Vertex Coloring and Applications

Shawn Seymour

Directed Study (MATH 3993)
University of Minnesota Morris

May 5, 2017

1 Introduction

Consider the map of the 48 contiguous states in the USA. Suppose we would like to color each state such that no two states that share a boundary have the same color. How many colors would it take to do this? This problem can be modeled as a *graph*. In general, we could represent every state with a *vertex* and draw an *edge* between two states that share a border. A graph, denoted $G = (V, E)$, is a set of vertices V and a set of edges E. A *simple graph* is a loopless, undirected graph where no two edges connect the same pair of vertices. For our purpose, assume all graphs are simple graphs. This problem, along with many others, can be solved using vertex coloring. We can model the map of the 48 contiguous states as a graph. We can represent every state with a vertex and draw an edge between two states that share a border.

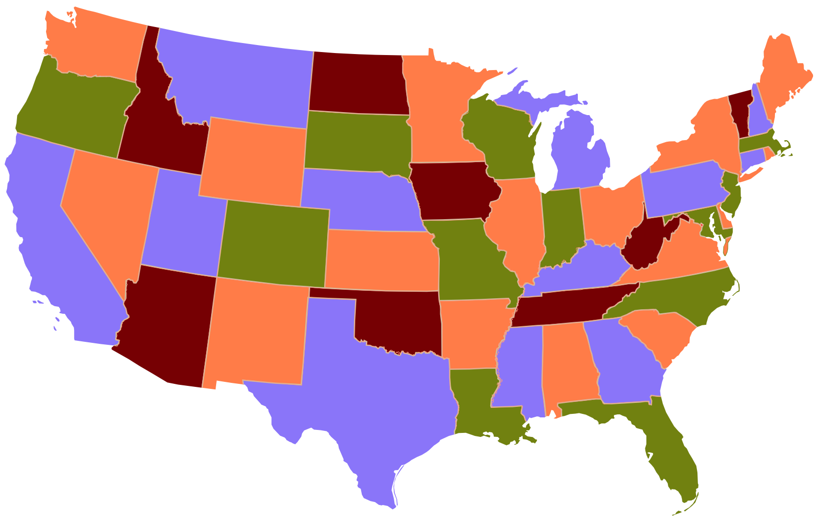


Figure 1: A proper vertex coloring of the 48 contiguous states of the USA

A *vertex coloring* of a graph G is an assignment of colors to each vertex of a graph. A *proper vertex coloring* assigns colors to a graph G such that no two adjacent vertices share the same color. This can be described as a function $f : V \to S = \{1, 2, \ldots, k\}$ such that $\forall u, w \in V$, if $(u, w) \in E$, then $f(u) \neq f(w)$. Note that this constraint is the same constraint we applied to our map example. This means we could color our map according to our constraint with a proper vertex coloring. For our purpose, assume we mean proper vertex coloring when referring coloring a graph.

The *chromatic number*, denoted $\chi(G)$, is the minimum number of colors needed to have a proper vertex coloring of a graph G. The *vertex coloring problem* (VCP) is to find $\chi(G)$ of any given graph G. By applying the vertex coloring problem to our map example, we can determine how many colors one would need to color it with regards to our constraint. An example coloring of our map is shown in Figure 1. The *k-Colorability problem* asks if a graph can be colored with k colors. If a graph can be colored with k colors, it is said to be *k-colorable*.

2 Computational Complexity

Finding an optimal solution to the vertex coloring problem is not easy. As the VCP is an optimization problem with regards to a constraint, we know it will require heavy computation to find an optimal solution. In mathematics and computer science, we classify computationally complex problems based on type of problem they are and based on how many steps it takes to solve the problem with respect to its input size.

In this section, we will first discuss how to classify hard mathematical problems. Then, we will classify the VCP as a mathematically hard problem and show how this classification can be done.

2.1 Classifying the Vertex Coloring Problem

We call any problem that can be answered with "yes" or "no" a *decision problem.* To classify the vertex coloring problem, we must first introduce some classes, defined by Moret [11]:

- **P**: Given a decision problem, we can solve the problem in polynomial time.
- **NP**: Given a decision problem, we can verify if a given solution is correct in polynomial time but cannot solve the problem in polynomial time.
- **NP-hard**: Given any problem, it is at least as hard as NP problems. They do not have to be in NP, however, as they do not need to have solutions verifiable in polynomial time.
- **NP-complete**: Given a decision problem, we can transform it to any other NP problem in polynomial time and still verify a given solution in polynomial time.

