Washington & Jefferson College
Math 330

Graph Coloring

Authors: Frances Sinclair & Cheryl Fergerson

April 25, 2019

Contents

1 Introduction

The issue of graph coloring applies to many different areas and studies, and as the graph grows, so doe the many different techniques and algorithms used to find solutions to this growing problem. Applications of graph coloring techniques are often used in time tabling, scheduling, radio frequency assignments, computer register allocations and much more[3]. Though all of the applications listed seem like they would only come up during a modern era with computers and large sets of data that need complex difficult solutions, there is also a large historical aspect to the concept of graph coloring, specifically coloring maps with neighboring countries using only four colors.

Though it may already be understood, it is important to mention that the goal of a graph coloring problem is to color the vertices a graph in such a way that no two adjacent vertices are the same color, and in such a way that the number of colors used in the graph is minimized. This paper will explore the history of the four color problem, algorithms for solving large coloring problems, and will include a closer look at three specific graph coloring algorithms.

2 History of Four-Color Problem

The history of the four color problem is a vital foundation for the concepts used in the coloring of graphs and for graph theory more generally. The four-color problem asks the question: "Can every map be colored with at most four colors in such a way that neighboring countries are colored differently?" At first glance, it seems like a very simple problem to solve, but it took about a century and contributions from many intelligent individuals to formulate a proof that would show that every possible map can be colored in this way.

The idea was first posed by Francis Guthrie, English lawyer, botanist, and mathematician. While coloring a map of England, he found that he could color the entire map only using four colors with the condition that no countries with common borders were colored with the same color, and he believed that he could apply this principle to any map. Francis Guthie's younger brother, Frederick Guthrie, proposed Francis's finding as a mathematical theory to his professor Augustus De Morgan in October 1852. De Morgan was extremely intrigued by the problem and wrote a letter to Sir William Rowan Hamilton the same day that he received the letter from Frederick Guthrie. Hamilton was not as fascinated by the four-color problem as De Morgan, and replied to him, stating his disinterest. The conjecture was made known to the mathematical world by De Morgan. Many thought that he was the original "inventor" of the four-color problem. Even though De Morgan always acknowledged Francis Guthrie as the discoverer of the four-color problem, Guthrie was not known as the originator by the mathematical world until Frederick Guthrie published his article, "Note on the Colouring of Maps" in 1880.

In 1853, De Morgan sent a letter about the conjecture to William Whewell, and one to Robert Leslie Ellis in 1854. In these letters, the main idea that he

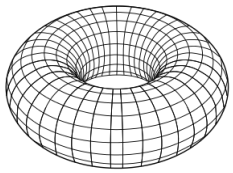
focused on was whether in a set of four countries where each pair of countries shared a different common border, one of the countries would have to be completely bordered by the other countries. In 1860, De Morgan wrote a review for Whewell's book *The Philosophy of Discovery* in the journal *Athenæum*, where he included an anonymous reference to the Four-Color Problem stated in the way that he described it in his letters to Ellis and Whewell as seen in the photo below.

Figure 1: A, B, C, and D represent different countries and colors.

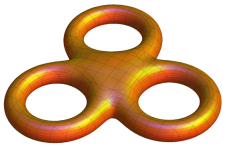
From 1860 to roughly 1880, the four-color problem disappeared from the mathematical radar. However, in 1878, Arthur Cayley inquired if anyone had proposed a solution to the problem in the mathematics section of the Royal Society, an English scientific academy. A year later, Alfred Bray Kempe published a full proof of the problem in an American, mathematics journal founded by James Joseph Sylvester. Because of the friendship between Cayley and Sylvester, Kempe published his proof through this unknown journal and brought the problem to attention in America. William Edward Story, the associate editor of the journal, made some additions to Kempe's proof that addressed special cases. When the proof was presented at a Scientific Association meeting at John Hopkins University in 1879, it included the additions provided by Story. Later, Charles (Santiago) Pierce, American philosopher, logician, and mathematician, added a few more supplements to Kempe's proof, and in 1879, the Four-Color Problem was believed to be solved.

