Final Report for TJHSST Computer Systems Lab

Graph Coloring with Grover's Algorithm: Optimizing Time and Space Efficiencies

Soham Jain

June 4, 2025

Dr. Yilmaz - Period 1

Table of Contents

Abstract

Graph coloring, an NP-complete problem, involves assigning colors to vertices of a graph such that no adjacent vertices share the same color. With an exponential time complexity of $O(m^V)$, where m is the number of colors and V is the number of vertices, graph coloring is a problem that classical algorithms struggle to solve due to high time complexity and memory consumption. This project leverages Grover's algorithm, a quantum search algorithm with $O(\sqrt{N})$ time complexity for unstructured searches, to solve the graph coloring problem on a map of the 50 states. In particular, this study compares time and space complexities in Qiskit and Q#, common programming languages for quantum computing. Qiskit achieved an average execution time of 14.94 seconds over 20 trials, outperforming Q# at 34.78 seconds. The classical recursive Python algorithm was slowest at 43.75 seconds. Furthermore, Q# demonstrated better space efficiency with a circuit depth of 26-38 gates compared to Qiskit's 34-42 gates. This study's finding suggest that Qiskit was the most efficient programming language in terms of time, but Q# had better space efficiency. Quantum approaches also showed competitive performance against classical methods. Current quantum hardware limits testing on larger graphs, however. Future work could apply the method to US counties or other constraint satisfaction problems. Additional quantum languages, such as Cirq and Ocean, could be also tested in the future.

I. Introduction

Graph coloring is a constraint satisfaction problem that involves assigning a set of colors to the vertices of a graph such that no two adjacent vertices share the same color. Moreover, graph coloring is a type of NP-complete problem, as it can be verified in polynomial time but takes an exponential number of steps to solve. NP-complete, in particular, represents the hardest problems in NP.

Time and space complexities are common challenges that must be addressed to solve the graph coloring problem. As the number of vertices increases, the number of possible colorings grows exponentially, leading to significant time and memory challenges for classical algorithms. This is demonstrated by the problem's $O(m^V)$ time complexity, with m representing the number of colors and V representing the number of vertices. Thus, coloring a graph of the 50 United States using four colors results in over 59,000 possible configurations, from which there are only a few valid solutions.

This project applies Grover's algorithm, a quantum search algorithm, to the graph coloring problem across multiple programming languages—namely Qiskit and Q#. The goal is to compare time and space efficiencies across both programming languages.

II. Background

Classical approaches to solving the graph coloring problem typically rely on either greedy or recursive algorithms. The greedy algorithm colors each vertex one at a time by ordering the list of colors in the constraint set, as well as ordering the vertices. This process moves linearly through the graph, making decisions based only on local information. While efficient and easy to implement, greedy algorithms are not guaranteed to find a valid solution if one exists, especially for more complex graphs where the coloring of early vertices can block valid options later on. In contrast, recursive algorithms use a backtracking approach, attempting to color the graph by exploring all possible assignments through a depth-first search. If a conflict arises, the algorithm backtracks to a previous decision point and tries a different color. This method is guaranteed to find a solution if one exists, but it quickly becomes computationally expensive as the number of vertices or colors increases, since it may need to explore an exponentially large space of possibilities.

Quantum computing provides a compelling alternative by reframing the search process itself. Grover's algorithm uses superposition to evaluate multiple colorings simultaneously and amplify correct solutions. An oracle function is used to mark the valid colorings, and a diffuser amplifies the probability of measuring one of these valid solutions. While this does not reduce NP-complete problems to polynomial time, it offers a quadratic speedup over classical unstructured search. Prior research by Kjer (2023) applied Grover's algorithm to a map-coloring task involving 13 provinces in Canada using Qiskit. In contrast, this project scales the problem to 50 U.S. states and compares performance across two quantum programming environments, Qiskit and Q#, highlighting both time and space efficiency differences. Additionally, Clerc (2023) observed that for graphs constrained to fewer than four colors, certain instances of the problem become solvable in polynomial time. However, since this study uses four colors, the problem remains NP-complete.

