Solving the General Planar Graph Coloring Problem Using Grover's Algorithm

Bhuwan Bhandari, Pratyush Bhattarai, Swayam Chaulagain, Ashlesha Dangal, Mukesh Ghimire, Madhav Khanal, Lenish Pandey, Sooraj Sahani, Sampada Wagle

August 10, 2023

Abstract

In this paper we discuss the structure and implementations of the planar graph-coloring problem (PGCP). We briefly look at well-known classical algorithms used to solve the PGCP, but primarily focus on the quantum computational angle. Grover's search is a well-known quantum algorithm that offers a quadratic advantage relative to its classical counterparts. We inspect its application to the PGCP and build a corresponding quantum circuit.

Introduction

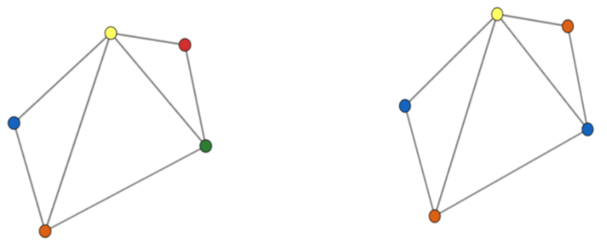
Figure 1

Mathematical and computational methods of problem solving have grown exponentially over the past century, providing efficient and effective solutions to various problems. Mathematical algorithms have been integrated to different fields of human endeavor, such as finance, health, agriculture, education, and so on. The development of problem-oriented computational algorithms dates back to the mid-20th century (Knuth, 1977). Researches including Turing (1936), Dantzig (1951), Hoare (1961), Haigh (1993), Copeland (2004), Belvos (2013), Montanaro (2016), Childs et. al. (2018) have reported compelling algorithms ranging from logic, linear functions, universal computation to optimizations. With advancement of quantum mechanics, quantum computing algorithms have also been intensively used in the fields of optimization, cryptography and cryptoanalysis.

Taking Nepal as a sample location to map out different real-life scenarios in terms of mathematical and computational models for efficient problem solving, a number of areas can be considered. We can map out the location of disaster-prone areas and based on the distance between the major necessities, allocate the appropriate resources. We can effectively plan hydropower plant scheduling, based on the energy consumption of a certain area, the number of workers, total electricity production, as well as the medium of transmission. We can optimize bus routing based on distance and traffic mobilization for deterministic time frames. We can also allocate network bandwidth in a way that it meets transmission requirements with least interference and maximize network efficiency.

Aforementioned issues, based on problem domains and requirements, fall under resource allocation and effective scheduling. For a general resource allocation problem, a set of resources is to be allocated to different specifications. Different constraints are then defined which adds objectivity to the problem. Additional constraints can be defined to maximize the output function based on the problems. Likewise, as a scheduling problem goes, a set of works to be scheduled for a given time based on work-specific and time-specific parameters can be determined.

For the purpose of our paper, the problem we shall be focusing on is to allocate specialist doctors to different hospitals in the Kathmandu valley such that no doctor of similar expertise lands on the same or even adjacent hospitals at a given period of time. The hospitals have been chosen based on accessibility, proximity, clinical metrics and resources. This sort of problem with adjacency in terms of resource allocation and scheduling can be modeled using graph coloring.

Various classical algorithms have been developed to address the graph coloring problem. However, we have opted for the use of the quantum algorithm "Grover's algorithm" to solve the problem. After mapping the information of hospitals and specialists in the graph coloring problem, we apply Grover's algorithm which, through repeated iterations, gives us the most effective solution. The sections to follow cover the details of mapping a graph coloring problem using different classical algorithms as well as implementing Grover's algorithm.