In 1880, Scottish physicist, Peter Guthrie Tait published a "new" proof, but it only contained a few noteworthy changes to Kempe's proof, and Kempe retained credit for solving the problem. Many began to discredit the worth of the Four-Color Problem after Tait's publication, and rumors that the problem had already been examined in August Ferdinand Möbius' lectures in approximately 1840. However, this was untrue, but the rumors were rampant until 1959 when H.S.M. Coexter uncovered an error in this rumor.

After this buzz died down, the Four-Color Problem disappeared from the mathematical radar again until 1890 when Percy John Heawood found a mistake in Kempe's proof. Heawood then put out a proof for what is known as the Five-Color Theorem. Additionally, Heawood explored, as did Kempe, the problem in regards to spheres, the torus, the genus 3, and other surfaces.



(a) Torus



(b) Genus 3

After Heawood's revelation, no progress was made on the Four-Color Problem. William Thomas Tutte classified two methods of solving the problem: the quantitative approach, which followed Heawood's methodology, and the qualitative approach, which more aligned to Kempe's methodology. Heawood attempted to find a solution to the Four-Color Problem using number theory. Based off this work, Oswald Veblen presented the problem using linear equations covering a finite space in 1912. Veblen's colleague, George David Birkoff, along with Phillip Franklin let all of the topologists at Princeton take on the problem. Originally, Birkoff's work was based in Veblen's work, the quantitative approach. Later, he made noteworthy progress using Kempe's qualitative approach with his idea of reducible rings.

After this, little progress was made until the early 1960s when Henrich Heesch made a significant development. Heesch devised an algorithm to prove "D-reducibility", an idea that he came up with. With the help of computers, he was able to use the algorithm. He also constructed "unavoidable sets of configurations" in an attempt to get closer to a proof of the problem. With Karl Dürre's help, Heesch was able to computerize his algorithm. In 1965, the algorithm went through its first series of tests. Improvements in technology over the next two years made the algorithm able to apply to more complex cases. Because the United States had more advanced technology, Heesch went to the United States multiple times to advance his work. He wanted to solve the problem alone in Germany, however the resources he needed were not allocated to him because many researchers did not believe in the project. Heesch had an aptness for applying the mathematics to structures easily visualized and using observations from these structures to help solve the problem. His technique made it easier to present the many models needed, nearly 2000, for proving the Four-Color Theorem.

Jean Mayer, a French literature professor, contributed greatly to the problem, but his progress was overshadowed by the complete proof of the Four-Color Theorem. However, progress that he made working with Kenneth Appel and Wolfgang Haken made it into the complete proof of the Four-Color Theorem.

Heesch's trips to America can be said to represent a turning point in the history of the Four-Color Problem. It created extensive communication between Heesch and Haken. Through this communication, Heesch's work was brought to the United States and translated to English, as it had previously only been in German. This brought in a lot of contributions from the English-speaking world. Also, Kenneth Appel got involved with the problem because of Haken.

Together, they worked on a proof using the qualitative approach, which was thought to be nearly impossible. Appel and Haken worked together and relied on the use of computers to apply the algorithms they produced.

In 1975, Haken had a new idea for a method of solving the problem. This new method would be too complicated to modify for a computer, and they would return to using more hands on calculations. During this time, they also added an American graduate student, John Koch, to their team. He mainly worked on calculations that had been disregarded until then. In 1976, The complete proof of the Four-Color Theorem was finally attained with the aid of an IBM 360 computer system. The complete proof contained thousands of figures and errors were made. However, Koch, Haken, and Appel found an effective method for correcting errors that did not disrupt the conclusions of the proof. All of the errors found until 1989 have been corrected by Appel and Haken in an updated version of their proof. This proof has not terminated interest in the Four-Color Problem. Many are still trying to find proofs of the Theorem without the use of computers. This has created many questions that have yet to be answered, but maybe in another century, a proof without the use of computers may exist.

