2011 41st IEEE International Symposium on Multiple-Valued Logic

Improved Complexity of Quantum Oracles for Ternary Grover Algorithm for Graph Coloring

Yushi Wang
Portland, USA
mseesquared@gmail.com

Marek Perkowski
Department of Electrical Engineering
Portland State University,
Portland, USA
mperkows@ece.pdx.edu

Abstract- The paper presents a generalization of the well-known Grover Algorithm to operate on ternary quantum circuits. We compare complexity of oracles and some of their commonly used components for binary and ternary cases and various sizes and densities of colored graphs. We show that ternary encoding leads to quantum circuits that have significantly less qudits and lower quantum costs. In case of serial realization of quantum computers, our ternary algorithms and circuits are also faster.

1. Introduction

Unlike traditional computers, which use only two states on a bit (0 and 1), quantum computers are able to store a superposition of bits, which allows for faster computation [4,6]. Quantum algorithms have been created to solve many problems more efficiently than standard computers [4,7]. So far, these algorithms have mostly been binary, but few authors [2,11] proposed the multiple-valued counterparts of binary algorithms.

Quantum circuits used in quantum algorithms can realize multi-valued logic. Muthukrishnan and Stroud [1] introduced for the first time examples of such multi-valued (MV) quantum logic gates. Few MV quantum algorithms have also been proposed and have been shown to hold certain advantages over binary quantum algorithms; this was done for Quantum Fourier Transform (QFT) [11], Deutsch-Jozsa [2] and Grover [2]. However, no complete quantum circuits have been synthesized in these papers, and no analysis has been done on the timing or circuit complexities of such circuits.

In this paper, we investigate what practical advantages can be gained from introducing MV quantum oracles for Grover Algorithm [16]. Background is given in section 2 to make the paper self-contained. Using the classical problem of graph coloring as an example, we create a variant of Grover Algorithm that uses ternary circuits (sec. 3). The costs of the binary and ternary oracles are compared in terms of 1-qudit and 2-qudit gate primitives, or gates that cannot be built from smaller gates (sec.4). The correctness of the oracles is verified by simulating a small graph coloring problem with MATLAB (sec. 5). Finally section 6 concludes the paper and outlines future research.

2. Background

2.1. Map and Graph Coloring

The classic map coloring problem is as follows: given a map with disjoint regions, or "countries", the goal is to color all the regions such that no two bordering regions have the same color. Graph coloring is a generalization of map coloring: given a set of vertices or nodes and edges (line segments that connect some nodes), the task is to assign a color to each node in such a way that any two nodes linked by an edge have different colors. These colors are represented in hardware by a number in binary [12,13] or k-valued [2]; ternary in this paper case. A correct coloring is one such that the above condition is satisfied for all pairs of edge-connected nodes. In addition, the graph coloring problem calls for a coloring that uses the minimum number of colors, called the chromatic number. In this paper we denote the number of nodes by **N**, and the number of edges by **c**.

We assume here that the number of colors is given. However, if no such number is known, the minimum number of colors can be determined by finding a maximum clique with the maximum number of nodes in the graph [12]. Fig. 1 has two maximum cliques: {1, 2, 3} and {2, 3, 4}. Each node in every clique must have different colors. Using the size of the clique as a lower bound of the chromatic number, we increase the expected chromatic number by 1 and iterate the algorithm

0195-623X/11 $26.00 © 2011 IEEE
DOI 10.1109/ISMVL.2011.42

until the minimum coloring is found [12]. Such a coloring will have the minimum number of colors. Fig. 1 shows a sample coloring with the minimum number of colors = 3.

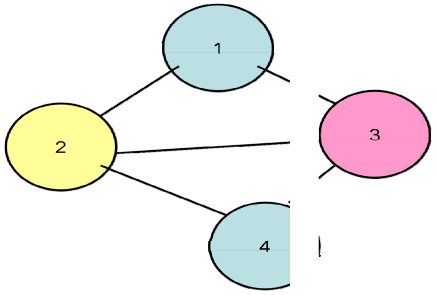

Figure 1 - Simple graph coloring of a graph with 4 nodes and chromatic number 3.