Take note that these classes are not exclusive: any member of P is also a member of NP. NP-complete problems can be thought of as the problems that are both in NP and NP-hard. Although it may seem like all problems that are NP are also NP-complete, this is not the case. There are a select few problems in NP that are not NP-complete or in P. See Ladner [9].

Proposition 1. *The vertex coloring problem is an NP-hard problem.*

It is NP-hard to compute the chromatic number of a graph. This can be proven by showing it is at least as hard as a problem in NP. This is typically done by a reduction (a transformation) to or from 3-SAT, which we'll examine later. This was proven by Garey et al. [5].

Proposition 2. *The k-Colorability problem, i.e. can a given graph be colored in k-colors, is NP-complete.*

To prove this, we need to reduce a known NP-complete problem to the vertex coloring problem in polynomial time. By reducing a known NP-complete problem to k-Colorability, we can conclude that k-Colorability is NP-complete as all NP-complete problems are reducible to all other NP-complete problems. See Garey and Johnson [4]. We will reduce the well-known 3-SAT problem first to the 3-Colorability problem. The 3-Colorability problem asks if a graph can be colored with 3 (or fewer) colors. Obviously, if a graph can be colored with 2 colors it can also be colored with 3. From here, the reduction from 3-Colorability to k-Colorability follows.

2.2 Reduction from 3-SAT to 3-Colorability

Before we prove Proposition 2, we must first introduce 3-SAT. The 3-SAT problem is a decision problem that is a variation of the original NP-complete problem, Boolean satisfiability (SAT). The 3-SAT problem was proven to be NP-complete. See Garey and Johnson [4]. SAT consists of a conjunction of m clauses, i.e. $C_1 \wedge C_2 \wedge \cdots \wedge C_m$. Each clause is a disjunction of at most n literals, i.e. $L_1 \vee L_2 \vee \cdots \vee L_n$. Each literal can be a variable or negative of a variable, i.e. $x_1, \neg x_1, x_2, \neg x_2, \ldots, x_k, \neg x_k$ where k is the number of distinct literals. Each literal has a value of $true$ or $false$. 3-SAT has exactly 3 literals in each clause ($n = 3$). We will assume all of our satisfiability problems are in conjunctive normal form (CNF). 3-SAT asks whether the variables can be assigned *true* or *false* in such a way that the whole expression evaluates to *true*. We define SAT and 3-SAT in Definition 1.

Definition 1. SAT is a decision problem that decides the truth value of a Boolean function based on the evaluation of the clause. It determines whether there are truth values to each clause in a set of clauses that will return a truth value to the given Boolean function. 3-SAT is a variation of SAT where each clause contains exactly 3 literals.

To show a reduction from 3-SAT to 3-Colorability, we need to:

1. Construct an instance of 3-Colorability, given an instance of 3-SAT
2. Show that if the resulting graph is 3-colorable, then the given 3-SAT problem is satisfied

To show this reduction, we'll create an example 3-SAT problem and reduce it to 3-Colorability. This reduction will create a graph $G = (V, E)$ having $(2n + 3m + 1)$ vertices and $(3n + 6m)$ edges. See Moret [11]. The process for creating the graph, described below, was taken from the proof given by Sharma and Chaudhari [12].

Let's introduce the following 3-SAT expression E:

$$E = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

To create the graph of this 3-SAT problem, we:

1. Create a *triangle* for each variable, i.e. x_i. Each has a common vertex B, called the base vertex. This base vertex will be colored a color, say p_1, so that the other two vertices correspond to the variable and it's negation.
2. Create a *triangle* for each clause in the formula.
3. Connect each vertex of a clause triangle to the corresponding literal vertex.

Following the above steps, we generate graph G, shown in Figure 2, from our 3-SAT expression E.

