Quantum Graph Coloring

Rana Urek and Sadi Gulcelik
Columbia University

(Dated: November 20, 2023)

We explore the graph coloring problem and potential quantum approaches.

I. INTRODUCTION TO GRAPH COLORING:

When we seek to color a map, one (sensible) requirement is that every constituent of the map should have a different color. A simple solution to the problem would be to pick a unique color for every constituent, but this could get annoying for a multitude of reasons: imagine trying to tell the difference between two similar shades of green assigned to neighboring states. Instead, we can think about trying to color a map with a finite set of colors. For the contiguous United States it turns out that one needs at least 4 colors in order to achieve a coloring where neighboring states are all differently colored.

It turns out that for (reasonable) 2 dimensional maps, 4 colors is always sufficient.[1] Despite the seemingly 'simple' nature of the problem, it turns out that the proof is incredibly convoluted and required heavy use of computation in order to check all the cases once they had been reduced to a finite (but enormous) number.

Map coloring is a special case of graph coloring, an extension of the above problem that allows us to consider that any two 'countries' (which we will henceforth refer to as nodes) have 'borders' (which we will now term edges). Graph coloring is an NP complete problem: roughly speaking, this means that the problem is computationally difficult to solve, and that we do not believe there is a 'fast' solution. To describe what we mean by 'fast,' we will introduce some complexity theory.

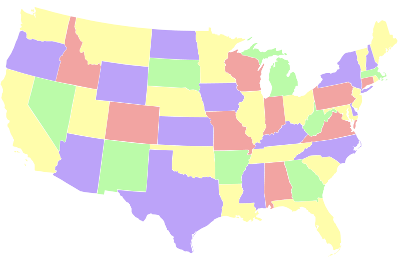

FIG. 1. https://www.cs.cmu.edu/ bryant/boolean/maps.html

[1] By reasonable, we mean that if there are countries with non-contiguous territories, they are not required to be the same color, that corners do not count as borders, and that the map is planar / can be 'projected' onto a planar surface (the requirement being that the planar graph encodes the same constraints.

A. Complexity Theory

A number of computer science problems fall into one of two classes: polynomial time and exponential time. Polynomial time algorithms are those that can be solved in a number of computations that are proportional to some polynomial function of the input; that is given an input size N, its runtime is bounded by CN^p, where p is finite (and positive). Exponential algorithms are a different beast; their runtimes are bounded by Cb^N, where b is finite and greater than 1. For sufficiently large N, exponential runtime algorithms will always take longer than polynomial time algorithms since

$$\lim_{N\to\infty} \frac{b^N}{N^p} \to \infty \quad (\text{given } p > 0, \quad , b > 1, \quad N > 0). \quad (1)$$

NP problems are the class of problems that can be verified in polynomial time. P, the class of problems that can be solved in polynomial time, is a subset of NP. It is an open question of whether P and NP are the same (whether all NP problems can be solved in polynomial time). It is widely believed that this is not the case, and that there are classes of problems in NP which can never be solved in polynomial time on a classical computer.

The hardest of these problems (that are believed to be unsolvable in polynomial time) are called NP-complete. NP- complete problems have the property that a (potential) polynomial time solution to one of them would imply and yield a polynomial time solution to all other NP complete problems, and in general all problems in NP. Thus, any method to solve NP problems polynomially would prove P=NP. [1]

II. QUANTUM ADVANTAGE

In our exploration of graph coloring, in addition to a survey of different quantum algorithms for solving combinatorial problems, we seek to examine the question: are quantum computers useful for solving NP complete problems?

The closest benchmark we have is Shor's algorithm, which takes a difficult classical problem (that is potentially exponential time, but strongly not believed to be NP-complete), and solves it in polynomial time. However, this particular problem is very well suited to quantum algorithms due to the utility of QFT in solving the factorization problem.

