example, if node "a" was color 2, its qubits would be 1-0. Colors 0,1,2 and 3 will be represented by 2 qubits; and colors 4,5,6 and 7 by 3 qubits. So for an arbitrary power of two (p), $2^{(\text{number of qubits-1})}$ = p. Thus, each color count requires q+1 qubits, such that 2^q is the largest power of two smaller than or equal to the color count. Another way to write this is: q = $||\log_2 n||+1 = ||\log_2 n+1||$, where ||x|| is x rounded to the largest whole number smaller than or equal to x. For this circuit, we need q*c = $||\log_2 n+1||$*n qubits.

Step 2: We ensure that no two qubits connected by an edge have the same value (color). This will require inequality gates [1], which consist of 2q Feynman gates, 2q+1 NOT gates, one

q-bit Toffoli gate, and an ancilla bit. This process must be repeated e times. Thus, the best case cost of this whole step is:
= e(2q(Feynman)+(2q+1)(NOT)+(q-bit Toffoli))
= e(2q+2q+1+32q-96)
= **e(36q-95) basic gates** and **2e qubits** (e output ancilla bits)

The worst case cost is = $e(4q + 2^{q+1} - 2)$

Step 3: To make sure all conditions are satisfied, a global AND is required with e inputs. This will cost 32e-96 basic gates and 2 bits for best case, or 2^{e+1}-3 basic gates and 1 bit for worst case.

$$\boldsymbol{Total\ Best\ Case\ Cost} = \boldsymbol{e(36q-95) + 32e - 96} = \boldsymbol{e(36\|\log_2 n\| - 63) - 96}$$

$$\boldsymbol{Total\ Qubits} = \boldsymbol{2e + 2 + n\|\log_2 n\|}$$

$$\boldsymbol{Worst\ case\ Cost} = \boldsymbol{e\left(4\|\log_2 n\| + 2^{q+1} - 2\right) + 2^{e+1} - 3}$$

5. Maximum Clique Problem

The maximum clique problem has the same conditions as the graph coloring problem, except that the goal of the maximum clique problem is to find the largest number of nodes on the graph that are all directly interconnected (each node in a clique of n nodes has exactly n-1 edges to other nodes from this clique).

For this problem, Hogg's and Perkowski's encodings will be very similar. In this problem, Grover's algorithm proposes a random solution of "activated" bits, and the oracle determines whether the nodes in this guess are all interconnected. Thus, we need to determine if a certain bit is "activated", i.e., included in the random guess of nodes. If it is activated, its representative qubit will be of value 1. Thus only n qubits (one per node) are needed.

Perkowski's Encoding:

The following steps are required to calculate the cost of the oracle using Perkowski's encoding:

Step 1: Make sure that no two nodes that are not connected are both activated. This will require NAND gates. The total number of Toffoli gates needed is the number of possible connections minus the number of connections. According to the "handshake" [9] problem, the total possible connections is n(n-1)/2. We define the total number of connections with e. Thus we need a total of n(n-1)/2-e Toffoli gates. The cost is thus 5(n(n-1)/2-e) basic gates, and n(n-1)/2-e ancilla qubits.

Step 2: Perform a global AND of the results. Since there are n(n-1)/2–e outputs, 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 basic gates and two qubits for best case. For the worst case, it will cost $2^{\frac{n(n-1)}{2}-e+1} - 3$ basic gates and 1 qubit.

Thus, the total cost of this circuit is:

$$\boldsymbol{Total\ Best\ Case\ Cost} = \boldsymbol{5\left(\frac{n(n-1)}{2} - e\right) + 32\left(\frac{n(n-1)}{2} - e\right) - 96} = \boldsymbol{37\left(\frac{n(n-1)}{2} - e\right) - 96}$$

$$\boldsymbol{Total\ Qubits} = \boldsymbol{\frac{n(n-1)}{2} - e + 2 + n}$$

$$\boldsymbol{Total\ Worst\ Case\ Cost} = \boldsymbol{5\left(\frac{n(n-1)}{2} - e\right) + 2^{\frac{n(n-1)}{2}-e+1} - 3}$$

Hogg's encoding:

Hogg's method for this case 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, it is activated and vice versa. The "activated node" qubit for a certain node can be thought of as analogous to Perkowski's qubits. 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: 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 5n+32n= **37n** more basic gates, and **2n** extra qubits (n for each additional NAND gate and n for each additional starting qubit).