Problem Introduction

Leading Nepali news portals have often reported problems in health services due to unavailability of special doctors in rural Nepal. A fairly recent account of the threatened future of Nepal's health sector due to lack of superspecialist doctors has been reported (OnlineKhabar English News, 2023). According to Nepal Medical Council (NMC), there are 10,080 specialist doctors as of January, 2023. The specialist doctors have not been mobilized properly with all the specialists being concentrated in bigger cities, and due to the very limited number of specialists, it is very problematic for everyone to get access to effective health services. Thus, we tried to generate an algorithm which would allocate the specialists to different hospitals based on pre-set constraints, which in our case is proximity.

For our solution sampling, a total of 8 hospitals were taken, and based on clinical facilities available and the distance, edges were defined. The hospitals under observation are Norvic International Hospital, Civil Hospital, B&B Hospital, Nepal Mediciti Hospital, Megha Hospital, Patan Hospital, Sumeru City Hospital and Star Hospital. Based on the size and distance between the hospitals, we generated a map which would encompass the necessary information of the hospitals (vertices), and the edges. For further simplicity, we take the following graph assigning a numerical vertex to the hospitals.

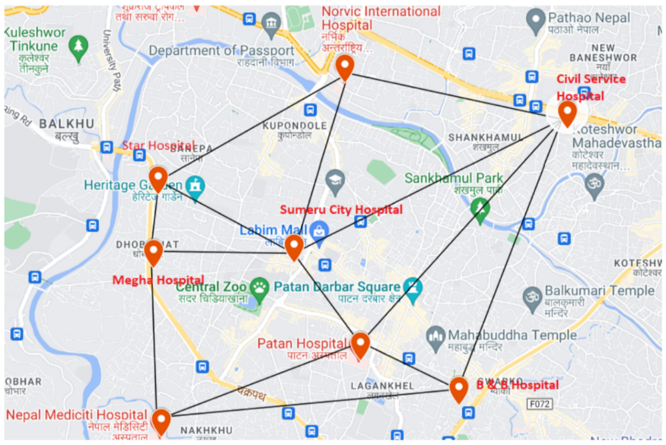

Figure 2

Based on the above graph, we can know the adjacent nodes.

Let E = {(0,1), (0,5), (0,6), (0,7), (1,2), (1,6), (2,3), (2,6), (3,4), (3,6), (4,5), (4,7), (5,7), (6,7)} be the set of all the edges in the graph. To solve the graph coloring problem, any two vertices associated with an edge .i.e. two adjacent vertices should not have the same color.

In our case, the color represents the specialist doctors.

Graph Coloring Problem

A graph is a collection of vertices(nodes) connected by the edges. Typically, vertices of graphs are represented by names or properties. Edge is often used to link any two vertices of the graph. In terms of symbols we represent graphs as G, vertices as V and edges as E. The vertices having an edge between them are often called adjacent vertices. Graphs are either directed or undirected. Edges of directed graphs have direction associated with them while the edges of undirected graphs don't have any direction. All the graphs discussed in this paper will be undirected graphs. Undirected graphs in our paper are simple graphs, that is there won't be more than one edge connecting the same pair of vertices.

Graph coloring, just like its name, is a way of coloring vertices of the graph such that no two adjacent vertices share the same color. For this kind of coloring the easiest hack is using the different colors for each node. Since the total number of nodes in a graph doesn't have any restrictions, using different colors for different nodes is not feasible. Thus, for proper coloring of a graph, one should color the graph using a minimum number of colors. You can notice how we can reduce the number of colors in Fig 1.

The lowest number of colors required to color a graph(G) is called the chromatic number of G, written as (G). A graph G with chromatic number (G)=k is k-chromatic. The graph G whose vertices can be colored using k colors is called k-colorable. Normally there are three types of graph coloring: vertex coloring, edge coloring, and face coloring. In this paper, we will stick with vertex coloring as all other coloring problems can be converted into vertex coloring problems.

