A Quantum Triangle Finding Algorithm and Quipper

Geoffrey G. Schumacher
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
schum476@morris.umn.edu

ABSTRACT

Quantum computing has undergone significant development in the last two decades. Almost twenty years ago, L.K. Grover developed a search algorithm that runs in $O(\sqrt{n})$. Since then, many other quantum algorithms have been introduced; some of which utilize earlier algorithms like that developed by Grover. While physical implementation of a quantum device remains the most restricting bottleneck of the field, there are also other aspects that have restricted its advancement. One such aspect is the topic of quantum programming languages. While there are numerous quantum algorithms in existence, there are relatively few languages that have been developed with the capability of implementing these algorithms in a practical manner. These programming languages help communicate and formalize established algorithms.

Keywords

Geoff's Keywords: Quantum Algorithm, Qubit, Superposition

1. INTRODUCTION

In the early 1980's, Richard Feynman observed that some quantum mechanical effects can not be simulated efficiently on a *classical* computer system. This observation brought upon the supposition that communication in general might be performed more efficiently if it made use of these quantum effects. [1] The form of computing that takes advantage of quantum effects is known as *quantum computing.*

The main benefit of quantum computing results in higher efficiency due to the amount of true parallelism that occurs in a quantum system. As [1] noted, the time it takes to perform certain computations on a classical system can be decreased by adding parallel processors to the system. An exponential decrease in time for a classical system then requires an exponential increase in the number of parallel processors. This then requires an increase of the same rate in the amount of physical space. However, in a quantum system, an exponential increase in parallelism requires only a linear increase in the amount of space needed. This effect is known as *quantum parallelism.* [1]

The drawback is that the access to the results of these parallel computations is limited. Reading the results is similar to making a measurement in the system. This measurement disturbs the quantum state, meaning that you can only read the result of one parallel thread at a time. On top of that, the measurement needed is probablilistic, which means you can not even choose which thread you will read. [1]

This paper discusses a quantum triangle finding algorithm and the quantum programming language Quipper, and is organized as follows. In Section 2, we briefly define key graph theory terms. Section 3 covers the basics of quantum computing. In Section 4, we introduce a quantum algorithm known as Grover's Algorithm. Section 5 walks through a quantum algorithm used to find triangles in undirected graphs. Part of this algorithm uses Grover's Algorithm. In Section 6, we introduce Quipper, a scalable quantum programing language. We then elaborate on an implementation of another triangle finding algorithm in Quipper. Conclusions are provided at the end.

2. BACKGROUND

In order to understand the afore mentioned algorithms, as well as quantum computing in general, some background knowledge must be stated about graph theory and quantum mechanics.

2.1 Graph Theory

This subsection defines key terms of graph theory, which will be used to discuss a quantum algorithm for finding triangles in a graph.

Definition 1. A **graph** G consists of two finite sets: a set $V(G)$ of **vertices** and a set $E(G)$ of **edges,** where each edge is associated with a set consisting of either one or two vertices called its **endpoints.** An edge with just one endpoint is called a **loop,** and two distinct edges with the same set of endpoints are said to be **parallel**. [2]

Definition 2. A **simple graph** is a graph that does not have any loops or parallel edges. In a simple graph, an edge with endpoints v and w is denoted $\{v, w\}$ [2]

Definition 3. Let n be a positive integer. A **complete graph on n vertices,** denoted K_n**,** is a simple graph with n vertices $v_1, v_2, ..., v_n$ whose set of edges contains exactly one edge for each pair of distinct vertices. [2]

UMM CSci Senior Seminar Conference, April 2014 Morris, MN.

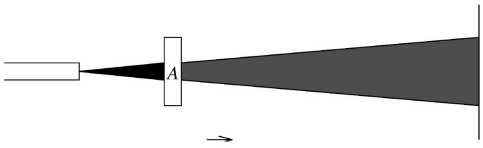

Figure 1: Photons from the light source pass through filter A.

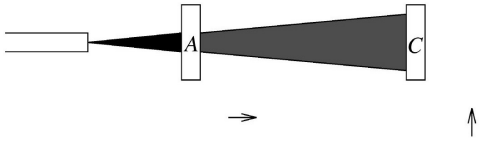

Figure 2: Photons from the light source pass through filters A and are stopped at filter C.