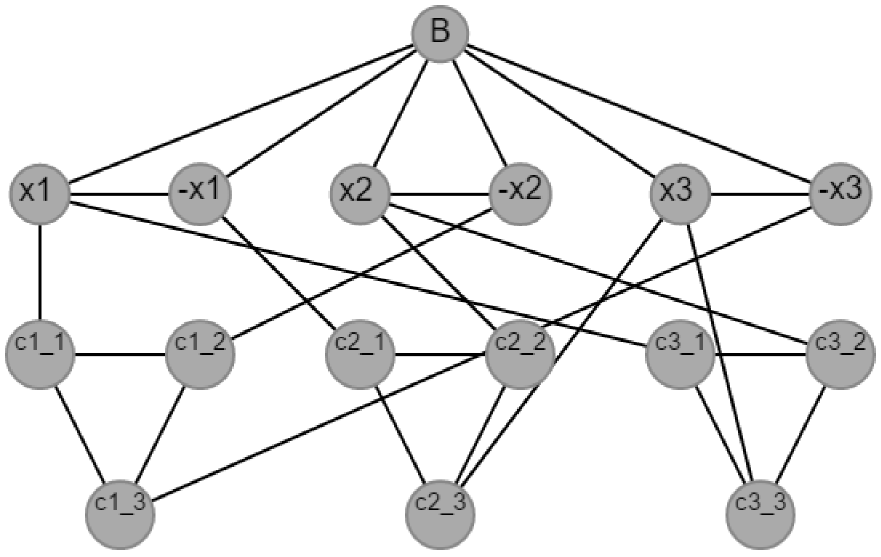


Figure 2: E represented as a graph G

If we can properly color graph G with 3 colors, then the original 3-SAT is satisfiable and the truth values of the variables are denoted by the color they received. This graph is *3-colorable* and the coloring is shown in Figure 3.

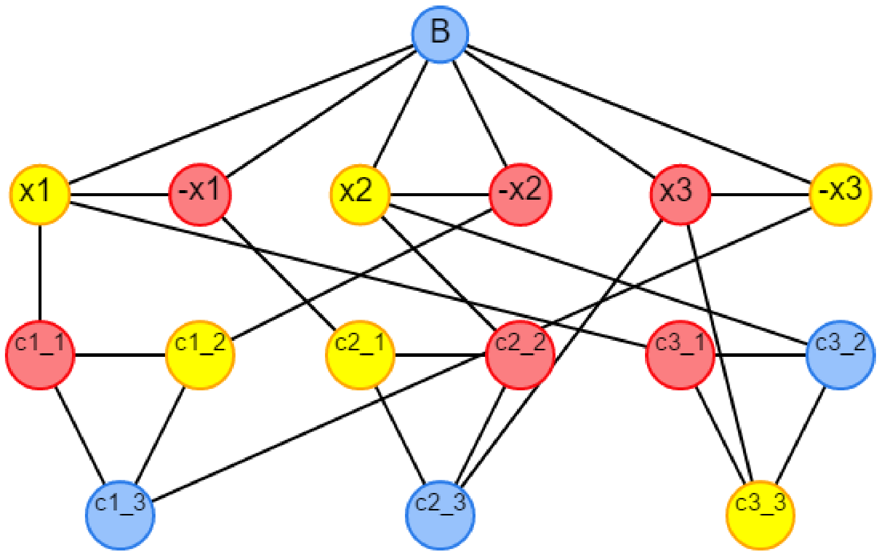


Figure 3: A possible 3-coloring of G

As we see above, *blue* is used for the base vertex and as part of some of the clause triangles. If we let *yellow* denote *true* and *red* denote *false*, we can see that our original 3-SAT is satisfied:

$$\begin{aligned} E &= (T \vee \neg T \vee \neg F) \wedge (\neg T \vee T \vee F) \wedge (T \vee T \vee F) \\ &= (T \vee F \vee T) \wedge (F \vee T \vee F) \wedge (T \vee T \vee F) \\ &= T \wedge T \wedge T \\ &= True \end{aligned}$$

This shows an example of the reduction from 3-SAT to 3-Colorability. If G was unable to be colored with 3 colors, then we would know that E is unable to be satisfied. From here, we can reduce 3-Colorability to 4-Colorability, and then generalize that reduction and reduce 4-Colorability to k-Colorability. The proof and illustration of this can be found in Sharma and Chaudhari [12].

3 Heuristics

As we have seen in section 2, the vertex coloring problem is NP-hard and thus solving it to optimality for any given instance of a graph is computationally "impossible". This gives us incentive to use approximation algorithms, called heuristics, to find good, but not necessarily optimal, vertex colorings. We will analyze and compare a few of the popular heuristics for solving the VCP. However, in this section, we will describe the four heuristics to solve the VCP. The input for all heuristics we define is a simple graph, $G = (V, E)$.