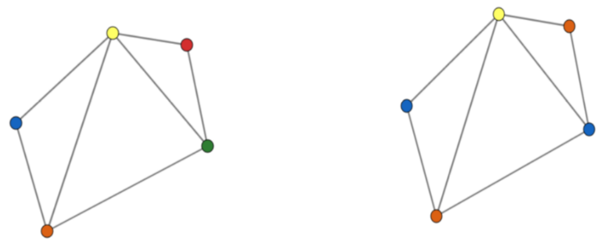

Figure 3

Discussing further about the chromatic number of a graph, we will discuss one of the landmark achievements in the field of graph theory widely known as Four coloring Theorem. The four coloring theorem implies that for any planar graphs their chromatic number is at most four. In other words, we can always color a planar graph with 4 colors. This interesting conjecture was first conjectured by Francis Guthrie in 1852 and remained unsolved for more than a century. Finally the major proof was given in 1977 by Appel and Haken (K. Appel W. Haken, 1977). Their proof was largely computer based as it required solving too many cases. Whether this kind of computerized proof actually constituted proof in the mathematical community is still controversial.

Note that this theorem is limited to planar graphs. Because of this interesting boundary on chromatic numbers for planar graphs, we will be dealing with planar graphs in our paper. So let's look at what exactly are planar graphs..

Basically, a planar graph is a graph which can be drawn in the plane such that no two edges cross except at a vertex. But we can't ensure if a graph is planar just by looking at it. You can see Fig 2 as an illustration.

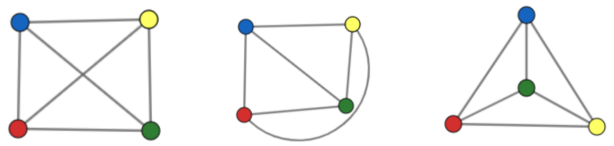

Figure 4

To overcome this problem, Euler formulated a famous theorem known as Euler's theorem. The theorem states that for any planar graph, No. of Vertices(v) - Number of Edges(e) + regions(r) equals 2, i.e. v-e+r=2. The number 2 in this theorem is not actually random as you can notice 2 usually has something to do with the plane. In this theorem, the region(r) of a planar graph is basically sections of a flat surface separated by a planar graph. You can look at figure 3 for its illustration. Imagine erasing vertices from the surface, it breaks into separate pieces, and each piece is called a region. We also need to be aware that there's always one special outer region that contains all the parts of the surface that go on forever. And for a region, degree (deg) is the number of edges that are adjacent to the region, written as deg(R). As an example of Euler's theorem you can see it works for the graph below. Since, v=4, e=6 and r=4, 4-6+4=2.

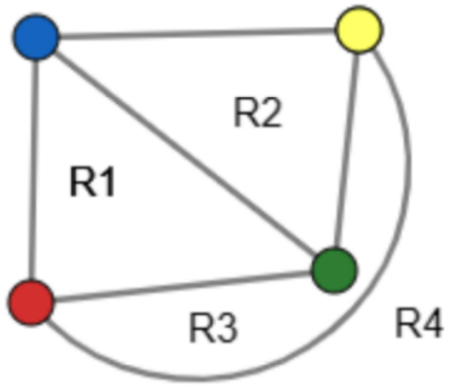

Figure 5

Euler's theorem can be proved using simple mathematical induction (Berman Williams, 2009). Euler's theorem itself doesn't help us much to see if the graph is planar since we need to redraw the graph, but some corollary of this theorem can be very useful for us to test the planarity of a graph.

If G is a planar graph with vertices (v) and edges (e), with $v \geq 3$, then it must satisfy the inequality $e \leq 3v - 6$.

Proof: We can notice that, for each region of a planar graph, its degree is 3, and each edge is adjacent to two regions. Now, we can derive that $2e = \sum$ degrees of r regions of a graph. Thus, $2e = 3r$ and $r = \frac{2e}{3}$. Substituting this in Euler's theorem, we can prove the inequality. However, this theorem is not two-sided so we need to be careful. For all the planar graphs this inequality must be satisfied, but a graph satisfying this inequality doesn't imply that it is planar. For example: In figure 4 you can see that v=5 and e=10 . So, 10 is not 9, the graph is not planar. Whereas in figure 5, v=6 and e=9 and 9 9. But if we check, the graph is not planar.

