Using Quantum Computing to Determine the Optimal Path on Cascading Graphs

Kennesaw State University

Ethan Hunt

ehunt16@students.kennesaw.edu

Michael Swann

mswann11@students.kennesaw.edu

Contents

Abstract:

Quantum computing has completely changed the computing paradigm. These special computers leverage the unique properties of quantum mechanics to solve problems that a classical computer cannot solve in polynomial time. Quantum mechanics such as superposition and entanglement are used to boost computational power exponentially in many problems. Many traditionally NP-complete problems, such as breaking the encryption of public-private key systems, are solvable with quantum computing in polynomial time. In this project, we will review quantum computing basics using real quantum computers and build on those basics to solve a subset of a graph optimization problems using both existing and new methodologies. Our research focuses on a subset of graphs named "*Cascading Graphs*" and finding the

"*best*" path based on a predetermined metric. To solve this problem, we plan to use a mixed approach for finding a mathematical algorithm and creating an implementation of the algorithm in a quantum computer. This mixed approach will be based on a cycle consisting of trying a find a mathematically rigorous proof and testing different cases to help build an understanding of the problem, which will then be verified using a quantitative approach.

Introduction:

A full graph traversal on a traditional computer can have up to exponential time complexity depending on the graph and implementation. On the other hand, quantum computers can make the same graph traversal orders of magnitude faster by computing multiple possibilities in a single quantum calculation.

Important Concepts and Notations:

- ***Cascading Graph/Tree***: A directed acyclic graph *G(V,E)* in which every vertex can have edges only connecting to a vertex exactly one vertex depth increase from the original vertex such that all paths leading to a specific vertex require the same number of vertices to be traveled. In addition, weights are nonzero positive numbers and each vertex of a depth n from the root shares a common depth weight(*W*). Such that edges going down to the next level from a given vertex must add up to the same value for all vertices at the same depth.

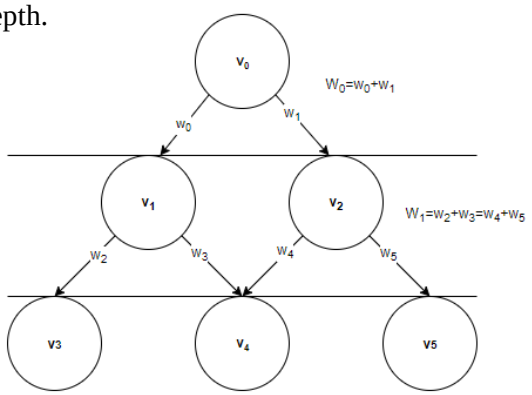


Figure 1: Example Graph

- ***Path vector $\vec{t}$:*** A vector whose entries correspond to the indexes of the vertices traveled. Each path vector has a corresponding cost vector ${}_t\vec{C}$ whose entries correspond to the weights in of the edges connecting each vertex in the path vector
- ***Full Path:*** A path on G is considered full if and only if the path starts at the root vertex and ends at a leaf vertex
- ***Set of path vectors T*:** The set of path vectors of a given length on G are contained in the set *T*
  $\mathrm{T} = \left\{ \overrightarrow{\mathrm{t_k}} \,|\left(\overrightarrow{\mathrm{t_k}} \in \mathrm{N^n}\right), (0 \leq \mathrm{k} > |\mathrm{T}|) \,, (0 < \mathrm{n}) \right\}$
- ***Cost function $Cst(\vec{t})$: A*** function that determines the cost of a given path along G of a given depth. The cost function is unique to each graph and defined explicitly using an optimization matrix

- ***Probability of a given Path $P(\vec{v})$:*** The probability of a path occurring on a cascading graph is $P(\vec{t}) \stackrel{\text{def}}{=} \prod_{i=0}^{n-1} \left(\frac{t c_i}{W_i}\right)$
- ***Probabilistic mapping of* G *to* G'**: A probabilistic mapping of G to G' is defined as when all the weights of edges in G are mapped on to an identical map G' where the weights of G' are defined as follows. The weight *w* of an edge E (w_e) is represented as the fractional probability of w_e over sum of the edges on a given initial vertex to all adjacent vertices $\frac{w_e}{\sum_i (w_i)}$.

Reasons for Quantum:

The main reason that quantum computers have the potential to be faster at solving these problems than traditional computers is because of their use of superposition in two unique ways. The first is that this approach cannot be perfectly implemented on a classical computer as this involves randomly selecting a given edge. This can easily be seen as an issue as truly random outcomes are not possible on a classical computer. However, that is not the case for quantum computers. By taking advantage of the true randomness of supposition we can fully realize true random walks on a Markov chain. The second is that superposition allows the computer to entertain all possible sates at once. Instead of simply starting with a given state and taking a selected number of walks and analyzing the most frequent of the walks, we can find the optimal solution from any start sate by using superposition. Simultaneously checking all outcomes at once allows us to find a perfect solution similarly to the brute-force approach on a classical computer without the inefficiency [1]. While the ability to simultaneously calculate all paths is a plus, the biggest advantage of quantum is the potential for quantum exclusive information, which allows us to decrease the number of possible combinations of walks required to shift though.