Heuristic A: Greedy

Heuristic A is a simple greedy algorithm. It is defined in Algorithm 1.

Algorithm 1 Greedy algorithm

```
1: Label each vertex in $V$, i.e. $v_1, v_2, \dots, v_n$
2: for each $v \in V$ do
3:     Assign a color $p_i$ to $v_i$ using the smallest available $p_i$
```

Heuristic B: Welsh-Powell

Heuristic B is the greedy algorithm with degree sequencing. Before applying the greedy algorithm, it orders the vertices according to the decreasing value of their degree. This is also known as the Welsh-Powell algorithm, which was defined by Welsh and Powell [14]. It is defined in Algorithm 2.

Algorithm 2 Welsh-Powell algorithm

```
1: Label each vertex in $V$, i.e. $v_1, v_2, \dots, v_n$, such that $d_G(v_1) \geq d_G(v_2) \geq \cdots \geq d_G(v_n)$
2: for all $v \in V$ do
3:     Assign a color $p_i$ to $v_i$ using the smallest available $p_i$
```

Heuristic C: DSATUR

Heuristic C is very similar to heuristic B, but colors a graoh based on the *saturation* degree of vertices, which is defined in Definition 2 [13]. The original DSATUR algorithm was proposed by Brélaz [2]. It is defined in Algorithm 3.

Definition 2 (Saturation degree)**.** The saturation degree of a vertex is the number of different colors used for vertices adjacency to it.

Algorithm 3 DSATUR algorithm

```
1: Find the vertex $\in V$ with maximum degree and color it with color $p_1$
2: while There are uncolored vertices do
3:     Choose an uncolored vertex with maximum saturation degree
4:     If there is a tie, choose the vertex with maximum degree
5:     Color the vertex with the least possible number
```

Heuristic D: Maximal Independent Set

Heuristic D colors a graph by finding maximal independent sets of G. A maximal independent set (MIS) is defined in Definition 3. This is quite different than our other heuristics as it does not depend on degree. Ideally, we would want to find a maximum independent set, not just a maximal independent set.

Finding a maximum independent set is NP-hard as detailed by Karp [7]. This means we're including a different NP-hard in an already NP-hard problem. We chose to forgo this in favor of just finding a maximal independent set. We define the heuristic in Algorithm 4.

Definition 3 (Maximal independent set)**.** An independent set is a subset S of vertices in a graph such that no two vertices in the set are adjacent. S is said to be maximal if no vertex outside the set can be added while still being an independent set.

Algorithm 4 Maximal independent set algorithm

```
1: $p \leftarrow 1$
2: while $G$ is non-empty do
3:     Find a maximal independent set of $G$, i.e. $S_i$
4:     Color all vertices in $S_i$ with color $p$
5:     Let $G \leftarrow G[V \setminus S_i]$
6:     $p \leftarrow p + 1$
```

4 Analysis of Heuristics A - D

Before comparing the heuristics, we first wanted to analyze the heuristics individually and look at the solutions they produce.

Proposition 3. *Heuristic A and Heuristic B produce a feasible, proper coloring of input graph $G = (V, E)$.*

Let $G = (V, E)$ be a simple graph. Assume that two vertices, say $v_1, v_2 \in V$, are connected by an edge $e_1 \in E$. Assume that v_1, v_2 are colored the same color, say p_1. We know that in both heuristics, we assign the smallest color p_i that is not being used by any of the neighboring vertices connected by an edge e_i. WLOG, the heuristic would color v_1 color p_1. The next iteration, while vertex v_2 is selected, the heuristic would see color p_1 is assigned to a neighboring vertex (v_1), and assign the next available color p_2. Thus, we've reached a contradiction.

Proposition 4. *Heuristic C produces a feasible, proper coloring of input graph $G = (V, E)$.*

Let $G = (V, E)$ be a simple graph. We color the vertex with the maximum degree color 1. If there are no uncolored vertices, we have properly colored G. If there are, then we find an uncolored vertex v with maximum saturation degree. We then color v with the smallest available color. This means we look at the set of all adjacent vertices to v and find the smallest unused color in this set. This means we will never color an adjacent vertex with the same color, and thus, will always result in a proper coloring.