The two algorithms that seem the most directly applicable to our graph-coloring problem are QAOA and Grover's algorithm. Grover's algorithm is a variation on classical search that can provide a reduction on the order $O(\sqrt{N})$: that is, it reduces the runtime to the square root of the classical run time. While a $O(\sqrt{N})$: reduction in runtime is certainly useful and powerful, in cases like $O(N^6) \rightarrow O(N^3)$, Grover's algorithm is not powerful enough to kick algorithms from exponential into the polynomial region $O(4^N) \rightarrow O(2^N)$,

We hypothesize that the limited advantage provided by feasible runtime reductions, that is, those on the order of $O(\sqrt{N})$, cannot be overcome by near term quantum computers due to error rates, limitations on the number of logical qubits, and the difficulties of translating classical optimization and heuristics into quantum algorithms.

III. METHODOLOGY AND RESULTS

We examine and develop several different approaches to solving the graph coloring problem [2]. And evaluate them according to complexity, accuracy, and number of gates (to protect against error). For each algorithm, we ultimately seek to conclude whether the algorithm obeys our hypothesis that near term quantum computers will not exhibit a quantum advantage in the graph coloring problem.

A. A classical-looking algorithm

We design and examine an algorithm outlined in a quantum computing paper [3]. The paper describes the complexity of their method as $O(N^2)$ in the number of qubits and $O(N^4)$ in number of gates. We believe that the claim is misleading and that the algorithm proposed in the paper ultimately provides no quantum advantage whatsoever. We also note that we had to modify the algorithm due to mistakes in the code presented in the paper.

The algorithm itself resembles the way one would conceive of solving this problem classically, in the sense that we are generating all the possible colorings of a graph and checking with the graph edge information to destroy the generated colorings that violate the *'adjacent vertices must be colored differently'* condition.

We have a general code for a k-coloring problem where we assign k many qubits to all the vertices in the graph. We use *'ancillary qubits'* with controlled X gates and Toffoli gates to generate all the possible colorings, meaning only one color assigned to one vertex of the graph (no more or no less). We encode in the *'same color qubits'* the critical information of which vertices in our generated colorings share the same coloring. Then all we need to do is compare the information encoded in the *'same color qubits'* with the information encoded in the *'graph edge qubits.'* We use multi controlled X gates for this last step.

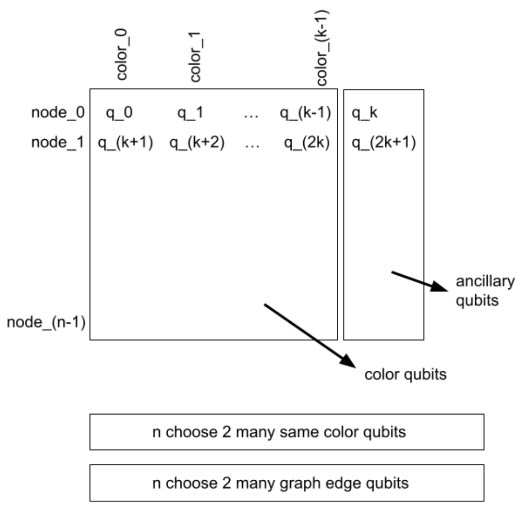

FIG. 2. A visualization of the qubits required to perform this algorithm

Ultimately, the qubits for our nodes end up in a superposition of correct colorings and 'destroyed' states, which are represented by the one hot encoding of the node being set to all 0. While one iteration of this algorithm certainly has a polynomial runtime, the authors of the paper seem to ignore the issue of the probability of getting a result. Their complexity section begins by correctly noting that "The worst-case complexity of any general k-colouring algorithm on a classical (Turing) machine is exponential in the size of the graph." However, their analysis of their quantum algorithm only considers the number of qubits and gates.[3]

In our simulations, we note that the number of null results are very large in comparison to the number of correct results. Upon examining the algorithm in depth, we conclude that the algorithm is ultimately doing nothing more than evaluating a random coloring of the graph. Since there are an exponential number of ways to randomly color the graph, it is exceedingly unlikely, given any constant number C of valid colorings. Specifically,

$$\mathbb{P}(\text{valid coloring}) = \frac{\text{C}}{k^N} \tag{2}$$

If we want the probability of finding a valid coloring given C and N to be greater than some non-zero constant p, we need to run the algorithm on the order of k^N times.