2.2. Basic Concepts of Quantum Computing

In quantum computing, the values 0 and 1 of a bit are replaced by the qubits $|0\rangle$ and $|1\rangle$ (this is the Dirac notation of quantum mechanics [4,14]). In addition to states $|0\rangle$ and $|1\rangle$, a general qubit $|\psi\rangle$ can also be in a linear combination (superposition)

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad (1)$$

where α and β are complex numbers, called amplitudes. $|\psi\rangle$ is a vector in a two-dimensional complex vector space, where $\{|0\rangle, |1\rangle\}$ forms an orthonormal (computational) basis. The matrix representations of $|0\rangle$ and $|1\rangle$ are given by

$$|1\rangle = \begin{bmatrix}0\\1\end{bmatrix} \text{ and } |0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \quad (2)$$

The physical interpretation of $|\psi\rangle$, in (1), is that it coexists in the states $|0\rangle$ and $|1\rangle$. The state $|\psi\rangle$ can store a huge quantity of information in its coefficients α and β, but this information lives in the quantum world. To bring quantum information to the classical world, one must measure the qubit. Quantum mechanics tells us that the measurement process inevitably disturbs a qubit state, yielding a non-deterministic collapse of $|\psi\rangle$ to either $|0\rangle$ or $|1\rangle$: one gets $|0\rangle$ with probability $|\alpha|^2$ or $|1\rangle$ with probability $|\beta|^2$. Thus,

$$|\alpha|^2 + |\beta|^2 = 1. \quad (3)$$

The gates used in this paper are the binary NOT, Controlled-NOT or Feynman, Hadamard, and Multi-controlled NOT or Toffoli gates. The NOT gate, as with classic and fuzzy logic, "flips" a gate by swapping the coefficients α and β. The controlled NOT operates on a bit B with a controlling bit A, which will flip bit B if A = $|1\rangle$. The Toffoli gate is similar to the CNOT gate, but has the coefficients α and β. The controlled NOT o$\rangle$ for the answer bit to be flipped. The Toffoli gate is not a primitive gate, as it is constructed from smaller 2-qubit gates. Finally, the Hadamard Gate multiplies a qubit by the matrix [4,14]

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix} \quad (4)$$

When operated on a $|0\rangle$ or $|1\rangle$ state, this gives a bit equal amplitudes of both states, creating a superposition.

Ternary quantum logic is a generalization of binary logic [5,7], which works with ternary ***qutrits*** with states $|0\rangle$, $|1\rangle$, and $|2\rangle$, which have matrices $|0\rangle = \begin{bmatrix}1\\0\\0\end{bmatrix}$, $|1\rangle = \begin{bmatrix}0\\1\\0\end{bmatrix}$, and $|2\rangle = \begin{bmatrix}0\\0\\1\end{bmatrix}$ (5)

The gates used are the +1, +2, Controlled +1 and +2, Multi-controlled +1, and Chrestenson Transform. The +1 and +2 shift the coefficients of the state up by 1 and 2, respectively [7]. Controlled gates are similar to binary, but instead of requiring 1 in the control qutrit, now require 2 to operate. The Multi-controlled +1 gate is a new concept constructed from smaller gates, and operates exactly like Toffoli gate, in that it requires 1 on each of the control qutrits to add 1 to the answer qutrit. The Chrestenson gate CH is a generalization of the Hadamard gate [5,9], which creates a superposition when it multiplies a qutrit by

$$CH = \frac{1}{\sqrt{3}}\begin{bmatrix}1 & 1 & 1\\1 & d1 & d2\\1 & d2 & d1\end{bmatrix}, \text{where } d1 = -(1+d2) = -\frac{1}{2}(1-\sqrt{3}i) = e^{\frac{2\pi i}{3}} \quad (6)$$

In the circuit synthesis notation, these gates are represented by the following: NOT by the symbol of $\oplus$, Hadamard by a boxed H, Chrestenson by a boxed Ch, and +1 and +2 gates by their respective boxed symbols. Observe that both Hadamard and Chrestenson gates are special cases of QFT [7,2]. Controlled gates will be signified by a large point on

the control wire(s) and a line connecting the control and output wires. Qubits in binary and qutrits in ternary are examples of qudits.