Definition 4. A graph H is said to be a **subgraph** of a graph G iff every vertex in H is also a vertex in G, every edge in H is also an edge in G, and every edge in H has the same endpoints as in G. In a graph G, a complete subgraph on three vertices is called a **triangle**. [2]

2.2 Notation

Here, we introduce necessary notations and terms. The set {1, 2, ... , n} is denoted by $[n]$. The neighborhood of a vertex, $v \in [n]$ in G is denoted by $\nu_G(v) = \{b|(v, b) \in G\}$. That is, the set of all vertices adjacent to v in G is denoted by $\nu_G(v)$. The complete graph on a set $\nu \subseteq [n]$ is denoted by ν^2. Let $t(G)$ denote the number of triangles in G. The number of paths of length two from $a \in [n]$ to $b \in [n]$ in G is denoted by $t(G, a, b)$. [6]

2.3 Quantum Mechanics

In this paper we give a brief introduction to the relevant aspects of quantum mechanics. To begin, a common notation used in quantum mechanics is to represent the quantum state of x as $|x\rangle$. Next, quantum mechanics will be introduced through a simple experiment as described in [1]. This experiment illustrates key aspects of quantum mechanics needed to understand quantum computing.

Photon Polarization Experiment. The only equipment needed for this experiment are a strong light source (such as a laser pointer) and three polaroids (or polarization filters). The filters A, B, and C are polarized horizontally ($\rightarrow$), at 45° ($\nearrow$), and vertically ($\uparrow$), respectively. We first insert filter A into the beam of light, as seen in Figure 1. The intensity of the output light will have half that of the input light, and all of the outgoing photons will be horizontally polarized. Next, filter C is inserted into the output light of A, as seen in Figure 2. The intensity of the net output is then zero. Finally, we insert B between filters A and C, as seen in Figure 3. Now, the net output is one eighth of the original output from the light source. This is an ambiguous effect. Classical experience would suggest that adding filters can only reduce the number of photons that pass through.

The Explanation. The polarization of a photon can be expressed as $a|\rightarrow\rangle + b|\uparrow\rangle$. Here, a and b are complex numbers such that $|a|^2 + |b|^2 = 1$ and $|\rightarrow\rangle$, and $|\uparrow\rangle$ represent horizontal and vertical polarizations (respectively). To determine the state of a particular photon, the system (the beam of light in this case) needs to be measured. Here, a polaroid acts as a measuring device for the quantum state of photons. The probability of a photon measured as being polarized horizontally is $|a|^2$ and that of being polarized vertically is $|b|^2$. However, measurement of all photons in the system changes the state of the entire system to one of these two polarizations. In this experiment, the entire state of the output of the light passing through a certain filter is polarized in the same way as the filter it passed through. For example, the state of the system after passing through filter A was polarized horizontally. That is, all the photons that pass through A are horizontally polarized.

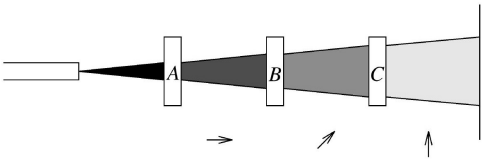

Figure 3: Photons from the light source pass through all filters A, B, and C.

Assuming that the light source produces randomly polarized photons, A will measure 50% of the photons as horizontally polarized. In the second scenario involving only A and C, all the photons that reach C are polarized horizontally, so at this point, the probability of measuring the state of the system as vertical is 0%, so no photons will pass through C. This is why the intensity of the light after C is zero in the first scenario. While, A and C measure the photons on the set of $\{|\rightarrow\rangle, |\uparrow\rangle\}$, B measures on the set of $\{\frac{1}{\sqrt{2}}(|\rightarrow\rangle + |\uparrow\rangle), \frac{1}{\sqrt{2}}(|\rightarrow\rangle - |\uparrow\rangle)\}$ which can be written as $\{|\nearrow\rangle, |\nwarrow\rangle\}$. Here, $|\nearrow\rangle$ and $|\nwarrow\rangle$ represent a polarization of 45° and 135° respectively. So in the third scenario, photons that pass through A have a 50% chance of being measured as the 45° polarization when passing through B. When photons of this polarization reach C, they have a 50% chance of being measured as $\uparrow$. Therefore, one eigth of the total photons pass through all filters A, B, and C.