6. Satisfiability (SAT) Problem

Satisfiability, or SAT, is a problem with v variables ORed and inverted together in various terms, all of which include a constant number of variables. These terms are all ANDed together. For example: $(\bar{a} + b + c)(a + \bar{b} + \bar{c})$. A completely simplified SAT formula would be $\bar{a}bd$, for example. This would be called 3- SAT, because there are three variables in every one of the ANDed terms.

Perkowski's encoding:

For Perkowski's method the oracle is very simple: for each term you would have an OR gate for each of the n variables, and then a global AND for all the terms. A large scale OR gate can be created in quantum technology by using a Toffoli gate

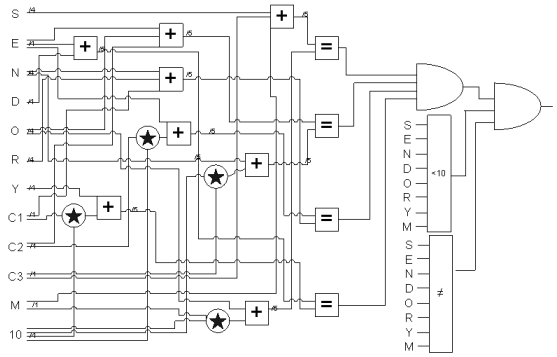

Fig 5: SEND MORE MONEY Oracle for Perkowski's method

with all the inputs inverted before and after the gate (to restore

original values). Note that for the term "a NOT", we would not have to include an extra NOT before the Toffoli, because we already are NOT-ing that input. Thus, the cost of this circuit would be m n-bit Toffoli gates, plus 2t NOT gates (t is the total of un-inverted terms), and then a final m-bit Toffoli at the end. Since n is usually small, we will use 2^{m+1} estimate.

Thus the best case cost is = **$m*(2^{n+1}-3)+2t+32m-96$ basic gates** and **$v+m+2$ qubits**.

The worst case cost is = **$m*(2^{n+1}-3)+2t+2^{m+1}-3$ basic gates**

Hogg's encoding:

For Hogg's method, the circuit is similar to Perkowski's circuit, except that no NOT gates are needed. Since we create a 1 and 0 (regular and inverted) qubit for every variable, the un-inverted variable can be represented by 1 qubit, and the inverted variable can be represented by the 0 qubit. However, there will be 2v qubits instead of v qubits in Perkowski's method.

Thus the best case cost is = **$2v+m*(2^{n+1}-3)+32(m+v)-96$ basic gates** and **$3v+m+2$ qubits**.

The worst case cost is = **$2v+m(2^{n+1}-3)+2^{m+v+1}-3$** basic gates**.**

Thus, the most efficient method in this case largely depends on values of t and v.

7. SEND MORE MONEY Problem

The formulation of this problem is shown below. Each of the eight letters represents a digit zero through nine. No letter can equal zero, and no two letters can equal the same number. This problem is part of a family of problems called cryptographic problems.

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

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

Some of these are simply used as entertainment puzzles but others have real life applications in robotics and AI.