2.3 Grover Algorithm

Grover Algorithm is a quantum search algorithm for "searching an unsorted database" [8,14,16]. The concept is to create a superposition of all possible solution states in a search and use a "Grover operator" (Grover Loop) to increase the probability of the desired search state for measurement. First, the superposition of the qubits created by applying the binary Hadamard (or Chrestenson for ternary) transform to the qubits. Then the compound "Grover operator" operates on the superimposed register of search states and an ancillary qubit. The Grover operator consisting of an oracle that only gives a one for the correct state and an inversion-about-the-average operation HZH, is applied to the superposition, which marks the correct state. The oracle marks the searched state by rotating its phase by π, while the inversion-about-the-average operator increases the amplitude of this state. After sufficient number of iterations, the amplitude of the correct state will be high enough for the superposition to be measured with probability close to 1. Fig. 2 shows a block diagram of Grover Algorithm. Grover Algorithm gives quadratic speedup when compared with the classical search algorithm. It has hundreds of practical applications to solve CSP (Constraint Satisfaction Problems), transforms and others. Not much was however published on designing practical oracles and their complexity beyond [15].

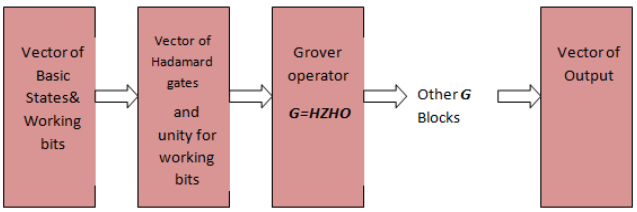

Figure 2 – Outline of Grover Algorithm

In our case, the searched state is a valid coloring of a given graph. Assuming the nodes are in order, the coloring is represented by giving each node a numerical value between 1 and k, each of which determines a color. The ordered string of these values thus represents the colors of multiple nodes.

One extension of Grover Algorithm has already been created by Fan [2], with a generalized QFT proposed in place of the Hadamard transform to produce a version of the algorithm which has multi-valued inputs and outputs. (In ternary, the QFT is the Chrestenson gate.) Fan proved that such a framework would accomplish the goal of increasing the amplitude of the searched state for measurement. However, no circuits or oracles were provided, no quantum costs calculated, nor simulation was used to confirm correct operation.

3. Binary and Ternary Oracle Designs

In a graph, we compare every pair of connected nodes and get c ancilla or working bits, one for each connection. These working bits help the circuit to operate by storing values temporarily before they are used again. Each working bit output will give either a 0 (if the colors of the two nodes are the same) or 1 (if they are not). At the end, a multi-controlled AND gate, the extension of the Toffoli gate, is applied to determine whether the coloring is valid or not. (This gate operates as a NAND, since it EXORS with constant 1). As defined above, we represent the number of nodes in a graph as N. Note that the number of possible colors is also N, since a graph can theoretically be complete (where every node is connected to each other). Thus we always assume that a worst case scenario, where every node has its own color, can exist.

In our Grover Algorithm, we will implement an oracle (a computing black box) that can determine whether a coloring of graph is valid or not. The basis of the oracle for assessing solutions for map colorings is the comparator, which compares two connected nodes by testing whether the qubits for both are exactly the same. Some of the gates and functions constructed here will not be exclusively binary or multi-valued. These functions will operate on ternary or binary inputs, use ternary logic, but give out only 0 or 1 as the result. In this paper, the final oracle is a multi-valued-input/binary-output function. An outline of the algorithm to be constructed is shown in Fig. 3.

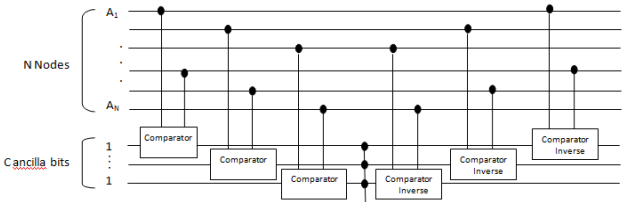

Fig. 3 - Outline of the graph coloring oracle

Note that in the ternary version of the oracle presented here, the comparator machine will have ternary inputs, but gives a binary output for a yes or no response.

In the design and comparison, we make four assumptions:

1. The design of oracles is close enough to optimal to make the comparison fair.
2. The cost of an algorithm may be measured in terms of basic one- and two-qudit gates, the cost of which remains approximately equal across all radices.
3. In any radix, a two-qudit gate costs twice as much as a one-qudit gate.
4. Since the oracles and comparators will be repeated many times in the Grover's Algorithm, it is necessary to eliminate garbage qudits which are not constants 0 or 1.