III. Applications

Graph coloring belongs to a broader category of problems known as constraint satisfaction problems, or CSPs. These are problems where a solution must meet a set of conditions or restrictions, and they appear in many areas of science, engineering, and optimization. In a typical CSP, variables must be assigned values from a given domain in such a way that a predefined set of constraints is satisfied. The graph coloring problem fits this structure exactly as vertices are variables, colors are values, and the constraint is that no two connected vertices can share the same color.

This underlying structure makes graph coloring a useful model for a range of real-world CSPs. Examples include scheduling tasks where no overlapping assignments can occur, assigning frequencies in communication networks to avoid interference, or solving logic-based puzzles like Sudoku. In all these cases, a valid solution must be found within a large space of possibilities while respecting strict limitations. Because of this, any improvement in how efficiently graph coloring can be solved can be applied more broadly to CSPs as a whole. This project's focus on optimizing performance with Grover's algorithm aims to contribute to that larger effort by offering a more scalable quantum-based approach to constraint satisfaction.

IV. Methods

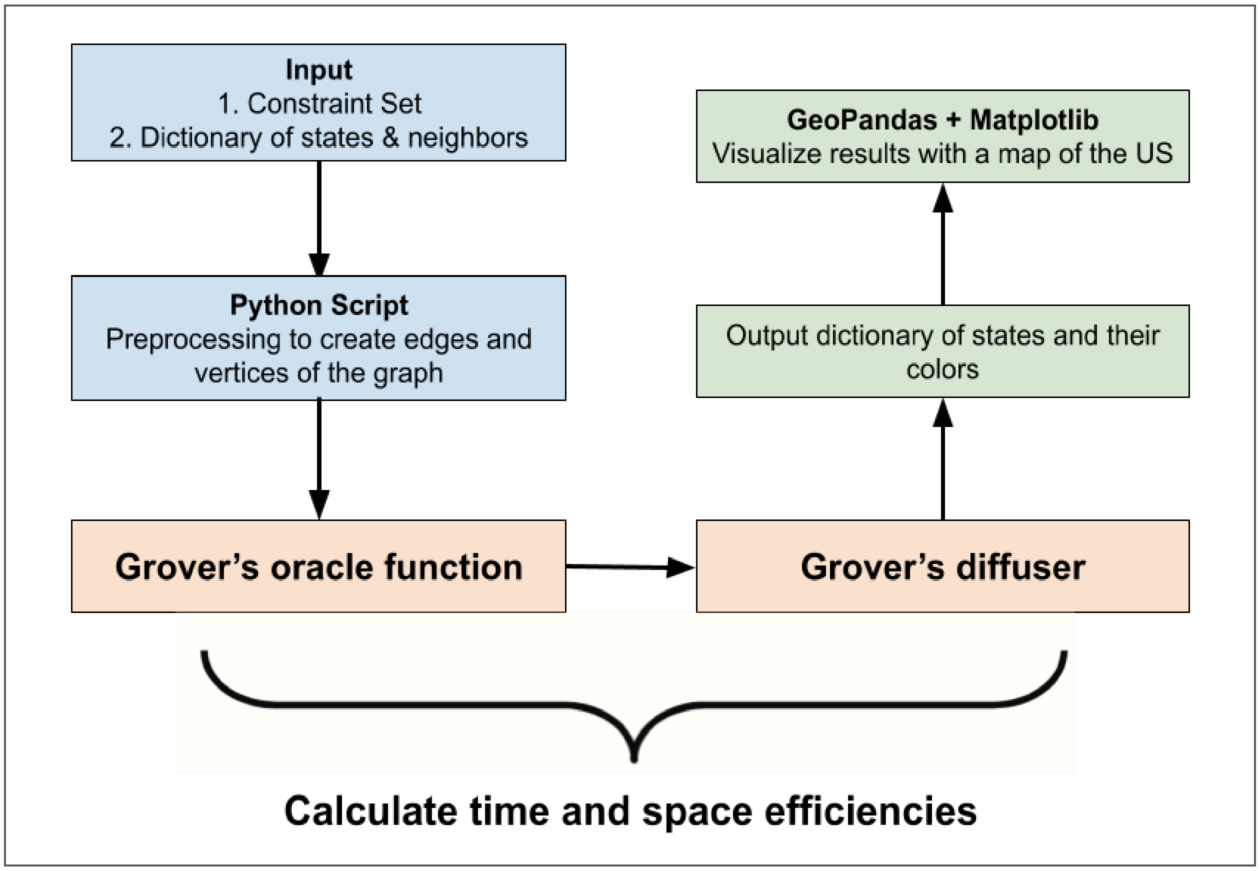
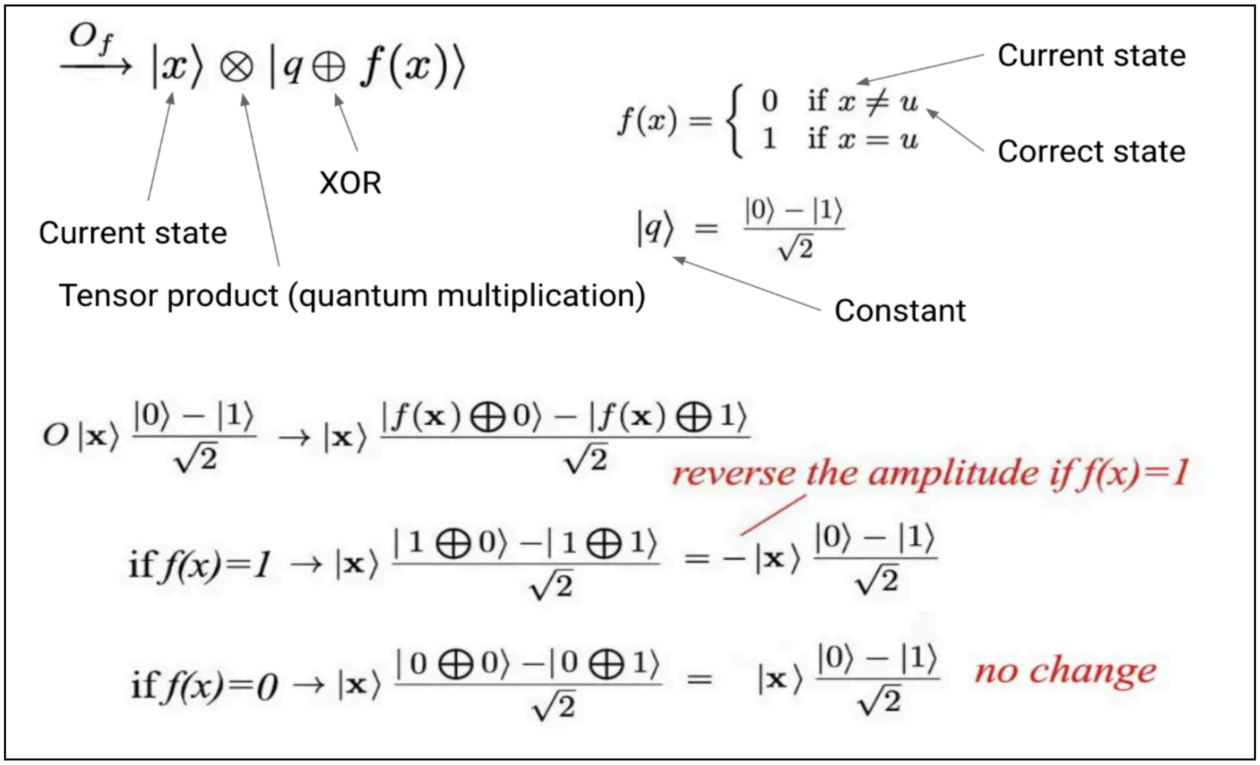