Perkowski's encoding:

To test the constraints of this problem in Perkowski's method, we created new variables, c1 through c3 (carry bits), such that they fit the constraints previously mentioned The cost of the oracle is calculated by following steps:

Step 1: We make sure that no two letters have the same value. This will require more inequality gates [14]. Since there are 9 possible values for the letters, each letter will get four bits (one through nine in binary), thus there are 32 qubits for the letters. Since there are 8 letters, there will have to be 8(8-1)/2 = 28 inequality gates. Since there are four qubits for each letter, the inequality gates will cost four Feynman gates, four NOT gates, a four-bit Toffoli (13 basic gates), and an ancilla bit. Thus the cost is 28(4*1+4*1+1*13) = **588 basic gates** and **28 ancilla bits.** This is represented in Fig 6 as the block on the bottom right.

Step 2: We have to ensure that none of the letters has value greater than ten. Because we have allocated four qubits to each letter, it is possible that the letters actually could have values up to 15, which does not agree with the constraints of the problem. Thus, we need to add a "<10" block for each letter and feed the results into the final AND gate. For all the binary numbers between ten and fifteen, the most significant bit of the four allocated bits is always 1, and either the second or third- most- significant bit is 1. Thus a simple "<10" block could be created by the expression (a1*(a2+a3))', where a1-a4 are the four bits representing each letter. This circuit would cost an OR gate (decomposed to a Feynman gate, two NOT gates, and a Toffoli gate), a Toffoli gate and a final inverter, for a total of 14 basic gates and 2 ancilla bits. Since we would need eight of these (one for each letter), the total cost is **112 Basic Gates and 16 Ancilla Bits**, eight of which are outputs.

Step 3: We test the above four equations with quantum gates. This will require quantum adder gates[13,14], and it will require us to multiply carry gates by 10. Since in binary, 10 is 1010, c*1010 = c0c0, given that c is a one digit binary number (which all the carries are). We create four ancilla bits, and transfer our preferred "carry" value to two of them using a two Feynman gates. Thus multiplying a carry by ten will cost 4 qubits and 2 basic gates each. We will also have to use a network of quantum adder gates[13,14] to perform additions. Each network of adder gates, including the multipliers costs 138 basic gates and 12 qubits. However, because this is a unique problem, we can "tailor" our adder networks to our specific problem, which will cut down on the cost. Thus, the overall cost of this step will be **684 basic gates** and **48 qubits**. Four of these will be output qubits.

Step 4: We need a global AND gate at the end, to verify a total of 28+4+8 = 40 outputs. Thus, the global AND will cost 32(40)-96 = **1184 basic gates** and **2 bits** in the best case. The worst case cost is **$2^{40+1}-3 = 2.20x10^{12}$ basic gates**

Thus, the overall best case cost of the circuit is = 588+684+112+1184 = **2568 basic gates** and 32+16+28+48+2 = **126 qubits**.

The worst case cost is **$2.199*10^{12}$** basic gates.

The vast difference between these two figures is due to the fact that the worst case cost is exponentially growing with the number of controls.

Hogg's encoding:

Hogg's method for this problem is 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 bit, we cannot use quantum adder gates. Instead, we have to come up with all possible solutions to the equations, 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.

Thus the cost is calculated by following steps:

Step 1: We first need gates to ensure one variable does not get two values assigned. 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 **1440 basic gates 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 nine times (once per number), we have 9*8(7)/2 = 252 Toffoli gates, or **1260 basic gates and 252 ancilla bits**.

Step 2: We then need a circuit to ensure 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 29*80 = **2320 basic gates 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 61 * 160 = 9760 basic gates and 160 ancilla bits. Because this is being repeated twice, we have **19520 basic gates 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 **61 basic gates and 1 ancilla bit**.

Step 3: To make sure that at least one of the possible conditions are satisfied after the NAND gates, we need to use a large quantum OR gate. A quantum OR gate costs 2n(NOT gates)+32n–96(Toffoli gates) = 34n – 96 basic gates and two qubits. Since we have to apply a quantum OR gate to 3 sets of inputs, we have 34(80) – 96 + 2(34(160) – 96) = 2640 + 10368 = **13312 basic gates and 6 qubits**. Three of these are output qubits.