Two concepts, garbage qudits and mirrors, are important in the circuit synthesis of all oracles. The mirror sub-circuits are often nested. When a working qudit is used in an instance but not returned to its original state, it cannot be used again, but still takes up space. This is called a garbage qudit. To prevent garbage qudits from accumulating in circuits, mirrors are used, which is an inverse of all the gates the qudit passed through, to return it to its original reusable state. Thus for every comparator at the left in Fig. 3 before the global AND gate of the oracle, there exists its mirror circuit, "inverse comparator" that returns an input constant to its original state.

3.1. Binary Comparator

For the binary oracle, there are two possible ways to construct a comparator: mirrored and normal, as shown. In both circuits, the output f is flipped from 1 to 0 iff the input qubits $\{a_1, a_2, \ldots, a_n\}$ are the same as $\{b_1, b_2, \ldots, b_n\}$, which implies an error in the coloring. A basic binary comparator is shown in Fig. 4.

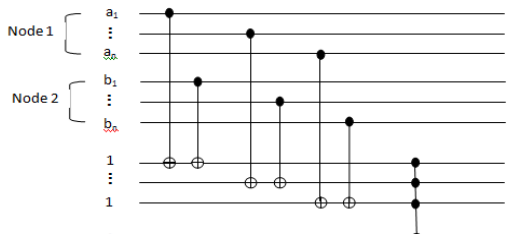

Fig. 4 – Binary comparator circuit without mirrors

In this comparator circuit, the working qubits of 0s become garbage bits, which are not reused and take up space and cost. When the gate operations are not mirrored, this results in n garbage qubits, one per pair of qubits in the input. However, if all gate operations are reversed later through a mirror, then the number of garbage qubits is drastically reduced, as working bits would no longer be required outside of the last answer qubit. A mirrored and logically factored version of the binary comparator is shown in Fig. 5.

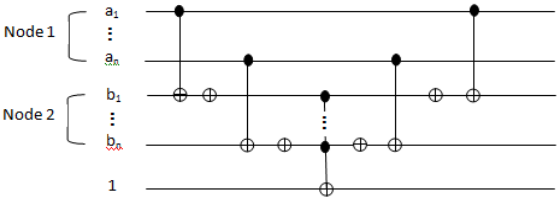

Fig. 5 – Binary Comparator machine, using mirrors

3.2. Ternary Comparator

In the construction of the ternary comparator circuit we use synthesis methods for Muthukrishnan-Stroud gates developed in [5,7,12]. We mirror everything in order to eliminate the cost of wires. The ternary circuit is shown below in Fig. 6. To save space, the mirror of the function is not shown. Instead, the break at the end represents the mirror of all the previous gates before the Toffoli gate, as the whole mirror will not fit.

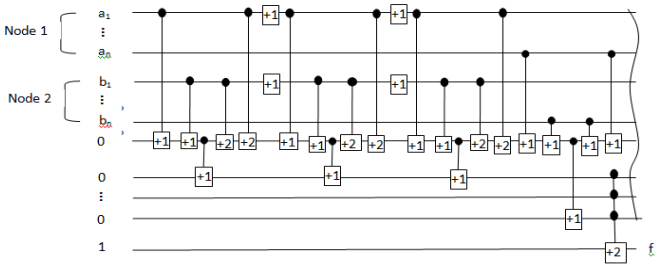

Fig. 6 – Ternary Comparator Circuit

This circuit produces the output f that is 0 iff $a_1 = b_1$, $a_2 = b_2$,…, $a_n = b_n$, and 1 otherwise.

It should be noted that since ternary quantum logic does not have an XOR gate, it takes many extra gates to simulate the XOR. In particular, 3 cases ($a_i = b_i = 0$, $a_i = b_i = 1$, $a_i = b_i = 2$) must be checked, rather than a simple XOR gate from binary.

3.3. Binary and Ternary Toffoli gate

In addition to the large gate at the end of both the binary and ternary oracles, there is a multi-controlled NOT, or Toffoli gate at the end of each comparator in both oracles. The cost of these gates adds significantly to the total cost of the circuit. In binary logic, the cost of the generalized Toffolli gate increases exponentially, or at best, linearly with a slope of 16 and with many garbage bits [2].