3 Algorithms

While looking at algorithms for solving Graph Coloring Problems, multiple definitions and ideas must be defined first, as well as how and why they work for solving these immense problems. After defining some terms, we will take a look at at 4 different strategies for finding approximate solutions to medium sized Graph Coloring Problems and then we will look at how those strategies combine together in order so solve large graph coloring problems. Lastly we will compare the different strategies and look closely at three specific algorithms and how they relate the the strategies described above. This section will not focus on the history aspect of discovering these algorithms such as who discovered it or worked on the problem or how it has developed; rather this section will focus on the theory behind the algorithms and how well the work in solving problems.

3.1 Defining terms and ideas in Graph Coloring Algorithms

It is important to begin this section by explaining that there are two different kinds of algorithms or solutions to graph coloring problems. There are exact and approximate algorithms that will result in exact or approximate solutions respectively. Though at first it may seem like a waste of time to go over approximate algorithms and what they entail since they are not truly solutions since a solution to a graph coloring problem uses as few colors as possible, they are vital and important pieces in understanding exact algorithms. Work on exact algorithms has only occurred recently (the work we refer to was published in 2014) whereas approximate algorithms have been around for much longer.

If two adjacent vertices a and b have the same coloring, they are called

conflicting. If that conflicting coloring is inside of the solution for the graph coloring problem, the solution is called illegal. A solution, also know as a k-coloring, with no conflicting vertices (aka without conflicting edges), is called legal. The variable $\chi(G)$ is used to denote the minimum number of colors, k, in the solution to the GCP (Graph Coloring Problem) and is referred to the chromatic number of the graph in such a way that $\chi(G)$ is a legal k-coloring. The color of a vertex is denoted c(x). A color class is the set of vertices that have the same value c(x). The union of all color classes is the graph G, essentially every vertex in G has some value c(x). f(c) describes the number of conflicts in an illegal k-coloring. In any subgraph of G, partitioned by the putting the different colored vertices in their respected partition, f(c) is the summation of the number of edges between vertices in the same graph for every partition of G.

3.2 Different Strategies to Solve Graph Coloring Problems

There are four main strategies used in solving GCP. These four strategies are Local Search strategies; a LS (local search) uses a set of solutions and uses a crossover operator to combine those different solutions and to find a solution for the entire graph. The LS has allowed for algorithms to greatly improve and to divide up infeasible GCP into manageable problems for a computer (source 6). The four different strategies are the legal strategy, k-fixed partial legal strategy, k-fixed penalty strategy, penalty strategy.

3.2.1 The Legal Strategy

In a legal strategy only legal colorings are considered. Kemp chain interchanges are used in this strategy as "a more efficient neighborhood structure for this search space". A Kemp Chain for two subgraphs V_i and V_j of G is a connected component in the subgraph of G induced by $V_i \cup V_j$. The Kemp interchange does the following: If there is a vertex $v \in V_i$ without loss of generality, and there is a triplet (v, i, j). The Kemp interchange for that triplet is cone by replacing V_i with $(V_i - K) \cup (K - V_i)$, and V_j with $(V_j - K) \cup (K - V_j)$, where K is the Kempe chain for V_i and V_j. Note that if the subgraph $V_i \cup V_j$ is a connected subgraph then the Kemp chain interchange is not performed since it would only swap the two color classes. (source 6) Remember that the goal of GCP is to minimize the number of colors used in the solution, thus the Kemp Chain interchange does nothing to minimize the number of colors used, but rather helps to minimize the number of illegal colorings in the k-colorings. Thus the Kemp chain minimizes f(c). The overall goal of the Legal Strategy is to target the smalled partition of vertices into one color and to reduce it's size, making the number of elements in that partition eventually equal to zero and thus being able to eliminate that color from the solution, and lowering the value of k.