Process and Methodologies:

The approach taken to researching and developing an algorithm was mixed approach that uses both qualitative and quantitative assessments and evaluating them together.

1. Create an attempt at a solution
2. Test attempt at solution
3. If all tests seem sufficient to explain the mechanics of the problem
    - **A.** Try to prove solution,
        - I. If proved: Go to step **4**
        - II. Else: revaluate potential solution and go to step **1**
4. Record your findings

Results/Findings:

The research in this paper can be split up into two categories: pure and applied findings. The pure findings deal with more abstract and theoretical implications, while the applied findings deal with the applications of the pure findings in actual quantum computers and the ways to improve the implementation of the pure findings. We used proof methods and structures taken from "Discrete Mathematics with Applications 3rd Edition" [2] in formulating various proofs

Pure Findings:

The pure findings section deals with mostly in the pure math and algorithmic side of the research. This section goes over the concepts of searching on Cascading graphs outside of a quantum context. This was used to verify the conceptual validity of our algorithm.

Cascading Graphs:

As defined above a *Cascading Graph* is a DAG in which moving along a given path always results in moving with increasing depth from the root or start vertex. In addition, all vertices of equal depth must have the sum of their edge weights be equal to given constant for that level.

Cascading Graphs are named after the cascading motion traveling along a given path creates as they have a constant downward motion akin to a waterfall. This analogy will become more apparent in the context of a cascading search.

Additional Properties:

Lemma 1. All subgraphs of a Cascading Graph are also DAGs
Proof by Contradition

A. Assume that a subgraph g of a Cascading Graph G is not a DAG
B. Since g is not DAG it lacks at least one of the properties of a DAG
C. Since g is contained within G there exist a section of G that lacks at least one of the properties of a DAG
D. Therefore G is not a Cascading Graph as all Cascading Graphs are DAGs
E. Because statement $A.$ is true, G is a Cascading Graph
F. Therefore statement $A.$ and $D.$ are both true which is a Contradiction
G. Therefore all subgraphs of a Cascading Graph are also DAGs ■

Cascading Graph Search:

The reason for choosing a cascading graph over other types of graphs is its unform total cost at each depth level. This allows for additional information to be obtained when comparing walks of a certain depth on the graph. For example, assume that each depth has Weight(W)=1 for all depth levels on the graph, which means all values of the edge weights are on the interval (0,1) (aka a mini-Markov chain for each walk of depth one). This means we can represent weights as the probability of taking one path over another. This is coupled with the fact that each probability is independent of the path taken prior. Not only can these probabilities be mapped to qubits, but they can also use another trick to fully take advantage of this unique effect. Assume there is a cascading graph *G* that we map to a cascading graph *G'*, which is a probabilistic mapping of *G*. For this to work, we need to normalize the levels of our graph. We accomplish this by changing each weight in a level to the weight divided by the W of the depth. This maintains the first 3 axioms of Probability Theory as defined in the "Introduction to probability, statistics, and random processes" [3]. The main benefit of using this property of a Cascading Graph is that we can distinguish between different critical edges. If we do not uses these level weights, we end up with local maximums that cannot be distinguished from one another as their probabilities simplify getting rid of the value of the weight in the denominator. There by erasing the original values of the weight of the edges involved making it impossible to determine the best option without looking back at G.

To show what happens when you do NOT take these local maximums into account, we will use a graph that does not have a set level weight and has two vertices at the same depth each with two edges where the weights of the first vertex are twice that of the second. When applying the incorrect probabilistic mapping of G to G', one such possible graph is show in the following example.