B. Grover's algorithm

Grover's algorithm is a safe way to gain a complexity advantage over classical computers.

Grover's algorithm works by combining an oracle and a diffuser in order to successively amplify valid solutions. Grover's algorithm has a runtime of $\sqrt{O_c(n)}$ where $O_c(n)$ is the classical runtime, and returns a valid result with probability at least $\frac{1}{2}$. We borrowed our diffuser from the qiskit textbook, but otherwise designed our architecture and oracle independently. [4]

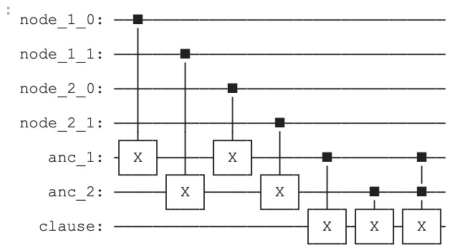

FIG. 3. A visualization of the qubits required to perform this algorithm

In order to implement Grover's algorithm, we used a much denser encoding than the above paper and achieved better complexity than the paper in the number of qubits. For a k-coloring problem, our grid encodes the nodes as a collection of b qubits where $b \sim \log_2 k$ is the number of bits required to encode the color space. In addition to these $n \cdot \log_2 k$ qubits, we need clause checking qubits in order to evaluate the constraints for our oracle. For a 2 color problem, we only need as many clause qubits as the number of edges, as we do not need any intermediate qubits to help combine the results of an edge checking operation. As we add colors, the number of qubits per node grows as $\log_2 k$ in order to be able to combine the results of comparison options in the tree structure below.

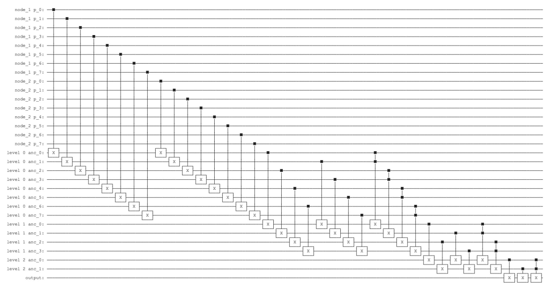

FIG. 4. Clause computation circuit for 256-color nodes

C. QAOA

We implemented a third algorithm known as quantum approximate optimization algorithm (QAOA). This algorithm uses VQE (Variational Quantum Eigensolver) principles we've learned in our course. The paper we started out with has the following Hamiltonian for the k-coloring problem for a graph G (V, E):

$$H = A\sum_{v}(1-\sum_{i=1}^{k}\mathrm{x}_{v,i})^2 + A\sum_{(uv)\epsilon E}\sum_{i=1}^{k}\mathrm{x}_{u,i}\mathrm{x}_{v,i} \quad (3)$$

where $\mathrm{x}_{v,i}$ corresponds to a binary variable which is 1 when vertex v is colored with color i, and 0 otherwise. [5] The second term in the Hamiltonian penalizes the same coloring of two vertices that are connected with an edge, similar to the way one would conceive of writing the Hamiltonian of an antiferromagnetic Ising model. The first term of the Hamiltonian enforces the constraint that each vertex is colored with only one color, and penalizes cases where more than one color or no colors are assigned to one vertex.