3.2.2 The k-fixed Partial Legal Strategy

In this strategy the number of colors in the solution is fixed. Let the number of colors in the solution be a. Thus there are a+1 partitions at all times in this solution. 1 through a are the partition of vertices by their color and a+1 is the set of vertices not yet assigned a color. Note that in this strategy f(c) is always zero and thus the only way to get the partitioning of the vertices to legally fit into a subgraphs, the only option is to reduce the size of the partition of vertices not yet assigned a color. This is done by either normally adding they to one of the subgraphs 1 through a such that f(c) remains zero, or when that is no longer possible a move called an "i-swap" is performed. An i-swap is done by moving a vertex v in the partition a+1 to some other partition 1 through a and moving any vertices adjacent v, in its new partition, into the partition a+1. This eventually allows for the set a+1 to be diminished until it is an empty set (assuming that the graph G can be legally k-colored where k=a.

3.2.3 The k-fixed Penalty Strategy

In this strategy the number of colors used, k, is fixed and unchanging. However this strategy differs from the k-fixed partial legal strategy in the fact that there are only a partitions and f(c) does not equal zero until the solution is found. Essentially the objective to finding a solution to a GCP is to minimize f(c). This is done using 1-moves which merely change the color of a vertex if that vertex is part of a conflicting edge.

3.2.4 The Penalty Strategy

This strategy is essentially the same as the k-fixed penalty strategy, the main difference here is that in the 1-move the vertex in question could either be moved to a different subset, essentially changing its color, or it could be assigned a new color, not yet in use, creating an additional subset since k is not fixed. The goal of this strategy is to both decrease f(c), the number of conflicting edges, and to decrease the size of the smallest partition so that the partition is eventually empty and k is minimized.

3.3 A Closer Look at Different Algorithms

3.3.1 Tabucol

Tabucol is an algorithm introduced to the public in 1987 by Hertz and de Warra and has since then been improved, but remains a strong basis for many GCP solving algorithms today. The version of Tabucol that will be analyzed in this paper is the the an improved version of Tabucol presented in Galnier P and Hao J-K's paper on evolutionary algorithms for graph coloring presented in the journal of combinatorial optimizations published in 1999. Thus it is important to mention that Tabucol may have undergone more improvements in the past 19 years, however they will not be reviewed or mentioned any further in this

paper for the sake of length.

Tabucol begins by generating an initial random k-coloring; this random coloring usually contains many conflicting edges which are later taken care of in further steps of the algorithm. Tabucol then modifies the color of a vertex trying to decrease f(c) until a k-legal coloring is found. 1-moves are used in finding a solution to the GCP and the performance of the 1-move (v, i) can be found by taking the difference of conflicting edges between the solution c+(v,i) and the original solution. Essentially the effect of the 1-move, denoted by $\delta(v, i) = f(c + (v, i)) - f(c)$. If $\delta(v, i)$ is positive then the 1-move creates more errors and Tabucol does not perform the 1-move; alternatively if $\delta(v, i)$ is negative that means that the 1-move creates less conflicting edges then the original solution and Tabucol performs the 1-move. It is important to note that for some vertices $\delta(v, i) = 0$, it is easy for us to see Tabucol should not perform this 1-move due to the fact that it creates no change in $f(c)$ and wastes time. This could also create a vicious cycle in the algorithm in which Tabucol performs an endless amount of 1-moves that have no effect on the algorithm's progress towards a legal solution. This problem is solved by adding a clause in the algorithm which requires Tabucol to only deal with the vertices responsible for conflicting edges. Tabucol decides its next 1-move by deciding which iteration is the best in terms of $\delta(v, i)$. It is easy to understand that Tabucol stops when f(c)=0.

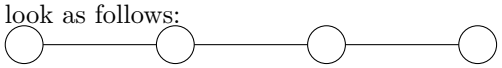
In most algorithms to find a solution to GCP there exists a thing know as a tabu list. This list contains some moves which the algorithm is not allowed to do. These are forbidden for reasons such as reversing the effects of moves performed recently or as mentioned before when $\delta(v, i) = 0$. This tabu list is essential in keeping the algorithm moving forward and promotes diversification in terms of which vertices have moves performed on they, or are considered for 1-moves.