Figure 3: Example of G

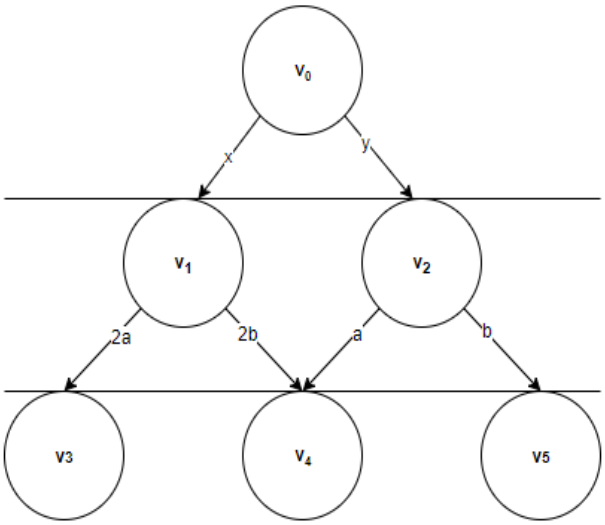
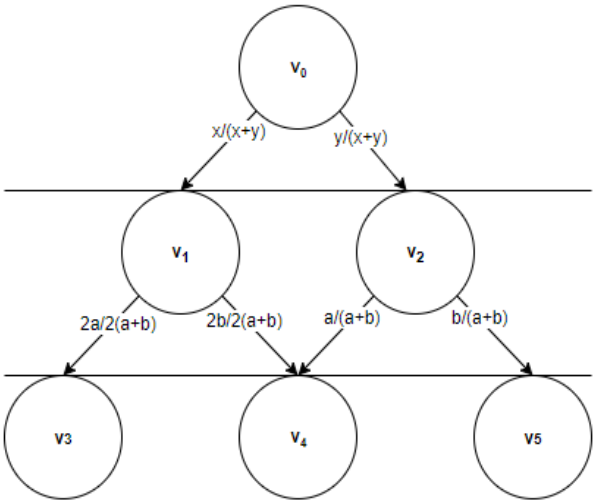


Figure 2: Example of G'



As seen in the example graph, you could simplify the left subtree to make the weights of the edges identical to the values on the right sub tree, yet looking at G, we know that their values are different. While this may not seem important at first, it prevents us from utilizing the major benefit of this method, which is that, for the majority of graphs G, the probability of the best path will often have the greatest value. The reason why it is *often* the case and not *always* the case has to do with the relationship between a product and the sum of the elements of a given vector. However, if our question involved looking for the path that had the largest product, it will always be the case, since the probability is proportional to the cost product. So, graphs where we're finding highest product path can take full advantage of this approach. If a cascading graph G has complete path(s) that are of different lengths, one of two things must happen before determining our results. We can circumvent the issue of having a path of lower product and depth rather than having a higher frequency for a path of a higher product and depth. One approach is called "*Completing the Graph"* which simply involves adding additional vertices to a graph in such a way as to lower the frequency of taking the incorrect depth path. The other solution is simpler, as all that is required is after the completion a random walk if the path vector is of a length that is not desired you simply don't add it to the result pool. This works because the frequency of a given path of a different length won't be affected if paths of other lengths are discarded. Thus, we simply pick the highest of a certain depth in our result pool.

Completing the Graph

Completing the graph could be thought of as another way of normalizing the original graph as we are augmenting the graph in such a way as to not change the properties of the graph. In this case we are trying to make frequency of paths with a shorter length decrease in frequency in comparison to higher depths. While at first this problem may seem unsolvable or quite complex, a rather simple solution exists. Since we need our vertices to maintain their original weight values, we can simply add 2 or vertices each having an edge of equal weight connected via last node in the path we are appending. Simply repeat this process for all paths that are less than the longest possible path. This makes it very unlikely that any of those paths will appear the most frequently in each sample. This is because every time a random walk would

normally result in a path with an incorrect depth, its occurrences are now divided up. Thus, limiting the frequency of any one incorrect path.

Cascading Search for Optimal Path of a Given Length

For given graph *G* mapped to *G'* and is performed *k* using walks

Steps:

1. Preform *k* random walks of desired length on *G'* and record the results
2. Using an either a classical search or Grover's Algorithm find the most common vector of the sample set

Cascading Search for Optimal Path for any Length

For given graph *G* mapped to *G'* with *m* different lengths for possible full paths and is performed *k* times walks

Steps:

1. For each length of a possible full path
    a) Preform *k* random walks of given length on *G'* and record the results
    b) Using an either a classical search or Grover's Algorithm find the most common vector of the sample set
2. Using G calculate the cost for each of the most common path vectors found in **1.b)**
3. Using an either a classical search or Grover's Algorithm find the optimal cost vector of the possible most common path vectors

Proof of Cascading Search for finding path with highest product

For a tree partition of Cascading Graph of a depth n with weights $w \geq 0$

$$G(V,E,n) \ \wedge T \wedge \left(P(\vec{t}) \stackrel{\text{def}}{=} \prod_{i=0}^{n-1}\left(\frac{{}_t\boldsymbol{c}_i}{W_i}\right)\right) \wedge \left(Cst(\vec{t}) \stackrel{\text{def}}{=} \prod_{i=0}^{n-1} {}_t\boldsymbol{c}_i\right)$$

$$Prove\ (\forall\, \vec{t}_i \in T)\left(\left(P(\vec{t}_{max}) \geq P(\vec{t}_i)\right) \longrightarrow \left(Cst(\vec{t}_{max}) \geq Cst(\vec{t}_i)\right)\right)$$