This relates to quantum computing in handling the state of a quantum bit as well as an entire quantum system (a system of multiple quantum bits). When the state of a quantum bit is measured, it changes the state of the entire system to the same state of the quantum bit being measured. Therefore, there is no value in measuring the state of one bit in the quantum system and then immediately measuring the state of another bit. After a measurement is made, the state of the whole system needs to be reset to what it was before the measurement was made (or possibly a different state). While polaroids A and C act as measuring devices, polaroid B resets the state of the system in the exact same matter as a quantum gate that is discussed in section 3.2.

3. QUANTUM COMPUTING

There are fundamental differences between classical and quantum computing. This section will discuss two that lay out the foundation of quantum computing, as well as a vital concept necessary for many quantum algorithms.

3.1 Qubits

One of the fundamental differences between classical and quantum computing goes down to the level of bits. In clas-

sical computing, a bit is represented by the set of values {0, 1}. In quantum computing, these values can be represented by the set of quantum bit, or *qubit*, values {$|0\rangle$, $|1\rangle$}. The bit values {0, 1} are traditionally thought as the state of a switch, either off or on, whereas the qubit values of {$|0\rangle$, $|1\rangle$} can be interpreted as corresponding to the spin of an electron or photon, either $\rightarrow$ or $\uparrow$. The important difference between the two is the state in which a bit could exist. A classical bit can be in a state of either 0 or 1 (exclusively). However, a qubit can be in a *superposition* of $|0\rangle$ and $|1\rangle$. That is, a qubit can be both $|0\rangle$ and $|1\rangle$ at the same time, such as $a|0\rangle + b|1\rangle$, where a and b are complex numbers that satisfy the equation $|a|^2 + |b|^2 = 1$. A qubit must be measured to determine whether it is in a state of $|0\rangle$ or $|1\rangle$ at any given moment, where $|a|^2$ is the probability that the qubit is $|0\rangle$ and $|b|^2$ is the probability that it is $|1\rangle$. The true power of qubits comes in numbers. In a classical system of n bits the combination of said bits can only be in one state at a time. However, in a quantum system of n qubits, the combination of said qubits can be in a superposition of 2^n different states. [1]

3.2 Quantum Gates: the Walsh-Hadamard Transformation

Gates in classical computing can be generalized to transformations of classical bits. For example, the NOT gate is the transformation of one input bit into the opposite bit:

$$\begin{aligned} NOT: \quad & 0 \rightarrow 1 \\ & 1 \rightarrow 0 \end{aligned}$$

In quantum computing, one important transformation on qubits is the *Hadamard transformation*, defined by:

$$\begin{aligned} H: \quad & |0\rangle \rightarrow \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ & |1\rangle \rightarrow \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{aligned}$$

When applied to $|0\rangle$, the transformation H creates a superposition state of $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. When H is applied to n qubits individually, it generates a superposition of 2^n possible states. These states could be viewed as the binary representations of the numbers from 0 to 2^{n-1} (inclusive). The transformation that applies H to n qubits is known as the *Walsh*, or *Walsh-Hadamard*, transformation W [4].

3.3 Oracles

Many quantum algorithms use the concept of an *oracle* at some point in their respective procedures. As Green *et al* described

> "An oracle is usually given by a classical function $f : Bool^n \rightarrow Bool^m$, describing some aspect of the input to the algorithm..." [4]

In other words, given an input, the oracle will produce an output that represents an answer to a question. For example, suppose the oracle described by the function f above represents predicting the results of flipping a weighted coin based on a previous trial. Given n boolean values representing the results of n flips of said weighted coin, the oracle would then produce a prediction of the next m flips.

For many algorithms that use an oracle, their complexity is measured by how many times the oracle is queried. This is known as the query complexity of an algorithm.

A quantum algorithm is similar in principle, however the exploitation of quantum properties allows it to be more parallel in nature when evaluating its response. This also makes it probabilistic. That means that the response the oracle gives is not guaranteed to be correct. The probability that the result from a specific oracle is correct will depend on its implementation.