3.3.2 cdclGCP (Conflict Driven Clause Learning Graph Coloring Problem)

As mentioned in the introduction to GCP solving algorithms there are both approximate and exact algorithms. The basis of this algorithm presented by Zhaoyang Zhou, Chu-Min Li, Chong Huang and Ruchu Xu and published in 2014 by Elsevier Ltd. The most unique part about their algorithm, which has not been mentioned in any other algorithms mentioned in this paper, is their take on implicit constraints. They explain that in graph coloring there are both explicit and implicit constraints among vertices. An example of an explicit constraint is as follows: if two vertices a and b are adjacent and a is yellow, then b cannot be yellow. However they delve in a lever further and determine that just because two vertices are not adjacent does not mean that there is no effect from the color of one vertex to the other. A simple example they used was as follows: a is adjacent to b, b is adjacent to c, c is adjacent to d. There are only those four vertices in G and only the three edges described. In the diagram below each circle represents one vertex a through d in order. The goal

of the algorithm in this example of to find a 2-coloring of G.

Let us consider the situation where the algorithm begins with a random coloring and colors vertex a and vertex d with the same color (as could feasible happen should an algorithm begin with a random coloring). The graph would look as follows:



As the algorithm fills in the rest of the vertices in the graph is must color vertex c and vertex d with different colors. Thus the colored graph would look something like this:

We can clearly see in the graph the 2-coloring is impossible from here forward because of how the algorithm began. However a 2-coloring of the original graph is clearly possible:

The question then becomes how to write an algorithm that can recognize situations like those described above; Z. Zhou et al. find a solution in what they call implicit constraints. An implicit constraint would be one such that when the vertex a is assigned a color, that color should be removed from the possible colorings of not only b, but also d so that the 2-coloring is possible.

The algorithm is able to learn implicit constrains like the one described above through a conflict driven clause learning algorithm. When a dead end is found in the process of finding a solution, such as the one above, the CDCL (conflict driven clause learning) then looks for and discovers an implicit constraint from the explicit constraints and (in some cases) previous implicit constraints. In order to do this CDCL uses three main steps:

1. Unit Propagation
   As the explicit and implicit constraints on a vertex increase the set of available colorings for that vertex decrease. When there is only one option in the set of available colorings, that vertex is then referred to as a unit vertex. If the set of available colorings is empty then v is referred to as an empty vertex and there is some error in the solution found thus far. When a vertex becomes a unit vertex, it is then used to create other unit clauses and other unit vertices. This is the process of unit propagation.

2. Analyze Conflict
   When an empty vertex is produced, an analysis of the error is done to find a clause representing the reason of the conflict, and the derived clause is then added to the database of implicit constraints.

3. Backtracking
   This procedure "backjumps to the second largest recursive level indicated in the derived clause" without undoing any of the anything. After backtracking the procedure then sets the derived clause to a unit clause, making

it so that the empty vertex is no longer empty and resolves the current conflict, and after that it branches out. The derived clause is added to the implicit constraints.

One issue with this algorithm is that all of the explicit and implicit constraints begin to take up a lot of memory and thus begin to slow the algorithm down. This problem is easily solved by deleting some of the memory, whether it be the oldest part of the memory, or the explicit and implicit constraints less relevant to certain subtrees, etc. Z. Zhou et al. admitted to having some difficulties determining what constraints to delete and which to keep after trying various methods and they drew no definite conclusion, due to the fact that that is not the goal of their paper[4].

3.3.3 MACOL

In 2009 Zhipeng L*ü* and Jin-Kao Hao presented their algorithm for graph coloring known as MACOL, which they describe as "a hybrid metaheuristic algorithm integrating a tabu search procedure with an evolutionary algorithm"[5] they also created an adapted multi-parent crossover operator (referred to from here on as AMPaX) to emphasize the importance of diversity of individuals and to get a good balance between intensification and diversification which finding a solution.