Direct Poof

A. $(\forall\, \vec{t}_i \in T)\left(P(\vec{t}_{max}) \geq P(\vec{t}_i)\right)$

B. $Assume\ \boldsymbol{A}.\ is\ true$

C. $\because \left(\left(Cst(\vec{t}) \stackrel{\text{def}}{=} \prod_{i=0}^{n-1} {}_t\boldsymbol{c}_i\right) \wedge \left(P(\vec{t}) \stackrel{\text{def}}{=} \prod_{i=0}^{n-1}\left(\frac{{}_t\boldsymbol{c}_i}{W_i}\right)\right)\right) \vdash \left(Cst(\vec{t}) = P(\vec{t})\left(\prod_{i=0}^{n-1} W_i\right)\right)$

D. $\therefore \left(Cst(\vec{t}) = \left(\prod_{i=0}^{n-1}\left(\frac{{}_t\boldsymbol{c}_i}{W_i}\right)\right)\left(\prod_{i=0}^{n-1} W_i\right)\right)$

E. $\because (\boldsymbol{C}.) \vdash \left(Cst(\vec{t}) \propto P(\vec{t})\right)$

F. $\because (\boldsymbol{A}. \wedge \boldsymbol{E}.) \vdash \left(Cst(\vec{t}_{max}) \geq Cst(\vec{t}_i)\right)$

G. $\therefore \left((\forall\, \vec{t}_i \in T)\left(\left(P(\vec{t}_{max}) \geq P(\vec{t}_i)\right) \longrightarrow \left(Cst(\vec{t}_{max}) \geq Cst(\vec{t}_i)\right)\right)\right)$ ■

Big O analysis of a Cascading Search Algorithm Single Level

Time Complexity of Single Walks:
A random walk on a cascading graph takes exactly N steps to complete where D is the depth we are searching to. We then multiply this with the amount steps the largest subcircuit in the algorithm takes to execute. This is because the quantum circuit preforms a constant number of operations per transition between vertices in node as that is predefined by the graph and only will increase if the subcircuit must be changed as the number of output qubits increases with an increase in the number of out edges the corresponding vertex has. Thus, the algorithm will not get increase in time complexity of a sub circuit if it simply adds no new edges between existing vertices. Thus, that is simply the amount of time it takes to simply compute the sub circuit with the highest degree of edges. Therefore, for we have an time complexity $O(D * N)$ for each walk

Time Complexity for a "Survey"
To help increase the accuracy of our answer we perform multiple random walks and compare the outputs. A passible number of shots is a number which gives the correct answer at least 50% of the time. One value that appears to give these results is the following value $\prod_i W_i \equiv S$. Where s is the size of tan adequate sample space. The reasoning behind this value is that denominators of all the probabilities of the different path vectors is equal to $\prod_i W_i$ .These "surveys" are like mini sample sets in which we are only interested in the most frequent value in the "survey". Thus, the time complexity for one of these surveys is $O(S * D * N)$

Time Complexity for a full algorithm:
To calculate the number the time complexity of the complete algorithm we need only count the number of surveys preformed. A passible number of surveys is one that yields the correct answer 100% of the time. In our test a good one value for the number of surveys is in the following range $[|V|, |V|^2]$. Thus, we have a time complexity of $O(|V|^2 * S * D * N)$.

The Case for parallel
If we have substitute parallel circuits such that each circuit preforms a survey and therefore splits up time required to complete the whole collection of surveys, we have a time complexity $O\left(\frac{|V|^2 * N * D * S}{P}\right)$

Other Cases:

In the case of finding the shortest or longest path using this cascading search approach instead of finding the highest or lowest product path, we must address the issue of cost calculation of the path walking on probabilistic vs a standard weighted graph. There may initially seem to be no big issue, since if you try a few cases, they seen to yield results of the paths as expected. However, there exist a range of values such that even if the sum of the cost a path vector's entries is one greater than the second path vector, the second will have the higher probability despite having a smaller total sum. This is because, when you compare the product of a vectors when the comparison of their sums is known, there exists this same exact rage of [a+1, b-2] where a is the smallest and b is the largest cost element in the better path vectors entries. This means that, for the second path vector to have a greater product and thus probability as the two measurements are proportional as shown in the proof of product search, the path vector would need to have entries that sum up to exactly one less than the first on the rage [a+1, b-2] where b>2a. If for all the paths this is not the case, then a search will yield correct results. For small difference in products, the difference in probabilities is proportionally small, so as the limit of the path size approaches infinity, the

differences will become zero. This is because as the path gets bigger and bigger, this small change in its cost gets less and less important as the cost increases. This does not help our implementation though, since the graphs we will use on a quantum computer are not infinite.