Proposition 5. *Heuristic D produces a feasible, proper coloring of input graph $G = (V, E)$.*

Let $G = (V, E)$ be a simple graph. Let p, the current color, be 1. By definition 3, we know that no two vertices in a maximal independent set are adjacent. Thus, when we color the vertices in the MIS with the smallest color p_1 for the first iteration, we know that it is properly colored. We then remove the found maximal independent set and increment p to 2. If G is now empty, then G is properly colored. If G is not empty, then we repeat the process except we use color 2. Because we are always incrementing the color for each iteration, we know the MIS found will always be properly colored with a different color than the past MIS.

Now that we know our heuristics produce feasible, proper colorings of a graph, how do we know they do not produce optimal solutions? We show they do not using counterexamples.

Proposition 6. *Heuristic A and Heuristic B do not produce optimal solutions, i.e. they do not always produce a minimum coloring for all graphs.*

We will show this for *heuristic B*. It is trivial to show the same for *heuristic A* as *heuristic B* includes the same steps as *heuristic A*. We will construct a simple graph, $G = (V, E)$, such that:

$$|V| \geq 8 \quad (1)$$
$$\chi(G) > \chi^*(G) \quad (2)$$

Let $\chi(G)$ refer to the coloring of G generated by *heuristic B*. Let $\chi^*(G)$ be the optimal coloring of G. We constructed a simple graph, $G_1 = (V, E)$, such that $\Delta(G_1) = 3$ and $\delta(G_1) = 1$.

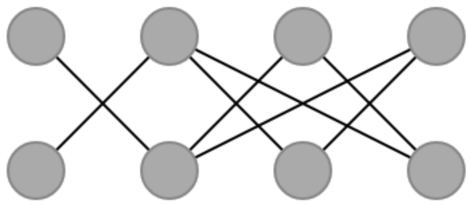
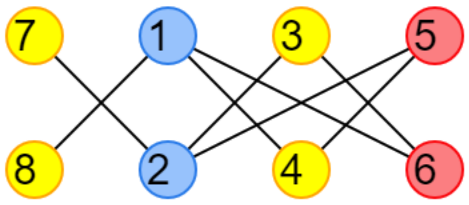


Figure 4: Uncolored original graph G_1

By applying *heuristic B*, we get the following coloring. The numbers indicate the order the vertices were colored in while applying the heuristic. Any vertex of the same degree got assigned arbitrarily. This results in $\chi(G_1) = 3$.



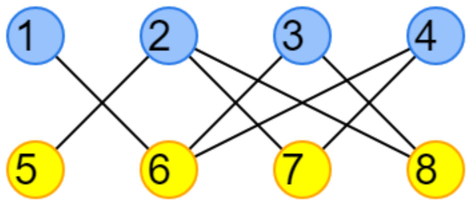


Figure 5: Coloring of G_1 after applying heuristic B

Figure 6: Optimal coloring of graph G_1

Definition 4 (Bipartite graph)**.** A bipartite graph is one whose vertex set can be partitioned into two subsets X and Y such that each edge has one end in X and one end in Y.

We can see that G_1 is a bipartite graph. Thus, G_1 is 2-colorable as all bipartite graphs are 2-colorable. See Asratian et al. [1]. This means $\chi^*(G_1) = 2$. G_1 is an example graph that satisfies our given conditions (1) and (2). The optimal coloring is shown in Figure 6.

Although Heuristic A and Heuristic B do not always produce an optimal coloring for bipartite graphs, Heuristic C always will produce an optimal coloring (2 colors). Heuristic C, however, does not produce optimal colorings for all graphs. Both of these results are shown and proven by Brélaz [2].

Proposition 7. *Heuristic C does not produce an optimal solution, i.e. it does not always produce a minimum coloring for all graphs.*

To show this, we've created a graph $G_2 = (V, E)$ for this example. This is shown in.

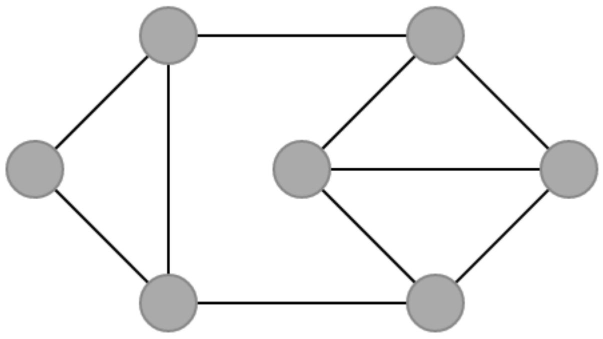


Figure 7: Uncolored original graph G_2

By applying heuristic C, we can see that G_2 is *4-colorable*. This is shown in Figure 8.