Input
1. Constraint Set
2. Dictionary of states & neighbors
Python Script
Preprocessing to create edges and vertices of the graph
Grover's oracle function
Grover's diffuser
Output dictionary of states and their colors
GeoPandas + Matplotlib
Visualize results with a map of the US
Calculate time and space efficiencies

Fig. 1. Systems Architecture

Graph coloring is a constraint satisfaction task in which each U.S. state represents a vertex in a graph, and edges connect states that share a border. Fig. 1 shows the systems architecture for this project. The input consists of a Python dictionary structure, where each key is a state and each value is a list of neighboring states. The goal is to assign one of four colors—red, green, blue, or yellow—to each state such that no two adjacent states receive the same color. The output is another dictionary mapping each state to a color in the constraint set. The constraint set and adjacency structure are manually defined and preprocessed using standard Python data types.



$$\xrightarrow{O_f} |x\rangle \otimes |q \oplus f(x)\rangle$$

$$f(x) = \begin{cases} 0 & \text{if } x \neq u \\ 1 & \text{if } x = u \end{cases}$$

$$|q\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

$$O|\mathbf{x}\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \rightarrow |\mathbf{x}\rangle \frac{|f(\mathbf{x}) \oplus 0\rangle - |f(\mathbf{x}) \oplus 1\rangle}{\sqrt{2}}$$

$$\text{if } f(x)=1 \rightarrow |\mathbf{x}\rangle \frac{|1 \oplus 0\rangle - |1 \oplus 1\rangle}{\sqrt{2}} = -|\mathbf{x}\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

$$\text{if } f(x)=0 \rightarrow |\mathbf{x}\rangle \frac{|0 \oplus 0\rangle - |0 \oplus 1\rangle}{\sqrt{2}} = |\mathbf{x}\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

Current state
Correct state
XOR
Current state
Tensor product (quantum multiplication)
Constant
reverse the amplitude if f(x)=1
no change

Fig. 2. Grover’s Algorithm Formulas

The mathematical formulas for Grover’s algorithm are shown in Fig. 2. Three algorithms were implemented and tested: a recursive backtracking algorithm in Python as a classical baseline, as well as Grover’s algorithm using both Qiskit and Q#. Quantum circuits were built using existing libraries, including Hadamard transform operations for superposition. In all cases, the solution space was evaluated for color assignments that met the adjacency constraint.

Grover’s algorithm was implemented by encoding each possible state (coloring) into a superposition and applying an oracle that marked valid configurations based on the constraint logic. The diffuser then amplified the amplitude of marked states. Circuit construction in Qiskit relied on the AerSimulator and built-in oracle/diffuser methods, while Q# used the Microsoft Quantum Development Kit. Performance metrics included total runtime (averaged over 20 trials per method) and space efficiency, approximated through circuit depth for quantum solutions. A simplified pseudocode of Grover's implementation involved: (1) initializing superposition with Hadamard gates, (2) applying the oracle to flip the phase of valid states, and (3) executing the diffuser to increase the probability of measuring a correct result.

V. Results

	Average Time (20 trials)	Circuit depth
Qiskit	14.94 seconds	34-42 gates
Q#	34.78 seconds	26-38 gates
Recursive (Python)	43.75 seconds	N/A

Table 1. Time and Space Efficiency Results

The time and space efficiency results of applying Grover's algorithm to graph coloring are shown in Table 1. On average, the recursive backtracking algorithm implemented in Python required 43.75 seconds to find a valid coloring for the U.S. state map. In comparison, the Qiskit implementation of Grover's algorithm completed the same task in an average of 14.94 seconds across 20 trials, demonstrating a significant speed advantage. The Q# version, while slower with an average runtime of 34.78 seconds, exhibited better space efficiency based on quantum circuit depth, ranging from 26 to 38 gates compared to Qiskit's 34 to 42 gates. These results suggest that Qiskit offers faster execution due to its highly optimized AerSimulator, whereas Q# provides better memory management through more efficient circuit construction. Overall, the quantum methods outperformed the classical baseline in at least one of the two key metrics, validating the potential of Grover's algorithm as a viable optimization for large-scale constraint satisfaction problems.