4. GROVER'S ALGORITHM

In 1996, Lov K. Grover published a paper for a quantum search algorithm with efficiency $O(\sqrt{N})$. This algorithm is often referred to as *Grover Search*. He generalized the problem which his algorithm was used in this way: Given a system of $N = 2^n$ states which are labelled $S_1, S_2, ..., S_N$, where the 2^n states are represented as n bit strings and $C(S_v)$ is a boolean function that returns true when a given state has the trait we're looking for and false otherwise, there is a unique state, called S_v, which satisfies the condition $C(S_v) = 1$, while for the other states S, $C(S) = 0$. The problem here is to identitfy the state of S_v. [5]

One noteworthy trait of this algorithm that should be stated is that it is probabilistic. That is, while the probability of the final result being the one desired is sufficiently high, it is not 1. Another important aspect noted by [3] is that Grover Search does not search through lists. It actually searches through funtion inputs. The algorithm takes a function, searches through the implicit list of possible inputs to the function, and returns the single input that causes the function to return true with high probablilty. [3]

The fundamental idea of this algorithm is to set the system with an equal probability to be in each state and then eliminate non-solutions. If a given state is rejected by function C, then it is already known to not be the solution. If a given state is accepted by C, then it is disturbed slightly to increase the probability that the final measurement is the desired result. After this step, a diffusion transform D is used. This transform D is defined by the matrix D as:

$D_{ij} = \frac{2}{N}$ if $i \neq j$ & $D_{ii} = -1 + \frac{2}{N}$.

In his paper, Grover describes his algorithm as follows:

> (i) Initialize the system to the distribution: $(\frac{1}{\sqrt{N}}, \frac{1}{\sqrt{N}}, \frac{1}{\sqrt{N}} ... \frac{1}{\sqrt{N}})$, i.e. there is the same probability to be in each of the N states. It is possible to obtain this distribution in $O(\log N)$ steps.
>
> (ii) Repeat the following unitary operations $O(\sqrt{N})$ times:
>
> (a) Let the system be in any state S: in case $C(S) = 1$, rotate the phase by π radians, in case $C(S) = 0$, leave the system unaltered.
>
> (b) Apply the diffusion transform D
>
> (iii) Sample the resulting superposition. If there is a unique state S_v such that $C(S_v) = 1$, the final state is S_v with a probability of at least $\frac{1}{2}$.

The reason for this algorithm's efficiency stems from the exploitation of superposition. Grover search finds satisfying inputs to a function by creating a uniform quantum superposition of states and then repeatedly cancels out non-solutions [3]. In the algorithm above, step (i) puts all S_N states in a uniform superposition of $\frac{1}{\sqrt{N}}S_1 + \frac{1}{\sqrt{N}}S_2 + ... + \frac{1}{\sqrt{N}}S_N =$

$\frac{1}{\sqrt{N}}(S_1 + S_2 + ... + S_N)$. Step (ii), in which the state of the system is disturbed slightly to cancel out some of the non-solutions, is then performed $O(\sqrt{N})$ times. In the third and final step the state is measured. If there actually was a unique state S_v, such that $C(S_v) = 1$, then the probability of step (iii) measuring the state of the system as S_v is at least $\frac{1}{2}$.

When Grover's Algorithm is applied to a graph, the function C is used to check the "markedness" of a vertex v. If v is marked, then $C(v) = 1$, otherwise, $C(v) = 0$. With the parallelized nature of Grover's Algorithm, it is able to run C on all vertices of a given graph at once.

5. QUANTUM TRIANGLE FINDING ALGORITHM

5.1 The Problem

One instance of the *Triangle Finding Problem* is described by [4]: An undirected simple graph G contains precisely one triangle, $\triangle$. G is given by an oracle function f, such that for any two nodes, v and w, of the graph, $f(v, w) = 1$ if (v, w) is an edge of G and $f(v, w) = 0$ otherwise. To solve this version of the Triangle Fnding Problem is to find the set of vertices $\{e_1, e_2, e_3\}$ that form $\triangle$ by querying f.