We've implemented this algorithm twice. We used the qiskit textbook code for the 2-coloring problem and modified that code for the 4-coloring case. [6] The 2-coloring problem is very simple since we can ignore the first term in the Hamiltonian completely. There is no need to assign two qubits for each node to store the 2-coloring information as the paper suggests. We can simply say vertex v corresponds to a binary variable which is 1 when it is colored with the first color and 0 when it is colored with the second color. And the second term of the Hamiltonian (only term of importance to us) can be expressed with the following gates for a graph with n vertices:

$$\mathrm{H}_{problem} = \gamma\sum_{(\mathrm{v}_i\mathrm{v}_j)\epsilon E}\mathrm{I}_0\otimes\mathrm{I}_1\cdot\cdot\mathrm{Z}_i\otimes\mathrm{Z}_j\cdot\cdot\otimes\mathrm{I}_{n-1} \quad (4)$$

```
qc_p = QuantumCircuit(nqubits)
for pair in list(G.edges()):
    qc_p.rzz(gamma, pair[0], pair[1])
    qc_p.barrier()
```

For the wave function ansatz, we'll rotate each qubit in our ensemble around the x axis by some parameter beta. Here is Hamiltonian that corresponds to this operation:

$$\mathrm{H}_{ansatz} = \beta\sum_{i\epsilon V}\mathrm{I}_0\otimes\mathrm{I}_1\cdot\cdot\mathrm{X}_i\cdot\cdot\otimes\mathrm{I}_{n-1} \quad (5)$$

Then all we need to do is optimize the parameters beta and gamma to find a minimum to the following energy function $\langle\Psi(\beta,\gamma)|\mathrm{H}_{problem}|\Psi(\beta,\gamma)\rangle$

The 4 coloring problem on the other hand is trickier in the sense that ignoring the first term of Equation (7) now has major consequences. So we thought of a way to get around this. Unlike what Equation (7) Hamiltonian suggests, we are not using 4 qubits assigned to each vertex to encode the 4 coloring information. Instead we are assigning 2 qubits per vertex where 00, 01, 10 and 11 combinations of these two qubits each correspond to a different coloring of the vertex. This way we can ignore the first term again, since all measurements of our assigned qubits correspond to a valid coloring. We will slightly alter our problem Hamiltonian in the following way:

```
qc_p = QuantumCircuit(nqubits)
for i in range(0,ncolors):
    for pair in list(G.edges()):
        qc_p.rzz(gamma, pair[0], pair[1])
        qc_p.barrier()
```

Even though this is a viable solution for our purposes, we are getting more errors with this implementation of 4-coloring as opposed to the simple 2-coloring case. The reason lies in the code: (1) we are unnecessarily penalizing the cases where two vertices connected by an edge are colored with let's say 00 and 01 colors (2) we end up rewarding 01,10 colorings for connected vertices more than we should.

We would not have this problem if we completely implemented the Hamiltonian in Equation (7) but as it turns out, the gate based implementation of the first term in that Hamiltonian as it is, is very tricky and the above is the best solution we were able to find. In the end for a complete graph with 4 nodes and 4 coloring (with 24 valid solutions), we were able to get 44 high frequency colorings where we had 66% success rate with the frequency of valid colorings. (In the future, we can try to determine a better threshold for what constitutes a 'high frequency' measurement to increase our success rate). We went from the search space of 256 possible colorings to 44 colorings with the implementation of our 4-coloring QAOA algorithm. The 2-coloring QAOA algorithm was a bigger success, since all our high frequency solutions were valid colorings.

For a graph with n nodes, our QAOA algorithms use $n \log_2 k$ many qubits, k being the number of colors. And worst case scenario we use $(2n + \binom{n}{2}) \log_2 k$ many gates.

D. Grover's algorithm, revisited

We now return to Grover's algorithm to see whether we can apply the diffusion operator to our first algorithm in order to do some amplitude amplification. First, we note that it is impossible to do so directly since the algorithm uses the same ancillary qubit for most of its operations and thus needs to reset that qubit multiple times. Doing so does not actually save any qubits, as each reset requires an additional ancillary qubit to be introduced and swapped with the qubit to be reset. However, Grover's algorithm often requires multiple iterations of the oracle to be applied, and should use the same set of clause qubits for each version of the oracle (otherwise the number of qubits required for clause qubits in the algorithm will scale by a factor of the number of iterations). Furthermore, the algorithm uses these qubits for both clause checking and to store the state of the system itself, which is not compatible with grover's algorithm, which only resets the clause qubits.