VI. Limitations

One key limitation of this work is that the time efficiency achieved using Grover's algorithm remains comparable to that of classical approaches, offering only modest improvements. Additionally, space efficiency could only be evaluated across different quantum programming languages, rather than against classical methods. The limited availability and capability of current quantum environments also restrict scalability; problems involving more vertices or a larger constraint set, such as those based on U.S. counties rather than states, cannot yet be effectively addressed using Grover's algorithm. These limitations highlight the current boundaries of quantum computing in practical applications of graph coloring.

VII. Conclusion

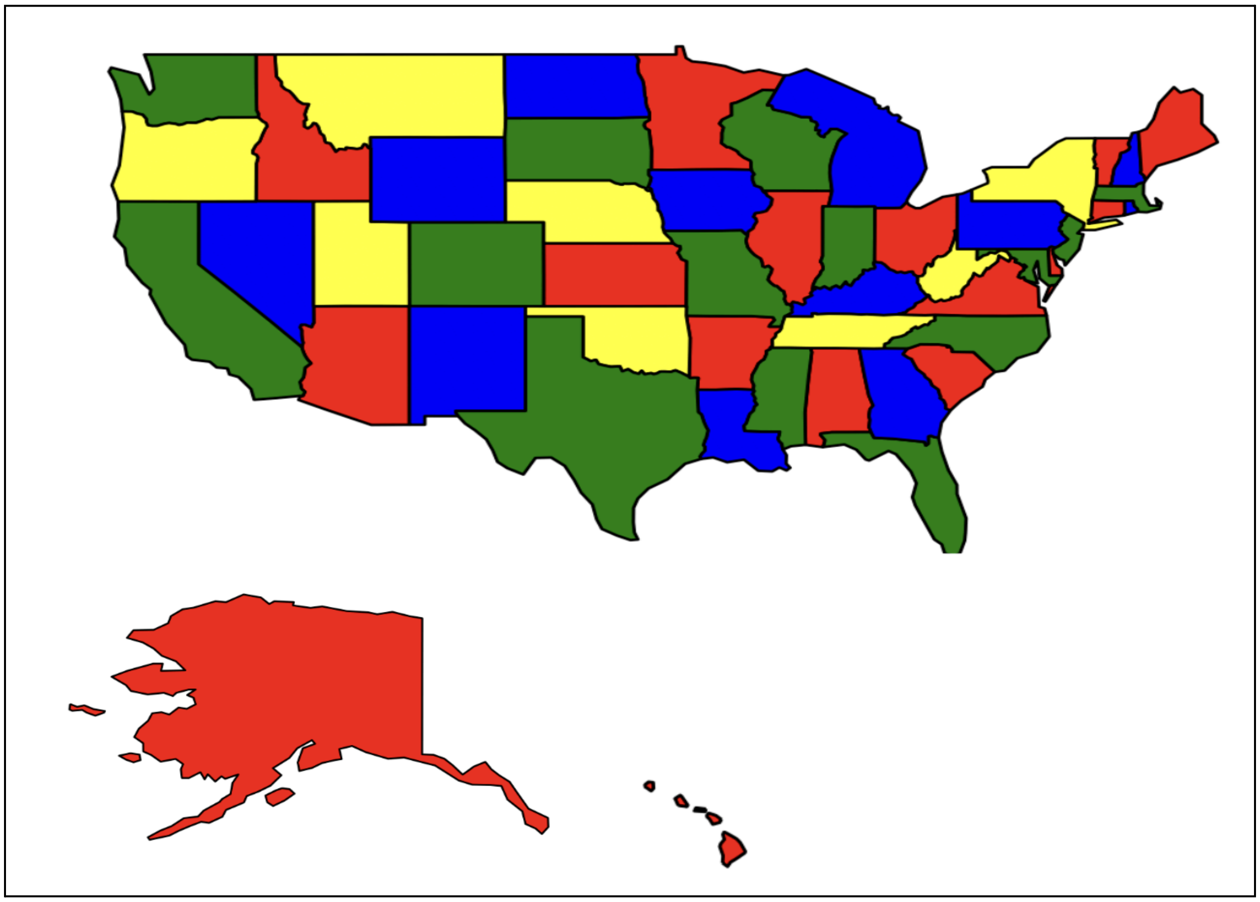


Fig. 3. Graph Coloring with Grover's Algorithm Output

Grover's algorithm successfully colored a 50-vertex US map, as shown in Fig. 3. This project explores the implementation of Grover's algorithm to solve the graph coloring problem more efficiently than classical approaches. Furthermore, by implementing the algorithm in both Qiskit and Q#, this project compares time and space efficiencies across quantum programming environments. The results suggest that Qiskit offers faster execution, while Q# demonstrates slightly better space efficiency. Quantum methods also require less memory in certain cases. This suggests that quantum computing holds potential for solving constraint satisfaction problems more efficiently as hardware improves. The novelty of applying Grover's algorithm to a 50-state map also sets a foundation for future work on larger datasets. In particular, expanding this method to U.S. counties or other real-world applications could better demonstrate its advantages.

VIII. Future Work

Future work will focus on scaling Grover's algorithm to handle larger and more complex graphs, such as those representing U.S. counties. As quantum hardware becomes more accessible and capable, these larger instances may become feasible to solve. Additional testing across other quantum programming languages like Cirq or Ocean could provide further insight into performance differences. Beyond graph coloring, this approach can be extended to other constraint satisfaction problems, including Sudoku, course scheduling, and resource allocation. Integrating hybrid quantum-classical methods could help bridge current hardware limitations.