5.2 Grover Based Subroutine

The triangle finding algorithm discussed in [6] is actually a classical algorithm where a quantum speedup can be applied to the partitioning of the edges. To do this, it uses a form of Grover's searching algorithm in its subroutines. This version of the algorithm is known as **Safe Grover Search**(t), which is based on t iterations of Grover Search. These iterations of Grover Search are then followed by a checking process for markedness of output instances. An important fact that [6] notes is that Safe Grover Search will always reject if there is no marked item or otherwise, find a marked item with a probability of at least 1 - $\frac{1}{N^c}$

5.3 The algorithm

Magniez *et all* defines their triangle finding algorithm as follows:

> **Combinatorial Algorithm**$(\epsilon, \delta, \epsilon')$
>
> 1. Let $k = \lceil 4n^{\epsilon} \log n \rceil$
>
> 2. Randomly choose $v_1, ..., v_k$ from $[n]$ (with no repetition)
>
> 3. Compute every $\nu_G(v_i)$
>
> 4. If $G \cap \nu_G(v_i)^2 \neq \emptyset$, for some i, then output the triangle induced by v_i
>
> 5. Let $G' = [n]^2 \setminus \cup_i(\nu_G(v_i)^2)$
>
> 6. Classify the edges of G' into T and E such that:
>
> -T contains only $O(n^{3-\epsilon'})$ triangles
>
> -$E \cap G$ has size $O(n^{2-\delta} + n^{2-\epsilon+\delta+\epsilon'})$
>
> 7. Search for a triangle in G among all triangles inside T
>
> 8. Search for a triangle of G intersecting with E
>
> 9.Output a triangle if it is found, otherwise reject

Here, n refers to the number of vertices in the given graph G. This algorithm takes three numeric inputs: ϵ, δ, ϵ'. These inputs determine the query complexity of the algorithm. The query complexity for this algorithm is calulated by the following formula:

$O(log_e(n)(n^{1+\epsilon} + n^{1+\delta+\epsilon'} + \sqrt{n^{3-\epsilon'}} + \sqrt{n^{3-min(\delta,\epsilon-\delta-\epsilon')}}))$

For example, with $\epsilon = \frac{3}{7}$, $\epsilon' = \delta = \frac{1}{7}$ the total number of queries is $O(log_e(n)n^{1+\frac{3}{7}})$. [6]

The algorithm begins by evaluating k, a constant integer value, as described in the algorithm above. Then, k vertices are randomly selected from G, with no repetitions. The neighborhood of each vertex from the random sample is then calculated. If the intersection of G and the complete set of vertex pairs of one of these neighborhoods is not empty, then this intersection is a triangle and is returned. This is one step where Safe Grover Search would be used because we are trying to find an edge in G. If no triangle is found here, G' is initialized to be the complete set of vertex pairs from G, minus all the vertex pairs just checked from the sample set.

At this point, a "classification" step, which is defined below, is implemented. The classification method takes G', δ, and ϵ' as inputs. Two empty sets, T and E, are initialized. The goal of the classification step is to make T have a limited number of triangles and make E have a limited size. The following steps of this paragraph are then repeated until G' is empty. While there exists an edge (v, w) such that the number of paths of length two from v to w is strictly less than $n^{1-\epsilon'}$, add said edge to T and remove it from G'. Then, a vertex of non-zero degree is picked. That is, a vertex with at least one adjacent vertex is picked and its neighborhood is calculated. If the degree, or the number of edges, of this neighborhood is less than or equal to $10 \times n^{1-\delta}$, then add all of the edges to E and remove them from G'. If the degree of this neighborhood is greater than or equal to $\frac{1}{10} \times n^{1-\delta}$, then this neighborhood is reexamined. Step 4 is reapplied to this neighborhood to check for a triangle. This is another instance where Grover Safe Search is used. If one is found, then it is returned. If no triangle is found, then all edges in $G'(\nu_G(v), \nu_{G'}(v))$ are added to E and removed from G'. Here, $G'(\nu_G(v), \nu_{G'}(v))$ refers to all edges connecting vertices in the neighborhood of v in G to vertices thereof for v in G'.

Step 2b uses the following sampling strategy:

> Set a counter C to 0. Query $\lceil n^{\delta} \rceil$ random edge candidates from the complete set of pairs of v with $[n]$. If there is an edge of G among them, add one to C. Repeat this process $K = c_0 log(n)$ times, where c_0 is a sufficiently large constant. Accept the low degree hypothesis if by the end $C < K/2$, otherwise accept the large degree hypothesis.