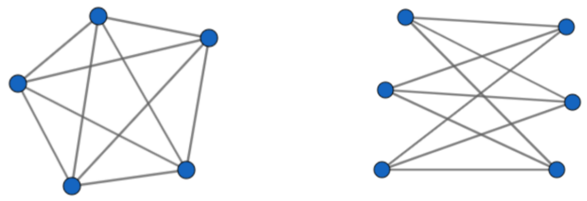

Figure 6

Why study Graph Coloring Problem?

Graph coloring problem is one of the most important aspects in graph theory because of its real-life implications like scheduling, resource allocation, assigning radio frequency, map coloring and many others. However, till the date there is no efficient algorithm for solving the graph coloring problem. It is one of the well known Np-complete problems. Np complete problems are the problems which don't have any established polynomial time algorithm. Polynomial-time algorithms are considered to be efficient because the execution times do not grow rapidly as the problem size increases unlike exponential time algorithms. Thus finding a polynomial time algorithm for any of the np complete problems can solve all of them. .

Classical Solutions

One of the popular classical algorithms for solving graph coloring problems is the Backtracking

Algorithm. Backtracking is a classical approach to solving graph colouring using recursion.It is proven better than other classical and brute-force methods, because the paths leading to false solutions are terminated earlier, preventing any further branches on that path. Although the upper bound of time complexity for both the backtracking and brute-force methods are of O(m^n).

Here, m is the number of colors and n is the number of nodes,the average time complexity of backtracking is lesser .Here is how backtracking works:

We want the adjacent nodes to be of different colors.First we make a list of nodes and a list of colors.Now we put the first color on the first node and move on to the adjacent node.Since the second node can't have the same color as the first,the algorithm looks for the next color on the color list and goes on till all the vertices are colored as required.When the algorithm reaches a node and can't find any color that works,the algorithm backtracks and colors the previous node with a different color.All the wrong branches are destroyed on their roots, and the number of iterations required is lesser than the brute force approach.

One simple example of coloring the vertices of a square (using blue and green color):

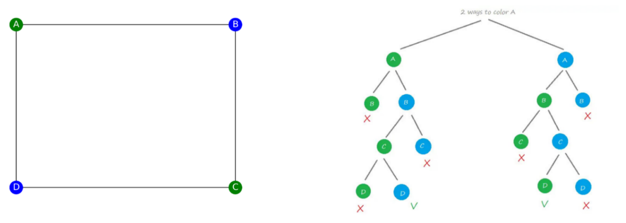

Figure 7

Grover's Algorithm

Say we have a list of unsorted data consisting of N elements and we want to find some desired elements from that list. Mathematically, we can define this problem as if we have a function f such that, f(x)=1 if x is marked (desired solution) f(x)=0 Otherwise

A classical algorithm will, on average, check the list for N/2 times to find the desired element. In 1996, Lov Grover proposed Grover's Algorithm that could find the desired elements in O N evolutions (Grover, 1996). Grover's algorithm uses quantum mechanical properties: superposition and interference and provides a quadratic speedup for searching unstructured databases (Saha et al., 2015).

There are three parts of Grover's algorithm: Initialization, Oracle, and Diffuser. Firstly all the qubits, which are used to translate the problem we want to solve using Gorver's algorithm, are applied to the Hadamard gate which creates a superposition of all the possible states with equal amplitude. This step is called Initialization.

$$|\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \tag{1}$$

In the second step, the oracle function is applied to these states in superposition (Ket s). The oracle Uw is designed such that it can mark the desired item(s) in the database by flipping the phase of that corresponding desired item. While it flips the desired item, the other states remain unchanged.

$$U_f |x\rangle = \begin{cases} -|x\rangle & \text{if } x \text{ is a solution state,} \\ |x\rangle & \text{otherwise.} \end{cases} \tag{2}$$

Geometrically, The oracle Uw applied to ket s can be visualized as a reflection around the set of orthogonal states to ket w (which is the desired state) written as,

$$U_w = I - 2|w\rangle\langle w| \tag{3}$$

An important thing to note is that the oracle is problem-based so each different problem to be solved we need to design the oracle accordingly.