Note: Given a theoretical constructed PMF of paths on G, we can then simply compare the frequency values and have a 100% that the path with the greatest probability is the path we desire. This is always the case of looking for the path with the highest product.

Applied:

This section deals with the application of the Pure findings in the context of a quantum computer

Transferring a graph to a quantum circuit

The following example of a walk of depth one is akin to the greed-best first approach. This approach is widely known not to given optimal solutions for a large number of problems, but is used when a acceptable alternative is likely to show up as well. The greedy approach cannot solve certain problems that require looking ahead. While individually it is not used to solve a problem, it is often used in conjunction with other methods to solve more complicated problems.

While seemingly disconnected form quantum algorithms, you can frame the question in a way that maps the greedy approach onto a graph made of qubits. To do this, we create an analog of our graph in the form of a Markov chain. This Markov chain will have edge weights that are probabilities that correspond back to our original graph.

To create an analogy between our graph and our new Markov chain, we must do the following calculation for each possible walks of depth one.

For given tree that represents a walk of depth one starting at a given state:

1. Add all the weights on the tree to get a total weight for the tree
2. For each edge on the tree, take its weight and divide it by the total weight to get is probability weight for the corresponding Markov tree

The reason this works as a mapping is that, for a sate on the Markov Chain, the probabilities of previous states do not impact the probabilities of the current state. This means that the probabilities of the possible sate changes must add up to one. This means that our weights must also maintain this property mapped. By taking only the total weight of the sub chain and dividing the weight by it we meet the conditions.

With the mapping of the Markov chain complete we can proceed to mapping it onto our quantum circuit.

Main Conjecture: There exist a mapping of a complete directed graph G(V,E) on to a quantum circuit such that the vertices of G become the qubits on the circuit and where there exist an edge on G connecting two vertices mapped to qubits $|A\rangle$ and $|B\rangle$ then the $P(B|A\rangle) > 0$. This means it is possible to jump from state ‘A’ to ‘B’ implying the existence of an edge between them. The weight *w* of an edge E (w_e) is represented as the factional probability of w_e over sum of the edges on a given initial vertex to all adjacent vertices $\frac{w_e}{\sum_i(w_i)}$. Thus, by taking a random walk on any given edge from an initial vertex to a destination vertex the edge with highest weight will be chosen with highest frequency. Thus, by taking repeated walks on edges and selecting the paths that result in a complete spanning tree of the graph, the

path of highest product of weights will appear the most frequently. By setting our edges to values that can be represented as higher values being better, we can solve a variety of problems as well as if the data set is normalized in such way it can solve other problems with high degrees of accuracy.

The Need for Mutually Exclusive Quantum gates

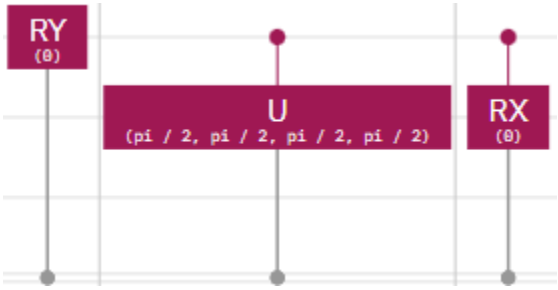
In order to adequately assess whether quantum computing can give the potential benefits outlined, we need way to keep track of our path and be able to distinguish between different walks on our graph.

This can be done by allowing a transition of an initial state to a destination state to be uniquely mapped to a given qubit's value being true after completion of the gate. This value can then be stored in a register corresponding to the depth on the walk. By saving the initial state in its own register and saving the destination state in its own register for each transition you can make the unique walks taken by each shot on the circuit. This can then be used to analyze our results.

However, this is resource heavy as we need as many qubits as vertices on the graph to represent a case where a state can transition to all possible destinations including itself. So this algorithm is best used on subsets of graphs where this is not the case and listed to a smaller subset of possible destinations for each vertex.

To make a proof of concept for a graph of three vertices and 9 edges, we decided to look for gates that could do this. We initially came up empty handed, but we eventually found a gate that did something not quite what we wanted: the *W* gate found in the "Qiskit textbook" [1]. This is where we played around with some combinations of the *W* gate until we found one that allowed us to have different probabilities of complaint to one for each qubit. This however has an issue, since the main part of the *Modified W* gate is difficult to map to a specific probability represented through three unique rotations around different axes on the block sphere

These can be represented using the following gates



It was more difficult to solve for the constants used to define our edges as we have 2 rotations in the y axis and one rotation in the axis. This means solving for 3 rotations and having to keep track of how each the different arises affect one another in for given values.

We decided to abandon looking further into the *Modified W* gate in hopes of finding a new gate that we can discreetly model using only two rotations on the y axis.