In ternary and other multi-valued logic, we can find a way to construct the multi-input Toffoli gate in such a way that the cost of it merely increases linearly. Furthermore, there are no additional garbage outputs, as the only input qutrits are the outputs of the comparators themselves. In this construction, the first qutrit goes through a +1 gate and controls +1 gate operator on the second qutrit, which controls a +1 gate on the third, and so on (Fig. 7). This is a new realization of a binary input/output, internally ternary gate, not known from the literature.

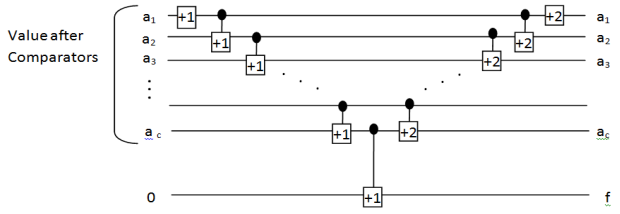

Fig. 7 – Ternary Toffoli NOT gate with *c* inputs

A simple truth table analysis shows that the multi-input ternary Toffoli gate from Fig. 7 has the same function as the binary Toffoli gate. For f to equal 1, the control bit for last controlled +1 gate must be 2. Since all inputs can only be 0 or 1, all must equal 1 for the next bit to trigger, or else the last control bit will not equal 2 as it controls the 0 qutrit's +1 gate.

Furthermore, this gate can be easily generalized to any radix. By changing the +1 gates on all but the last qudit to +(n-2), the multi-input Toffoli gate can be realized in any radix $n \geq 2$.

4. Cost Calculation and Comparison

As stated earlier, the cost calculation and comparisons will be done by evaluating the number of gates in the circuits, using "simple" (1 and 2-qudit) gates as the standard unit of cost. When a gate has more than 2 qudit inputs, the cost is evaluated by calculating the number of 2-qudit gates required to construct the gates.

Again, we assume that the cost of the two-qudit gates in any radix is constant in terms of NMR pulses [7,12,13] (or similar detailed, technology specific measures), though this may be a point of further research.

4.1. Cost of Generalized Toffoli

The multi-controlled NOT gate, also known as the generalized Toffoli, has 2 or more inputs $\{a_1, a_2, \ldots, a_n\}$, operating on an additional input f. If all inputs are equal to 1, then f is flipped. Otherwise, f is not affected. The ternary version will perform the same function, though the function being controlled is +1 or +2. The generalized Toffoli gate is not a primitive gate, as stated in sec. 2. Thus we must calculate its cost in primitive gates to get an accurate cost for the oracle.

In [3], the cost of the binary *n*-bit Toffoli gate was found to be one of the following: For a small number of input gates, such as from 2-5 inputs, the optimal construction of the Toffoli gate currently costs 2^n - 3 basic quantum operations. For larger numbers, the cost grows linearly, at $32n - 96$ gates, plus one garbage bit per input. Thus, the complexity of the Toffoli gate overall grows linearly.

Using the same standard cost of 2-qudit gates, the calculated cost of the multi-valued Toffoli gate is significantly lower. Using the above diagram, one can construct a multi-valued extension using only $2n + 1$ gates, which is also linear, but at 1/8 of the binary version's cost. A graphic comparison of the three is shown in Fig. 8. The cost of the

binary Toffoli gate, when built from primitives, quickly becomes very large, even when using the optimized construction by Maslov [3]. Observe that such gates exist in most if not all oracles [12,13,15] and make high percent of their costs.

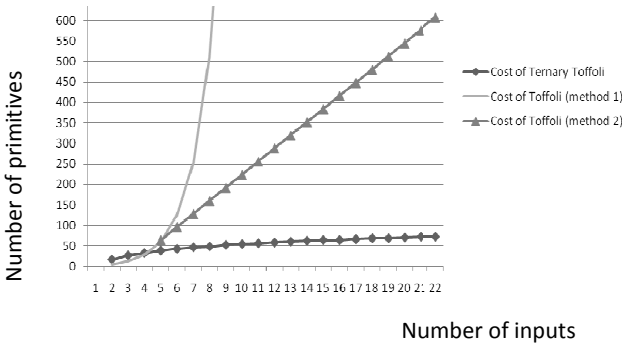

Fig. 8 – Cost of Constructing Toffoli gates from primitives

4.2. Cost of Comparators

As established in Fig. 3, the numbers of comparators in both the binary and ternary oracles are the same. Thus, the cost of the comparators themselves can be directly compared. Using the diagrams of the mirrored comparators, we can see that the cost of the ternary comparator is $34(\log_3 N)$ two-qutrit gates, $4(\log_3 N)$ single-qutrit gates, and one Toffoli gate with $\log_3 N$ inputs. The binary comparator costs $4(\log_2 N)$ two-qubit gates plus one $\log_2 N$-input Toffoli gate. The costs of the comparators minus the Toffoli gate is shown in Fig. 9.