In the third step, after applying the Oracle function, we apply the diffusion operator, which reflects the amplitudes of the states about the mean of all the states. The diffusion operator does the work of amplifying the amplitude of the solution state(s) and suppressing the amplitude of the non-solution states.

$$U_d = 2\left|s\right\rangle\left\langle s\right| - I \tag{4}$$

Geometrically, we see the action of applying diffusion as a rotation of $U_w\left|s\right\rangle$ about the uniform superposition state ($\left|s\right\rangle$) with an angle θ. So the combined effect of U_w and V_D on $\left|s\right\rangle$ does the work of rotating the $\left|s\right\rangle$ state through an angle $\theta = 2\arcsin\left(\frac{1}{\sqrt{N}}\right)$ towards $\left|w\right\rangle$, in one iteration of applying U_w and V_D on $\left|s\right\rangle$. And it turns out that when we repeat the operation of applying Oracle and diffusion operators to the order of R times, where $R = \frac{\pi}{4}\sqrt{\frac{N}{M}}$, N is the number of elements in the database, and M is the number of marked states we need (Saha et al., 2015), it has the net effect that the amplitude of the desired answer is almost 1, while the amplitudes of undesired answers reduce to almost 0. So following the measurement, we easily find the answer we desired.

Graph Coloring Circuit

There are several methods of implementing Grover's search algorithm in a quantum circuit. For our use, we developed a basic form of a quantum circuit to implement Grover's Search algorithm to solve the graph coloring problem for planar graphs.

Our paper will present an implementation of Grover's Search algorithm to solve the graph coloring for a simple planar graph with 4 nodes and 4 edges (see Fig: 8). We will also present a more general method that works for any planar graph with n nodes and e edges between them.

Suppose, for simplicity's sake, we have the following graph:

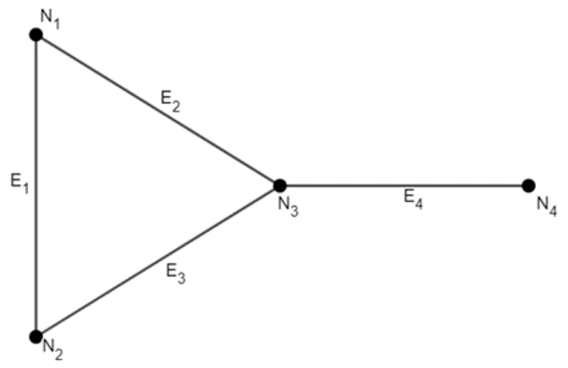

Figure 8

Since () proved that a planar graph can be colored using 4 colors, suppose we have the following four colors with a two-bit binary number representation as:

Binary Digit	Corresponding Color
00	Red
01	Blue
10	Green
11	Yellow

Our problem now is to assign these two-bit binary numbers (or simply these colors) to the $n = 4$ nodes such that no two adjacent nodes have the same binary numbers. To accomplish that, let's develop a system to keep track of the different nodes, their colors, and the information between adjacent nodes.

As a first step, we can take $2n = 8$ qubits and understand that the 4 back-to-back qubit pairs represent each of the $n = 4$ nodes with the first two qubits representing N_1.