Here, a certain number of random edges are selected from the complete set of pairs of v with $[n]$. Safe Grover Search is then used to check if one of these edges is also in G. If so, then the counter C is incremented. This process is then repeated a sufficient number of times. If, at the end of this process, the counter is strictly less than $K/2$, then the low degree hypothesis is accepted. Otherwise, the large degree hypothesis is accepted.

This classification step is formally defined as:

Classification(G', δ, ϵ')

1. Set $T = \emptyset$, $E = \emptyset$

2. While $G' \neq \emptyset$ do

(a) While there is an edge $(v, w) \in G'$ s.t. $t(G', v, w) < n^{1-\epsilon'}$, Add (v, w) to T, and delete it from G'

(b) Pick a vertex v of G' with non-zero degree and decide

1. low degree hypothesis: $|\nu_G(v)| \leq 10 \times n^{1-\delta}$

2. high degree hypothesis: $|\nu_G(v)| \geq \frac{1}{10} \times n^{1-\delta}$

(c) If Hypothesis 1, add all edges (v, w) of G' to E, and delete them from G'

(d) If Hypothesis 2, then

i. Compute $\nu_G(v)$

ii. If $G \cap \nu_G(v)^2 \neq \emptyset$, output the triangle induced by v and stop

iii. Add all edges in $G'(\nu_G(v), \nu_{G'}(v))$ to E, and delete them from G'

Now we are out of the classification step. At this point, we know that T contains a certain number of triangles and $E \cap G$ has a fixed size. The final three steps are then self explanatory. Safe Grover Search is used for both steps 7 and 8. This algorithm will always reject if there is no triangle in the given graph G. Otherwise, if G does contain one triangle, this algorithm will return said triangle will probability $1 - O(\frac{1}{n})$.

6. QUIPPER: A SCALABLE QUANTUM PROGRAMMING LANGUAGE

While much has been done in the last twenty years to develop quantum algorithms and the theoretical side of quantum computing, relatively little advancement has been made to develop applications of these algorithms and theories. Namely, the subject of quantum programming languages has not been explored extensively. However, in 2013, Green *et al* introduced a new quantum programming language called *Quipper* [4].

While there are very few functioning quantum computers in existence, there are benefits to developing quantum programming languages. Namely, it allows discussion about efficiency in terms of qubits and quantum gates needed to implement the quantum algorithms that have already been developed.

6.1 Knill's QRAM Model

The authors of [4] describe Quipper as being run on a quantum device known as *Knill's QRAM model for quantum computation*, in which a quantum computer is a specialized device that is attached and controlled by a classical computer. The quantum device contains n individually addressable qubit (for some fixed n) and is controlled by only two kinds of instructions. The first instruction is of the form: "apply built-in unitary gate U to qubit k," "apply gate V to qubits j and k," etc. The quantum device will respond with an acknowledgement that the operation has been performed. The second takes the form: "measure qubit k." The quantum device then returns the measurement result, which is either 0 or 1.

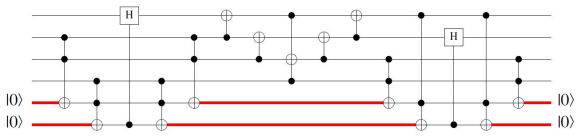

Figure 4: A quantum circuit with two ancillas, and the regions where the ancilla is in state $|0\rangle$ are highlighted.

6.2 Quantum Circuits: Ancillas and Scope

A number of quantum algorithms require *ancillas*. That is, they require "scratch space" qubits whose state is $|0\rangle$ outside of certain well-defined regions where the ancilla is being used. For situations where all gates must be unitary, ancillas usually are treated as additional global inputs and outputs to the algorithm. These variables are assumed to be in a state $|0\rangle$ at the start at the algorithm and expected to be reset to $|0\rangle$ after each use. The regions where an ancilla could be used is referred to as the *scope* of the ancilla. Figure 4 shows an example of a quantum circiut with two ancillas (the bottom two lines). Circuits are read left to right, where the horizontal lines represent wires, boxes representing quantum gates, and vertical wires representing controls on a gate. [4] The box with an H label is an implementation of the Hadamard transformation discussed in Section 3.