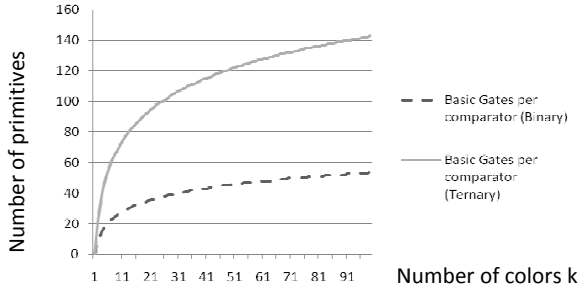

Fig. 9 – Cost of Comparators, excluding Toffoli, using primitives

Using the formulae for the cost of Toffoli in terms of two-qubit gates, the cost jumps significantly to $4(\log_2 N) + 32(\log_2 N) - 96$ gates, assuming $N \geq 5$. For $N < 5$, the cost is $4(\log_2 N) + 2^{(\log 2N)} - 3$ gates.

Now factoring the cost of Toffoli gates into the equation, the cost becomes much more favorable towards ternary logic, as shown in Fig. 10.

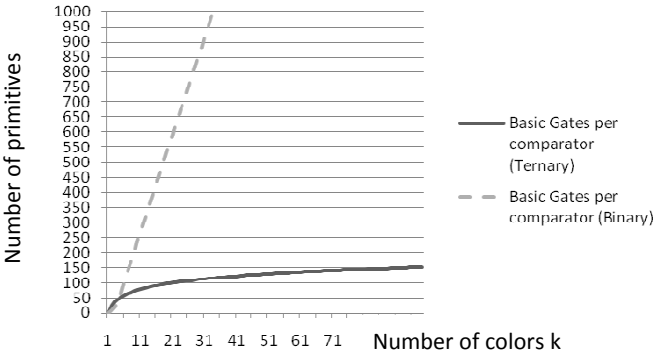

Fig. 10 – Cost of Comparators, including Toffoli, in primitives

5. Simulation of Graph Coloring Oracle

To prove that the algorithm proposed above works, we use MATLAB to simulate the comparator circuits described in Fig. 4. However, MATLAB requires that gates be simulated through Kronecker products, and as such, cannot accommodate crossovers used in the original proposed design. We use the same gates and sequence as above, but to resolve crossovers, we also use the SWAP gate, as shown in Fig. 11. Thus the circuit satisfies the Linear Nearest Neighbor (LNN) Model.

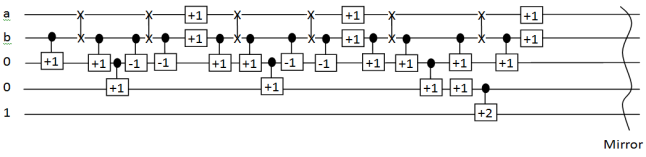

Fig. 11 – Simulated Comparator Circuit in MATLAB

We implement this comparator into Grover Algorithm that can determine whether a coloring is good for a complete 3-graph, Fig. 12. Furthermore, we implement this into Fan's framework [2] for Grover Algorithm by creating the ternary Grover Algorithm and implementing the oracle in it. Unfortunately, MATLAB has memory constraints limit our simulation sizes to having at most 6 connections and nodes total. The full simulation of one loop of Grover operator is shown in Fig. 12.

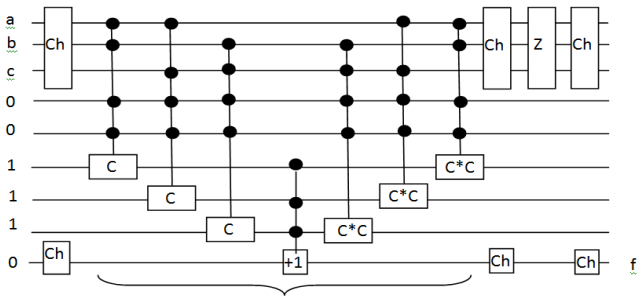

Fig. 12 – Simulation of one iteration of the Grover with oracle.