This way, the four pairs of qubits' values can be interpreted as the color in the four nodes. Suppose for now that we have these eight qubits with their values as: $\left|01000110\right\rangle$ Because this representation has the first two qubits' value as 00, so (N1) is colored blue. Likewise, this representation implies that nodes (N2), (N3) and (N4) are colored red, blue and green respectively. In a circuit, that would be,

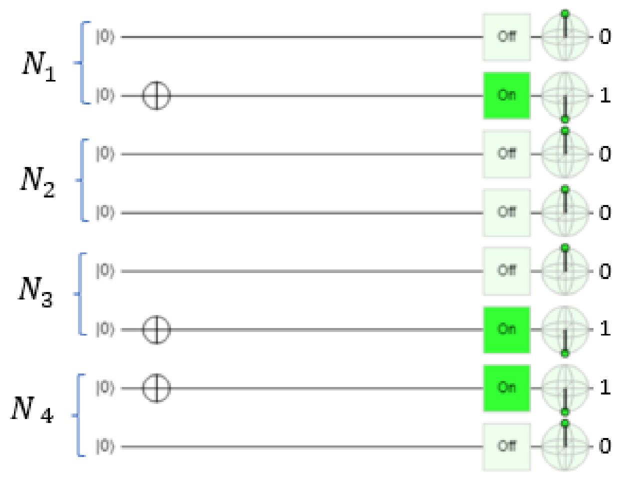

Figure 9

Notice how after applying Grover's algorithm, our final output will be a $2n$-bit binary number whose leftmost two values represent N_1's color, then the next two values represent N_2's color, and so on until the last two values represent N_n's color. For our use, we'll refer to the qubits represented by the N_n node as N_n qubits. So, the second last pair of qubits is the N_3 qubits.

Since these are the only qubits that represent our color, we can apply the Hadamard H gate to make them into a uniform superposition. Any ancilla qubit we need/use later will not need to be initialized because their final values are of no use to us.

We now move on to designing the oracle to invert the phase of the coloring combination in which adjacent nodes have different colors. To achieve that, we break the oracle's function into two parts, so that the gate implementation is simpler. First, we need to identify the solution states whose adjacent nodes are differently colored. Second, we need to invert the phase of the identified solution states.

For now, let's just look at N_1 and N_2 from our specific case. Since they're adjacent, the first part of the oracle must identify if they have the same color. We'll use multi-control Toffoli gates in the following combination (see Figure 10) to see if they have the same color and store its value in an ancilla, which we'll call an edge ancilla. Note that the edge ancilla indexing is the same as the edge indexing from the graph. This is done for simplicity's sake.

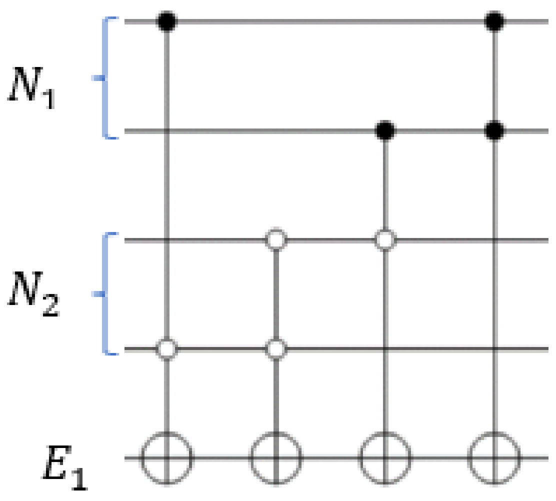

Figure 10

A truth table with all possible inputs in the N_1 and N_2 qubits shows that the edge ancilla returns 0 only when the N_1 and N_2 qubits have different values (i.e., different colors). When the colors in the adjacent nodes are the same, only one of the Toffoli of the four flips the sign of the edge ancilla, making it a 1. Else, two Toffoli gates are activated which act like inverses of one another to give an output of 0 in the edge ancilla.

Because of their frequent use, the combination of these Toffoli gates to check if any pair of adjacent nodes N_i and N_j with edge E_k have different colors or not, we'll be calling them color-checking gates ($C(i,j)$) and their corresponding edge ancilla is E_k. For ease of notation, we'll name the inverse of $C(i,j)$ to be $C(j,i)$ because $C(j,i)$ must have the Toffoli gates in the reverse order of $C(i,j)$.