Hence, we modify the paper's algorithm in order to achieve a version compatible with Grover's algorithm. To do so, we restructure the architecture to store only the one hot encoding of the nodes colors, and introduce a clause qubit for every constraint: that exactly one color be turned on for each node, and that edges do not share colors. We iterate through first the nodes to encode the first constraint in the clause qubits, then iterate through the edges to encode the edge constraints, calculate our output qubit, then repeat the above process to uncompute our clause qubits. We then wrap our oracle in Grover's algorithm and demonstrate that this modified version of our first algorithm works with Grover's, although the method of encoding chosen in the paper creates a much larger number of qubits than our above algorithm, and also forces us to actually impose the constraint of "every node has to have exactly one color" manually rather than by design.

IV. DISCUSSION

We examine the complexity of Grover's algorithm for 4 - coloring.

Since the classical algorithm we based grover off has complexity $O(4^N)$ in the number of nodes (a brute force implementation), our implementation of Grover's algorithm has complexity $O(2^N)$, assuming that the oracle takes $O(1)$ time to run; this O(1) is roughly possible if we assume that each node only has a constant number of edges, and we can do edge computations in parallel, since the gate propogation is constant with respect to k (specifically, proportional to $log(log(k))$, the depth of the edge comparison circuit). While this might seem like a fascinating improvement, we have yet to consider two factors:

A. Error Propagation

Quantum gates have errors, and these errors propagate. Ignoring the diffusion operator (which uses far less gates than the oracle and thus does not contribute significantly to the total error), all our gates are contained in the edge comparison operation. To compare an edge we use $2b + b + b/2 + ... \approx 4b$ CX gates, and $b/2 + b/4... \approx b$ CCX gates, where $b = log_2 k$ and k is the number of colors (thus, $b = 2$ in our 4 color case). Then, assuming say $3N$ edges per node, we have a total of, $12Nb = 24N$ CX gates, and $3Nb = 6N$ CCX gates in every iteration. Given probabilities p_1 and p_2 of an errorless computation (that is, without an error that affects the result), the probability of a good result is given by $p_1^{2^N \cdot 24N} \cdot p_2^{2^N \cdot 24N}$. This is a miniscule probability for any reasonable N and error rate, and since solutions (should be) sparse in the search space (otherwise the question is trivial), it is unlikely for us to randomly happen upon the right result.

As we were unable to test Grover's algorithm on a real quantum computer(due to qubit constraints), we will demonstrate what the performance of the quantum algorithms look like for estimated values of p_1 and p_2. We will also assume that the quantum computers gate operations have the same rate as a classical computer, and allow it to make mistakes 10% of the time. We try it for error rates 10^{-6} for CX gates and 10^{-5}, both of which are incredibly ambitious [2].

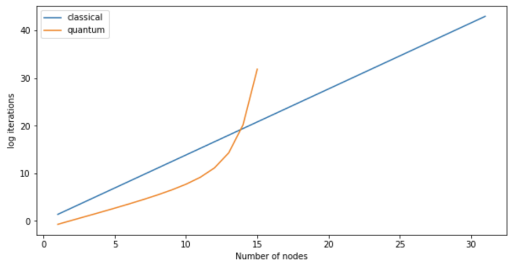

FIG. 5. Complexity comparison

From the Complexity Comparison graph, it is first clear that we have made too many unreasonable assumptions (such as identical clock rates) in favor of the quantum algorithm as the graph indicates that it outperforms the classical algorithm for small N. Without error correction, even with all these assumptions, the performance of the quantum algorithm actually gets worse as we increase the complexity of the problem; the errors, even for such small rates, are propagating very quickly. Error correction is necessary to the proper functioning of this algorithm, which in turn will drastically reduce the number of qubits.