Their attempt to solve for the smallest k in a k-coloring graph is done by waiting until their algorithm finds a solution for a k-colored graph, then setting the new parameters to k-1 so their program then looks for a (k-1)-colored graph. If that is found, then it looks for a (k-2)-colored graph and thus the cycle continues until no k-colored graph can be found, for whatever k is at that point. The main scheme of MACOL is a combination of some procedures and algorithms covered earlier in this paper, as well as some new ones:

1. Initial Population Generator
   The individuals of the initial population are randomly generated using a version of the DANGER coloring heuristic by C. Glass, and the next vertex is also selected using a "dynamic vertex dangers measure"[5]. They will not be explored in this paper. The color assignment for these vertices is determined by a measure of which color is least likely to be needed by its adjacent vertices. Thus the color assignment is done using probabilities.

2. Tabu Search Procedure
   The Tabu Search Procedure is used as MACOL's LS procedure. It searches for conflicts and only stops when the best k-coloring is no longer improving. The number of iterations of the tabu search is know as the depth of the tabu search and is referred to by α.

3. Adaptive Multi-Parent Crossover Operator (AMPaX)
   The idea of a crossover is to transmit color classes from a parent to an offspring in such a way that the valuable information from a parent is passed down through the sharing of "ideas of grouping genetic algorithm"[5]. The

AMPaX operator is designed to be an extention of the GPX crossover which is based on exactly two parents. Thus AMPaX adaptively chooses parents two or more parents to produce an offspring. This increases the probability that the offspring will be of legal coloring. An interesting aspect of AMPaX is that after a certain number of steps, a parent can no longer pass down its information to an offspring. This is to avoid having one vertex determine the entire coloring of a graph and to promote diversification.

4. Population Updating Rule
   When updating a population, or the set of colored vertices of a graph MACOL analyzes what the creators call a "goodness score". When a vertex is defined, using probability (thus it is not exact as seen in cdclGCP and can be changed), it's goodness score is determined and if it's goodness score is not good enough or has the same score and the same number of uncolored vertices then it is not added to the set of vertices with defined colors, and thus is not part of the solution.

Using all of these things MACOL is able to create a solution to the GCP.

4 Conclusion

In conclusion, graph coloring is an complex problem that is one of the easiest to visualize. The Four-Color Problem is one of the many problems based in graph coloring. It is one of the foundational problems in graph theory and took over a century to prove. From map coloring to scheduling, graph coloring can be used to make life simpler, even if the proofs for these theorems and the algorithms for for solving these problems are anything but.

The Four-Color Problem is difficult in and of itself, creating a challenge that has probed the great mathematic minds of the world, and yet it still creates a challenge for current minds and programmers. This problem will grow in complexity as we find new and differing solutions and algorithms for problems that were once seen as impossible.

5 References

1. FRITSCH, Rudolf, and Gerda FRITSCH. The Four-Color Theorem: History, Topological Foundations and Idea of Proof. Translated by Julie PESCHKE, Springer, 1998.

2. Wilson, Robin James, and Ian Stewart. Four Colors Suffice: How the Map Problem Was Solved. Princeton University Press, 2014.

3. Galinier, Philippe, and Alain Hertz. "A Survey of Local Search Methods for Graph Coloring." Computers and Operations Research, vol. 33, no. 9, 2006, pp. 2547-2562.

4. Zhou, ZY, et al. "An Exact Algorithm with Learning for the Graph Coloring Problem." Computers & Operations Research, vol. 51, 2014, pp. 282-301.

5. Lü, Zhipeng, and Jin-Kao Hao. "A Memetic Algorithm for Graph Coloring." European Journal of Operational Research, vol. 203, no. 1, 2010;2009;, pp. 241-250.