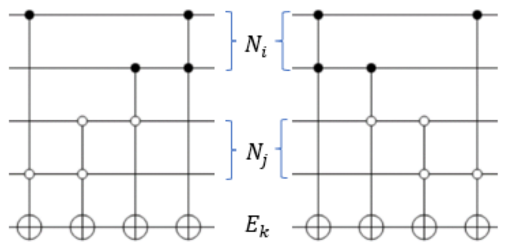

Figure 11

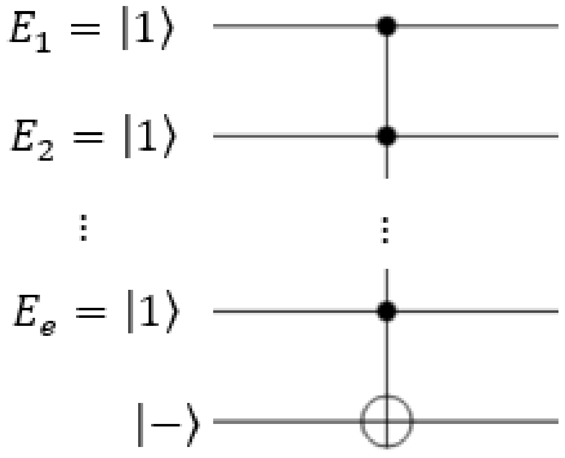

Figure 12

We now design the second part of the oracle to flip the phase of the correct states identified in the first part. For that, we'll use phase kickback (Alsing McDonald, 2011) by using a new ancilla, which we'll refer to as the **negative ancilla**. We'll again use a multi-control Toffoli gate to flip the phase of the solution states.

In the first part of the oracle, we know we'll have e edge ancillas. If all of these edge ancillas return 0, we know that is a valid coloring because the edge ancillas are 0 only when adjacent nodes have different colors.

We now take the edge ancillas as the control and the negative ancilla as the target in a Toffoli gate to flip the phase only when all the edge ancillas return 0. However, computationally, it is more efficient to make the controls that check 1, else we'll need to use X gates before and after all the controls.

Hence, in the initialization step itself, we make all the edge ancillas to 1 to reduce the cost of the quantum circuit. In a diagram, we would have,

So, up until now, we have initialized the states, checked which states have valid coloring and flipped the phase of states with valid coloring. To prevent the effect on phase due to C(i,j), we apply C(j,i) right after the phase kickback completing our oracle.

For the diffusion operator, recall that it can be written as

$$s = H^{\otimes n} |0\rangle \tag{5}$$

$$D = 2 |s\rangle \langle s| - I \tag{6}$$

$$D = 2(H^{\otimes n} |0\rangle) \langle H^{\otimes n} |0\rangle| - I \tag{7}$$

Because D can be written as a reflection about $|0\rangle$ by a change of coordinates using Hadamard gates, we can make a gate, call it M_0 (Diao, 2010), that reflects a state $|x\rangle$ about $|0\rangle$ and apply Hadamard gates before and after the gate to create the diffusion operator.

Mathematically, we want a gate such that,

$$M_0 |0\rangle = |0\rangle \tag{8}$$

$$M_0 |x\rangle = - |x\rangle \quad \text{for } x \neq 0 \tag{9}$$

Notice that M0 is essentially like an oracle that is checking if our solution is not —0⟩ or not. So, we can again use Toffoli gates with negative controls in all the Nn qubits and target in the same negative ancilla as previously used.