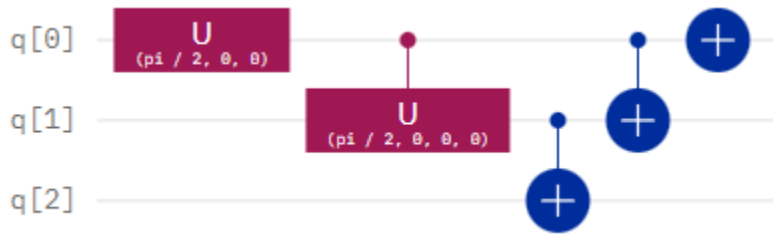
The ME3 Gate

Upon further looking into gate creation using the "Qiskits textbook" [1] as a reference point on notation and as a glossary of know gates, the concept of the *ME3* gate was created. The *ME3* works as follows. the first qubit is rotated by a set amount so that the probability of the first qubit is matched up with the probability specified on our Markov chain. The controlled *U* gate acts similarly to how a *cnot* gate entangles to qubits when the first is in superposition the. Just like with the first *U* gate this, gate acts on rotating the qubit by a set amount around the y axis to yield a desired probability. The rotation acts as the component which splits the remaining probabilities between the two qubits. By applying *cnot* gates and *not* gates to the qubits in reverse order, you change the probabilities of to be exclusive of one another. This is what allows the ME3 gate to choose its distribution of probabilities on its qubits using two constants.

This can be geometrically represented using two circles and three events (A,B,C). The total possible set of probabilities between two number can represented by the sum of magnitudes for all pairs of (x,y) on the perimeter of the unit circle. Since the unit circle has an radius of 1 the for all the ordered pairs of amplitudes (x,y), their sum of the magnitudes is equal to one squared, which is one. Thus, we can assign the following to each applied $\sqrt{P(A)} = x$ and $\sqrt{1 - P(A)} = y$. With the fact that the events are mutually exclusive, their probabilities are not independent, since it depends on if the other event doesn't happen, thus $P((B \vee C)|A) = 0$ so $P(B) + P(C) = \sqrt{1 - P(A)} = y$. Then, to represent the probability of the events B and C, we can view them as the proportional to the amplitudes of the ordered pairs (i,j) which lies on a circle with a radius of y.

Since both the controlled *U* gate and the *U* gate are specifically defined in [1], you can see how applying a given rotation effects the qubits. By taking the effect of the *U* gate on the whole system, you can solve for the given angles for a desired probability. This allows use to accurately model our desired weights on the quantum circuit.

***ME3* Gate:**



The Mutually exclusive 3-bit qubit gate is comprised of 2 *U* gates, 2 *CX* gates, and one *X* gate.

The benefit of using the *ME3* gate over the modified *W* gate is that the modified *W* gate lacks a simple way to find the exact values for the angle rotations. However, the *ME3* gate does and using linear algebra, geometric series, the fundamental theory of probability, trigonometry, and some algebraic manipulation, an equation for the constants can be derived.

Let the ordering of tensor product of n, n dimensional vectors for n equals 2 or 3 be $\begin{pmatrix}00\\10\\01\\11\end{pmatrix}, \begin{pmatrix}000\\010\\100\\110\\001\\011\\101\\111\end{pmatrix}$

respectively

Given an initial state of $|000\rangle$, the *ME3* gate has a state vector of

$$|S\rangle = cos\left(\frac{\theta}{2}\right)|001\rangle + cos\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)|010\rangle + sin\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)|100\rangle$$

Derivation of state vector:

All calculations were checked both by hand and using a matrix multiplication calculator [4]

This is the conversion of the circuit into matrix form as a set of linear transformations on a vector

$$\begin{vmatrix}1&0&0&0\\0&1&0&0\\0&0&cos\left(\frac{\vartheta}{2}\right)&-sin\left(\frac{\vartheta}{2}\right)\\0&0&sin\left(\frac{\vartheta}{2}\right)&cos\left(\frac{\vartheta}{2}\right)\end{vmatrix}\begin{vmatrix}cos\left(\frac{\theta}{2}\right)&0&-sin\left(\frac{\theta}{2}\right)&0\\0&cos\left(\frac{\theta}{2}\right)&0&-sin\left(\frac{\theta}{2}\right)\\sin\left(\frac{\theta}{2}\right)&0&cos\left(\frac{\theta}{2}\right)&0\\0&sin\left(\frac{\theta}{2}\right)&0&cos\left(\frac{\theta}{2}\right)\end{vmatrix}\begin{pmatrix}1\\0\\0\\0\end{pmatrix} = \begin{pmatrix}cos\left(\frac{\theta}{2}\right)\\0\\cos\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)\\sin\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)\end{pmatrix}$$