6.3 Automatic Generation of Oracles

Implementing a quantum oracle by hand typically requires four steps. First, the oracle must be expressed as a classical program acting on classical data types. Next, this program is translated into a classical circuit for the given input size. Then, the classical circuit is converted into a quantum circuit. This third step potentially introduces many ancillas to hold intermediate, or "scratch space" values. The final step is to make this circuit reversable. One of the powerful aspects of Quipper is that it has buit in functionality that automates the second, third, and fourth steps of this process. So given a classical representation of an oracle, Quipper can automatically generate an equivalent quantum oracle. [4]

6.4 Procedural Paradigm

The basic philosiphy of Quipper's procedural paradigm is that qubits are held in variables and gates are applied to them one at a time. Subroutines can be used to group gate-level operations together where the programmer finds it useful. When writing this kind of procedural code, the programmer may safely pretend (though this is not actually true) that the variables hold actual physical qubits and that the specified gates are aplied to them in real time. [4]

The basic abstraction offered by Quipper is that a quantum operation is a function that inputs some quantum data, performs state changes on it, and then outputs the changed quantum data. A simple example is that the following code produces the circuit shown in Figure 5:

```
mycirc :: Qubit -> Qubit -> Circ (Qubit, Qubit)
mycirc a b = do
  a <- hadamard a
  b <- hadamard b
  (a,b) <- controlled_not a b
  return (a,b)
```

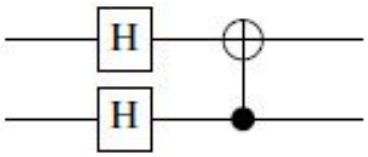

Figure 5: Result of the code sampling in Section 6.4

The *Circ* command can take a number of different types of inputs to signify what to put into the circuit. Here, *Circ* has two inputs of type *Qubit* to make a circuit with two qubits.

6.5 Implementation of Triangle Finding Algorithm

The authors of [4] implemented a different quantum triangle finding algorithm than the one discussed in this paper. This algorithm can be broken down to a number of parts, one of which is the oracle used by the algorithm. What we highlight here is their analysis of the number of qubits and quantum gates needed to implement this algorithm as a whole and specifically, the oracle it uses. Quipper has a simple command that can be used to compute the gate and qubit counts for a function. For the oracle portion of this algorithm, this command counts a total of 2,051,926 gates and 1,462 qubits. For the whole algorithm this produces a count of over 30 trillion total gates and 4,676 qubits.

6.6 Quipper vs QCL

The authors of [4] briefly discuss a C-style language known as *QCL* as arguably the oldest "concrete" quantum programming language. The authors also compare the two languages by implementing identical versions of another quantum algorithm, called the *Binary Welded Tree* algorithm, as well as a hand-coded oracle in each language. Knowledge of the BWT algorithm is not necessary for understanding the final results of this comparison. To compare the languages, the authors tallied up the total number of quantum gates and qubits used by each implementation. The results of this comparison were that the QCL version of the algorithm produced a total of 17,358 gates and used 58 qubits total in the circuit. The respective numbers for the Quipper verision were 1,300 gates and 26 qubits.

7. CONCLUSIONS

In conclusion, while there are still many questions left in the field of quantum computing, some of the more difficult problems are being answered in a meaningful way. There may still be a long wait in the development of practical implementations for the hardware of a quantum device, but recent developments in other facets of quantum computing are allowing the field to be more prepared for the day that useful quantum devices come into existence. There already is a large base of quantum algorithms that allow complex problems to be solved faster than they could be on a classical system. New developments in quantum programming languages allows more discussion on how these algorithms can be implemented by a programmer. It has even allowed analysis on the optimization of some aspects, such as quantum gate count, in these proposed systems. These kinds of developments help ensure that the field will be ready when a proper quantum device arrives in the world.

8. REFERENCES

[1] An introduction to quantum computing for non-physicists. *ACM Comput. Surv.*, 32(3):300–335, Sept. 2000.

[2] S. S. Epp. *Discrete Mathematics with Applications*. Brooks/Cole-Thomson Learning, Belmont, CA, USA, 2004.

[3] C. Gidney. Grover's quantum search algorithm. `http://twistedoakstudios.com/blog/Post2644_grovers-quantum-search-algorithm`, 2013.

[4] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. Quipper: A scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 333–342, New York, NY, USA, 2013. ACM.

[5] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. ACM.

[6] F. Magniez, M. Santha, and M. Szegedy. Quantum algorithms for the triangle problem. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 1109–1117, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.