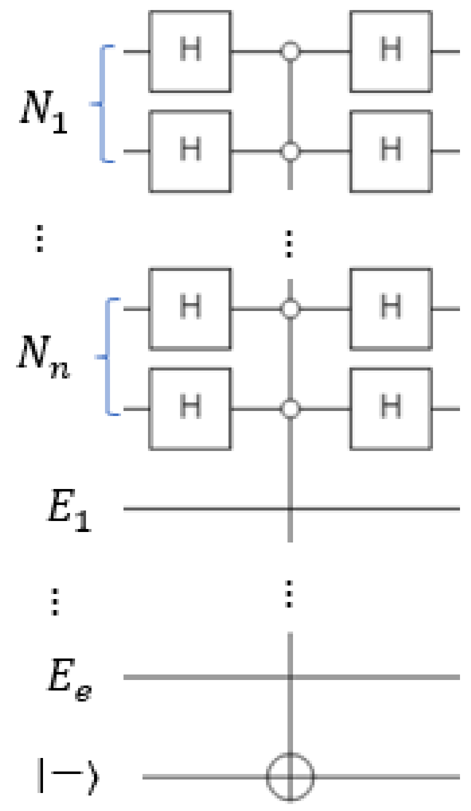

Figure 13

As a pseudo circuit, our final circuit would look like:

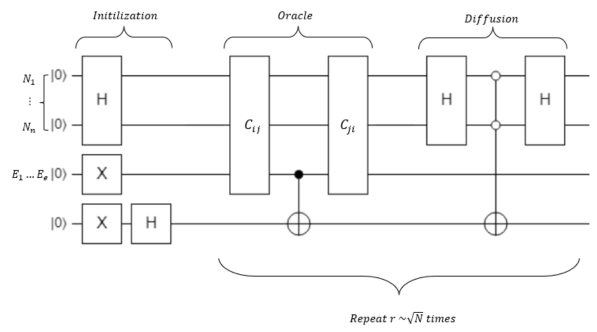

Figure 14

Ideally, the total number of items t we're looking for in an unstructured database is known, and theoretically, as shown above, if we run Grover's algorithm in the order of $\sqrt{\frac{N}{t}}$ times, we'd get the highest probability of finding the correct output.

However, since we do not know how many valid colorings there are for an arbitrary planar graph (i.e., we don't know the value of t beforehand for our cases), we run Grover's algorithm multiple times with a different number of iterations according to an algorithm presented by (Boyer et al., 1998) to find the solutions in the same time order as $O\left(\sqrt{\frac{N}{t}}\right)$.

Discussion

References

Alsing, P. M., McDonald, N. (2011). Grover's search algorithm with an entangled database state. Proc. SPIE 8057, Quantum Information and Computation IX, 80570R. https://doi.org/10.1117/12.883092.

Alama, J. (2009). Euler's Polyhedron Formula. Formalized Mathematics, 16(1), 7-17. https://doi.org/10.2478/v10037-008-0002-6.

Berman, L. W., Williams, G. I. (2009). Exploring Polyhedra and Discovering Euler's Formula. Resources for Teaching Discrete Mathematics. Mathematical Association of America.

Boyer, M., Brassard, G., Høyer, P., Tapp, A. (1998). Tight bounds on quantum searching. Fortschritte der Physik: Progress of Physics, 46(4-5), 493-505.

Diao, Z. (2010). Exactness of the Original Grover Search Algorithm. Physical Review A, 82. https://doi.org/10.1103/PHYSREVA.82.044301.

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing - STOC '96. https://doi.org/10.1145/237814.237866.

K. Appel, W. Haken "Every planar map is four colorable. Part I: Discharging," Illinois Journal of Mathematics, Illinois J. Math. 21(3), 429-490, (September 1977)

Mukherjee, S. (2022, February 8). A Grover search-based algorithm for list coloring.

Saha, A., Chongder, A., Mandal, S. B., Chakrabarti, A. (2015, December). Synthesis of vertex coloring problem using Grover's algorithm. 2015 IEEE International Symposium on Nanoelectronic and Information Systems, pp. 101-106. IEEE.

Lack of superspecialist doctors threatens the future of Nepal's health sector - OnlineKhabar English News. (2023, February 13). https://english.onlinekhabar.com/lack-superspecialist-doctors-nepal.html

Appendix A: Qbits and Gates

Appendix B: Quantum Circuits