In the above simulation, the comparators are mirrored. It should be noted that to mirror these, two comparators were used in quick succession. Since the functions are in base 3, performing any ternary function (with a combining operator being group) three times will revert the state of the output to the original input [12,7].

Grover Algorithm has been shown to require approximately $\sqrt{M}$ iterations rounded up, where M is the number of "items in database". Since we have 3 nodes, each with 3 states, the total number of items is 3^3, or 27. Thus, our solution algorithm should have $\lceil\sqrt{27}\rceil = 6$ iterations of the Grover operator. With each successive application of the operator, the probabilities of the "good" states should increase in amplitude, whereas the "bad" states should decrease in amplitude until they reach ~0. This effect is clearly shown in the next two graphs, Figs. 13, 14, which depict the distribution of amplitudes after 2 and 6 applications of the Grover loop, respectively.

The simulation results were verified by examining the amplitudes and the locations of the two distributions. After 2 Grover loops, the highest amplitude of the output was .655, whereas after 6 Grover loops, the highest amplitude had increased to .87.

Furthermore, the accuracy was verified by examining the locations of the peaks. Since the simulator assigns indices by counting up from the first input (0000000) to the last, (2222222) in ternary and assigning them values, we may see the inputs by first subtracting one (it started counting at 0000000) from the index and converting to ternary to get a string of inputs (the seven input qutrits).

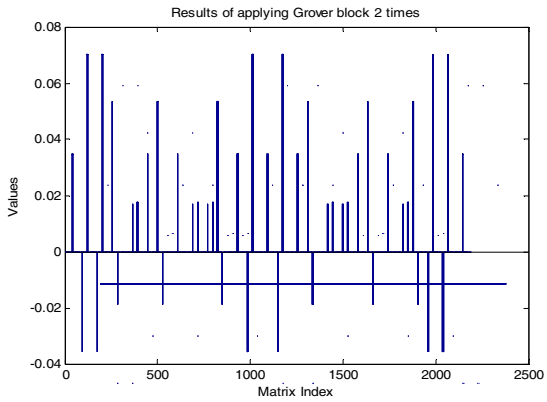

Fig. 13 - Distribution of Amplitude after 2 Grover Loops

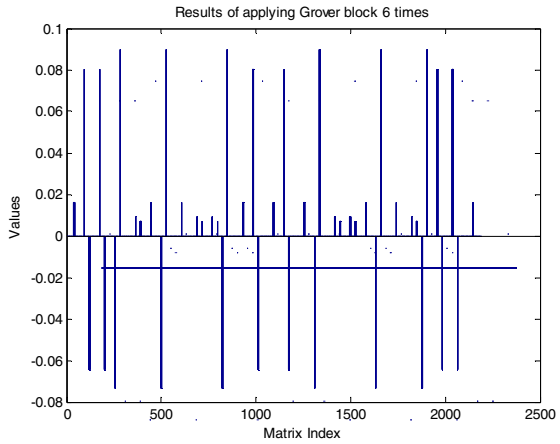

Fig. 14 – Distribution of Amplitude after 6 Grover Loops

One such peak is at index 1741, which had amplitude .87. Subtracting 1 and converting to ternary gives string 2101110. The first three digits, 210, are the colors of a, b, and c, respectively. Similarly, the other 5 peak values give correct colorations as well, giving us the verification that the simulation is correct.

6. Conclusions

We confirmed here, on a graph coloring problem, the theoretical results from [2] that the ternary oracle implemented in the multi-valued Grover Algorithm yields valid solutions. Furthermore, this algorithm, which obviously uses fewer qudits, was also proven to have a lower cost of oracles in gate primitives for large graphs. This translates to a lower delay in case of serial quantum computer model. The binary solution algorithm has a lower cost for small graphs because the ternary oracle has no direct XOR function, and the synthesis of the XOR function in ternary required many gates.