Assuming we want to solve a reasonably sized graph of 20 nodes and 60 edges, we need 40 qubits for the nodes, and $60 \cdot 3$ qubits for the clause checking. As 220 multiplied by an error checking factor is far too many qubits, especially given the limited (and in fact classically solvable) size of our problem.

B. Classical Optimization

Optimal classical algorithms can solve 4-coloring in $O(1.73^N)$ [7]. This is faster than even an errorless version of Grover. While one might argue that one could use Grover to speed up this $O(1.7^N)$ algorithm, instead of the earlier $O(4^N)$ brute force version, applying classical optimization to quantum algorithms is non-trivial (additionally, even if we assume it's possible, we run into the above issues). One particularly difficult issue is that Grover's algorithm only works with a constant branching factor, while classical algorithms take advantage of pruning in order to reduce the branching factor [8]. The heart of the issue is that the classical algorithm is pruning the very branches that Grover is able to compute simultaneously—in effect, the classical algorithm is matching the quantum advantage by replacing superposition of branches with good choice of branches.

C. Note on colors

In many of our implementations, we used a power of 2 for the number of colors due to the ease of implementation. A quick workaround to do 3 colors (for example) is to add a 'ghost node' that connects to all nodes, so that the ghost node be forced to occupy the 4th color and reduce the part of the system we care about to 3 colors. We are in the process of looking for more efficient solutions.

V. CONCLUSION

The first paper that we encountered in our research was effectively a dud: it offered no quantum advantage and merely evaluated a random coloring of the graph with quantum computing. It was not a useful algorithm. Our experiments with Grover and QAOA were more successful, although we had difficulty measuring effective error rates due to the limited number of qubits available to us on actual quantum computers. With respect to complexity, we were able to test the quick convergence of Grover's algorithm in our simulations with the immediate apperance of large numbers of solutions in a small number of oracle/diffuser iterations.

In our next steps, we will attempt to do complexity analysis of QAOA, and find access to slightly larger quantum computers in order to test both this complexity analysis, and effective error rates using non-trivial cases (k>2) for all our algorithms. Upon exploring the error rates, the logical next step would be efficient error correction, to see if we can modify the diffusion operator to kill errors and re-recover solutions whose amplitudes may have been erroneously reduced. Finally, we will research the speedups used by classical algorithms to achieve much better complexity, and research which of them could be applied to our quantum algorithms and see how close we can get to a $(O(\sqrt{N}))$ reduction on the optimal classical algorithm.

However, based on our research so far, it appears as if current error rates (particularly for the multi-qubit gates that we use so many times) combined with the limitation on the qubit count (which limits the opportunity for error correction), coupled with the final punch of outstanding classical algorithms may disqualify near-term quantum algorithms/computers from the realm of graph coloring and similar combinatorial problems.

[2] Although the rates could be considered somewhat reasonable as many errors early on in the implementation of grover might not affect the final result

[1] P versus NP problem | mathematics | britannica.

[2] Github repository of all our code.

[3] M. Clerc, A general quantum method to solve the graph K-colouring problem.

[4] Grover's algorithm.

[5] A. Lucas, Ising formulations of many NP problems, **2**.

[6] Solving combinatorial optimization problems using QAOA.

[7] F. V. Fomin, S. Gaspers, and S. Saurabh, Improved exact algorithms for counting 3- and 4-colorings, in *Computing and Combinatorics*, Vol. 4598, edited by G. Lin (Springer Berlin Heidelberg) pp. 65–74, ISSN: 0302-9743, 1611-3349 Series Title: Lecture Notes in Computer Science.

[8] A. Sequeira, L. P. Santos, and L. S. Barbosa, Generalised quantum tree search, 2103.13976.