$$\begin{vmatrix}1&0&0&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&1&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&0&0&1\\0&0&0&0&0&0&1&0\\0&0&0&0&0&1&0&0\end{vmatrix}\begin{pmatrix}cos\left(\frac{\theta}{2}\right)\\0\\0\\0\\cos\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)\\sin\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)\\0\\0\end{pmatrix} = \begin{pmatrix}cos\left(\frac{\theta}{2}\right)\\0\\0\\0\\cos\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)\\0\\0\\sin\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)\end{pmatrix}$$

$$\begin{vmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&0&0&1\\0&0&0&0&0&0&1&0\end{vmatrix}\begin{pmatrix}cos\left(\frac{\theta}{2}\right)\\0\\0\\0\\cos\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)\\0\\0\\sin\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)\end{pmatrix} = \begin{pmatrix}cos\left(\frac{\theta}{2}\right)\\0\\0\\0\\0\\cos\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)\\sin\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)\\0\end{pmatrix}$$

$$\begin{vmatrix}0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&1\\1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\end{vmatrix}\begin{pmatrix}cos\left(\frac{\theta}{2}\right)\\0\\0\\0\\cos\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)\\sin\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)\\0\end{pmatrix} = \begin{pmatrix}0\\cos\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)\\sin\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)\\0\\cos\left(\frac{\theta}{2}\right)\\0\\0\\0\end{pmatrix}$$

$$|S\rangle = cos\left(\frac{\theta}{2}\right)|001\rangle + cos\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)|010\rangle + sin\left(\frac{\vartheta}{2}\right)sin\left(\frac{\theta}{2}\right)|100\rangle$$

Solving for the Angles of Rotation

Solving for first Angle of rotation:

Steps.

1. $P(|001\rangle||S\rangle) = cos\left(\frac{\boldsymbol{\theta}}{2}\right)^2 = \frac{w_0}{\sum w_i}$
2. $\sqrt{\frac{w_0}{\sum w_i}} = \cos\left(\frac{\boldsymbol{\theta}}{2}\right)$
3. $\boldsymbol{\theta} = 2\text{ARCcos}\left(\sqrt{\frac{w_0}{\sum w_i}}\right)$

Two main methods for solving the second rotation

Method One:

Steps.

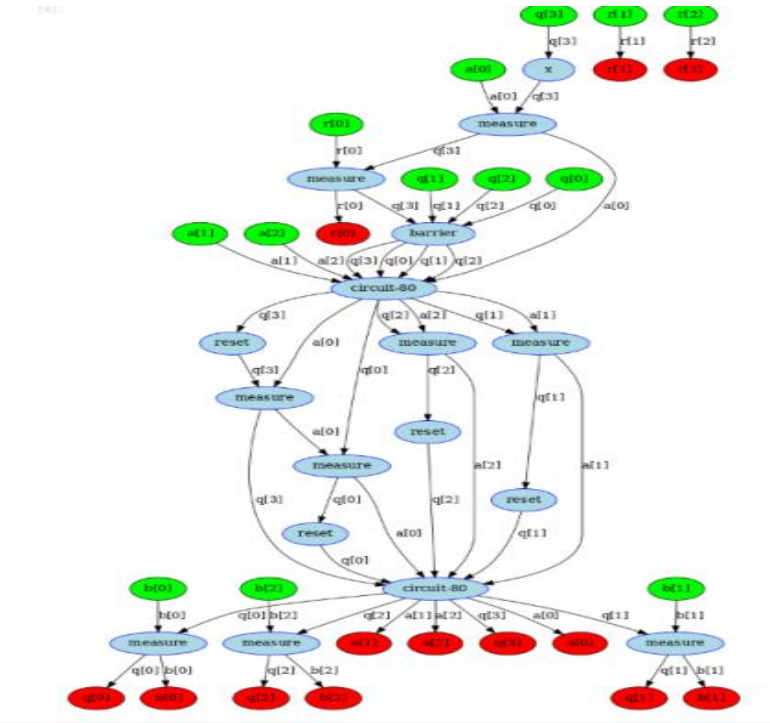
1. $P(|010\rangle||S\rangle) = \frac{w_1}{\sum(w_i)} = \sin\left(\frac{\boldsymbol{\theta}}{2}\right)^2 \cos\left(\frac{\vartheta}{2}\right)^2$
2. $\sqrt{\frac{w_1}{\sum(w_i)}} = \sin\left(\frac{\boldsymbol{\theta}}{2}\right)\cos\left(\frac{\vartheta}{2}\right)$
3. $\vartheta = 2\,\text{ARCcos}\left(\frac{\sqrt{\frac{w_0}{\sum w_i}}}{sin(\frac{\theta}{2})}\right)$

Method Two:

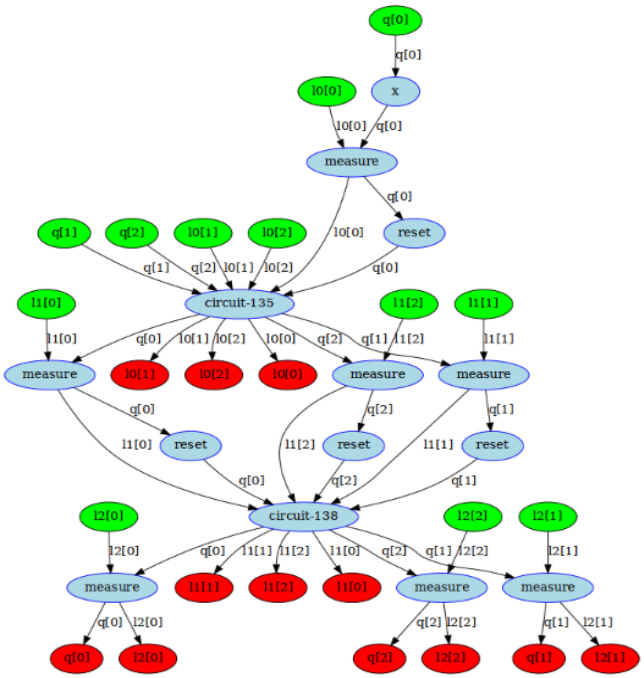
Steps.

1. $P(|010\rangle||S\rangle) = \frac{w_1}{\sum(w_i)-w_0}(1 - \frac{w_0}{\sum w_i}) = \sin\left(\frac{\boldsymbol{\theta}}{2}\right)^2 \cos\left(\frac{\vartheta}{2}\right)^2$
2. $\frac{w_1}{\sum(w_i)-w_0} = \cos\left(\frac{\vartheta}{2}\right)^2$
3. $\sqrt{\frac{w_1}{\sum(w_i)-w_0}} = \cos\left(\frac{\vartheta}{2}\right)$
4. $\vartheta = 2\,\text{ARCcos}\left(\sqrt{\frac{w_1}{\sum(w_i)-w_0}}\right)$

An Example Implementation using the Modified W gate



An Example Implementation using the ME3 gate



The above example is a walk on a graph of depth 2 with the sub circuits being a subcircuit that holds the possible state transitions that can be given on a walk. Due to limitations in the Qiskit language that are not hardware related, Qiskit does not offer control flow in its subcircuits needed for more complex graphs.

Conclusion and Future Work:

In this paper we have proven that for Cascading Search that the optimal product path is proportional to the probability of taking a random walk on G' that results optimal product path. Therefore, we know that cost must be greater than or equal for all path vectors with probabilities less than the optimal due. We then built a proof-of-concept simulation using IBM's Qiskit Lab to perform Cascading Searches on a quantum computer. To do this we need to design special gates that can mutually exclude qubits for collapsing to one. The next part of our research is going to be trying find the time complexity of using the cascading search algorithm on a quantum computer and compare it to that of classical search algorithms on the same graph. In which we assess the estimated number of steps required proportional to the original graph. Future research into the implications of the ability to contract the range without losing information over the domain in the context of transferring non probabilistic data to probabilistic data in the form of qubits. In addition research into expanding and improving of a cascading search algorithm to be able to more adequately a large subset of graphs faster.

Author Note

Listed here are the contributions of each author in the paper using the CRediT Taxonomy [5]

Author: *Ethan Hunt*

Email: *ehunt16@students.kennesaw.edu*

Role(s): *Lead Investigator, Mathematical Analysist, Software Developer*

Contributions: *Conceptualizations, Formal Analysis (including proofs), Methodology, Validation, Investigation, Writing-Draft, Software, Visualization*

Author: *Michael Swan*

Email: *mswann11@students.kennesaw.edu*

Role(s): *Project Manager, Investigator, Software Developer*

Contributions: *Project Administration, Investigation, Validation, Software, Writing-Edit and Review, Data Curation, Resources*

Works Cited

[1] T. Q. Team, "Qiskit Textbook," 2022 July 2022. [Online]. Available: https://qiskit.org/learn/.

[2] S. S. Epp, Discrete Mathematics with Applications, THIRD EDITION, Brooks Cole, 2003, pp. 125-295.

[3] H. Pishro-Nik, "Introduction to probability, statistics, and random processes," 2014. [Online]. Available: https://www.probabilitycourse.com/.

[4] Symbolab, "Symbolab Matrix Multiply, power calculator," Course Hero Symbolab , 2021. [Online]. Available: https://www.symbolab.com/solver/matrix-multiply-calculator.

[5] "CRediT," NISO, 2022. [Online].