However, the fewer qudits used in ternary and the lower cost of the Toffoli offset this when c, the number of edges in the graph, is approximately 6, as shown in Fig. 10. The Graph Coloring Grover Algorithm has several applications in logic synthesis, scheduling, allocation, planning, robot motion, robot communication, resource allocation, conflict resolution, floor-planning, and many others. It can serve as a "generic CSP solver" similar to a general SAT solver.

Quantum circuits are well-known for their intrinsic ability to perform multi-valued logic. The constructions of Toffoli gate and oracles used in the ternary oracle here can be generalized to arbitrary radices. The algorithm provides groundwork for further MV quantum logic synthesis. The binary Toffoli gate is often used in quantum logic synthesis, since it represents the AND gate. The ternary Toffoli gate, invented in this paper, performs a similar function and will be used as a building block for ternary logic synthesis in the future. Furthermore, the ternary Toffoli gate was shown to have 1/8 the cost of the binary Toffoli gate, as it scaled linearly with the number of inputs with slope two, whereas the binary gate required 16 more 2-qubit gates for each additional input.

Finally, the lower cost of the ternary quantum algorithm implies that, in general, multi-valued quantum logic can be more efficient than binary quantum logic in terms of gates and wires. The less gates and wires used in these algorithms translate to less computing time and perhaps power consumption used.

This paper used some simplifications. The oracle was simulated in MATLAB, so some of the physical restrictions of quantum computers were ignored. Future simulation will use QMDD [17]. Finally, the costs of the oracles were calculated in terms of primitive gates. This gives a cruder measure of the cost than calculating the actual number of NMR or other pulses required to generate each gate. These restrictions are dealt with in the next papers.

References

[1] A. Muthukrishnan, and C.R. Stroud, Jr. (2000). *Multivalued logic gates for quantum computation*. Phys. Rev. A 62, 52309.

[2] Y. Fan, (2008). *Applications of multi-valued quantum algorithms*. arXiv: Available at arXiv:0901.4163v4. also, Y. Fan: A Generalization of the Deutsch-Jozsa Algorithm to Multi-Valued Quantum Logic. *Proc. ISMVL* 2007: 12.

[3] *D.* Maslov, and G.d Dueck. (2004). *Improving quantum cost for n-bit Toffoli gates*. arXiv: Available at arXiv:quant-ph/0403053v1

[4] Ph. Kaye, et. al. (2007). *An Introduction to Quantum Computing*. Oxford: Oxford University Press Print.

[5] A.N. Al-Rabadi et al., *Multiple-Valued Quantum Logic. Proc. 11th ULSI*, Boston, MA., May 2002.

[6] R. Jozsa, (1999). *Searching in Grover's algorithm*. arXiv: Available at quant-ph/9901021.

[7] M. Perkowski, (2015). Lecture notes on Quantum Computing. http://web.cecs.pdx.edu/~mperkows/

[8] C. Lavor, and L.R.U. Manssur, (2003). *Grover's algorithm: quantum database search*. arXiv. Available at quant-ph/0301079.

[9] R. Jozsa, (1997). *Quantum algorithms and the Fourier transform*. arXiv. Available at quant-ph/9707033.

[10] A. Fabrikant, and T. Hogg, (2002) *Graph coloring with quantum heuristics*. American Association for Artificial Intelligence. Available at alex.fabrikant.us/papers/fh02.pdf

[11] Z. Zilic, K. Radecka, Scaling and Better Approximating Quantum Fourier Transform by Higher Radices. IEEE Trans. Computers 56 (2): 202-207 (2007)

[12] S. Hossain, "Classical and Quantum Search Algorithms for Quantum Circuits and Optimization of Quantum Oracles", *Ph.D. Dissertation*, 2009, Portland State University.

[13] 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. ISBP 2010.*

[14] M. Nielsen and I. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, 2000.

[15] L. Li, M.A. Thornton, M.A. Perkowski, "A Quantum CAD Accelerator Based on Grover's Algorithm for Finding the Minimum Fixed Polarity Reed-Muller Form," *Proc. ISMVL 2006*, pp. 33.

[16] L. Grover, "A fast quantum mechanical algorithm for database search," *Proc. 28th ACM Symp. on Theory of Computing* 1996, pp. 212-219 1996.

[17] M.D. Miller, and M.A. Thornton, "QMDD: A decision diagram structure for reversible and quantum circuits," *Proc. 2006 ISMVL*, Singapore, 6 pp, May 2006.