Step 4: A final AND gate is needed for 4+288+252 = 544 inputs. This will, in best case, cost 32(544) – 96 = **17312 basic gates and 2 qubits**. The worst case estimate will be $2^{545} - 3 = 10^{\log 2^{545}} - 3 = 10^{164.1} - 3$ basic gates.

The overall best case cost of the circuit is
= 1440 +1260+2320+19520+61+13312+17312
= **55225 basic gates,**
and 288+252+80+320+1+6+2 = **949 qubits**
The worst case cost is approximately $\mathbf{10^{165}}$ **basic gates**.

8. Conclusion

Efficient design of quantum oracles is very important when using Grover Algorithm to search for the solutions to very large problem instances, and an efficient data encoding method can drastically reduce the quantum costs of oracles. In most problems, it was found that Perkowski's encoding method is more efficient than Hogg's, but in some cases, the structure of the problem caused Hogg's encoding method to be more efficient. In Graph Coloring, the cost of oracles built using Perkowski's method was much lower than for Hogg's method, in both qubits and basic gates. In Maximum Clique, the cost of Hogg's method is similar to Perkowski's method. The only difference is the addition of few extra qubits. In SEND MORE MONEY, Hogg's method is extremely inefficient and Perkowski's method is the only viable method to use. However, in SAT, for a large number of AND-ed terms and negated 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. In conclusion, for most cases, Perkowski's method of encoding will cost less than Hogg's method, but in some special cases, Hogg's method will cost less than Perkowski's method. A general pattern for this rule is that for problems in which the variables can take on two values, Hogg's and Perkowski's algorithm costs are very similar. However, when a variable can take on a large number of values such as in Graph Coloring (a node in graph coloring can be one of n colors) and SEND MORE MONEY, Hogg's method becomes less efficient.

9. References

1. *M. Perkowski, S. Hossain, F. Zhao, Minimal Graph Coloring using the Quantum Algorithm of Grover and the Importance of the Quantum Composition/Layout Problem in the Complete Design of Quantum Oracles, Proc. 9th International Workshop on Boolean Problems, September 16-17, 2010, Freiberg (Sachsen), Germany.*
2. *T. Hogg, Quantum Computing and Phase Transitions in Combinatorial Search. Journal of Artificial Intelligence, 4, pp. 91-128, March 1996.*
3. *L.K. Grover, A Fast Quantum Mechanical Algorithm for Database Search. 28th Annual Symposium on the Theory of Computing, pp. 212 – 219, 1997.*
4. *Perry, R. T. (April 29, 2006). Temple of Quantum Computing. http://www.toqc.com/TOQCv1_1.pdf*
5. *Edwards, J. C. (). Principles of NMR. http://www.process-nmr.com/pdfs/NMR%20Overview.pdf*
6. *Barenco, A., Bennet, C. H., & Cleve, R. (1995). Elementary Gates for Quantum Computation. The American Physical Society, 52, 3457-3467.*
7. *Maslov, D., Young, C., & Miller, D. M. (2005). Quantum Circuit Simplification Using Templates. IEEE Computer Society, 1530-1591.*

8. *Maslov, D., & Dueck, G. W. (2004, March 5). Improved Quantum Cost for n-Bit Toffoli Gates*
9. *Murdock, J., Kamischke, E., & Kamischke, E. (2004). Discovering Advanced Algebra: An Investigative Approach. Emeryville, CA: Key Curriculum Press.*
10. *S. 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,*
11. *Nielsen, M. A., & Chuang, I. L. (2000). Quantum Computing and Quantum Information. Cambridge, UK: Cambridge University Press*
12. *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*
13. *Azad Khan, Md.M.H., 2002. Design of full adder with reversible gate. International Conference on Computer and Information Technology. Dhaka, Bangladesh, pp: 515-519.*
14. *Perkowski, M. A. (2009) Quantum Robotics. http://web.cecs.pdx.edu/~mperkows/X/index.html*

10. Results: Costs in terms of Qubits and Gates