IX. Materials

This project used Qiskit and Q# as the primary quantum programming environments to implement Grover's algorithm. All simulations were run on classical hardware using the AerSimulator for Qiskit and the Quantum Development Kit for Q#.

References

Adams, A. J., Khan, S., Young, J. S., & Conte, T. M. (2024, April 19). *QWERTY: A basis-oriented quantum programming language*. arXiv.org. https://arxiv.org/abs/2404.12603.

Brown, A. R. (2022, December 19). Playing Pool with |ψ>: from Bouncing Billiards to Quantum Search. https://arxiv.org/pdf/1912.02207.

Cornell University. (n.d.). Graph algorithms. https://www.cs.cornell.edu/courses/cs3110/2013sp/supplemental/recitations/rec21-graphs/rec21.html

Graph coloring. Graph Coloring - an overview | ScienceDirect Topics. (n.d.). https://www.sciencedirect.com/topics/computer-science/graph-coloring.

Grover's algorithm. Grover's algorithm | IBM Quantum Learning. (n.d.). https://learning.quantum.ibm.com/tutorial/grovers-algorithm.

Grover's algorithm and amplitude amplification. Grover's Algorithm and Amplitude Amplification - Qiskit Algorithms 0.3.0. (2024, April 10). https://qiskit-community.github.io/qiskit-algorithms/tutorials/06_grover.html

Maurice Clerc. A general quantum method to solve the graph K-colouring problem. 2023. https://hal.science/hal-02891847/document

Nathan Kjer. (2023, January 19). Quantum Computing: Map coloring via grover's algorithm. https://nathankjer.com/grovers-algorithm/

Appendix

I. **Code:**
   https://github.com/sjain2025/SysLab25
   A. **Qiskit:**
      https://github.com/sjain2025/SysLab25/tree/main/QiskitGrovers
   B. **Q#:**
      https://github.com/sjain2025/SysLab25/tree/main/QSharpGrovers
II. **Poster:**
   https://github.com/sjain2025/SysLab25/blob/main/Poster.pdf
III. **Presentation:**
   https://github.com/sjain2025/SysLab25/blob/main/Presentation.pdf
IV. **Google Drive:**
   https://drive.google.com/drive/u/3/folders/1gNiKUxtqkNOrHGueQXsFWkdaI2UIyzmF