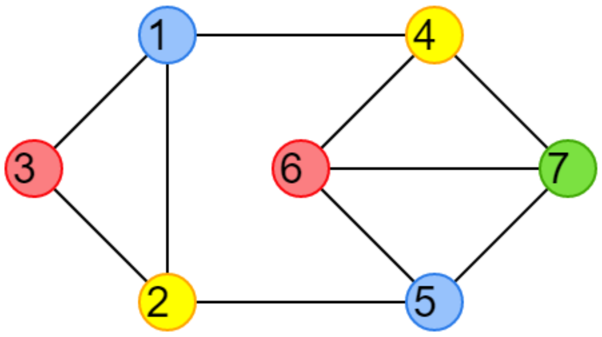


Figure 8: Coloring of G_2 after applying heuristic C

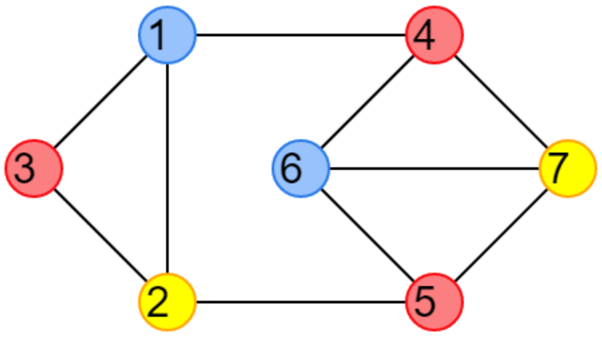


Figure 9: Optimal coloring of G_2

This is a non-optimal solution as this graph is *3-colorable*. This means $\chi^*(G_2) = 3$. Thus, we can see that heuristic C does not always give us an optimal solution. The optimal coloring of G_2 is shown in Figure 9.

Proposition 8. *Heuristic D does not produce an optimal solution, i.e. it does not always produce a minimum coloring for all graphs.*

To show this, we will create another example that satisfies conditions (1) and (2) from earlier. We've constructed a graph $G_3 = (V, E)$ for this example. This is shown Figure 10.

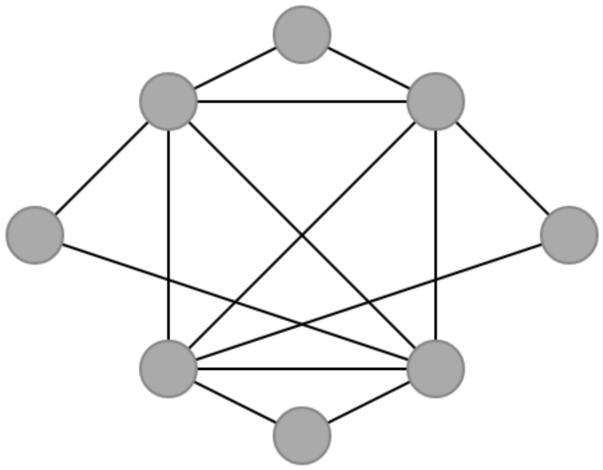


Figure 10: Uncolored original graph G_3

By applying heuristic D and finding maximal independent sets, we see that G_3 is *5-colorable*. This is shown Figure 11.

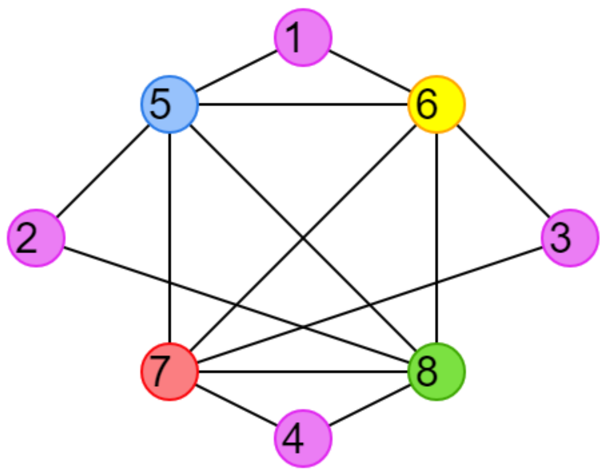


Figure 11: Coloring of G_3 after applying heuristic D

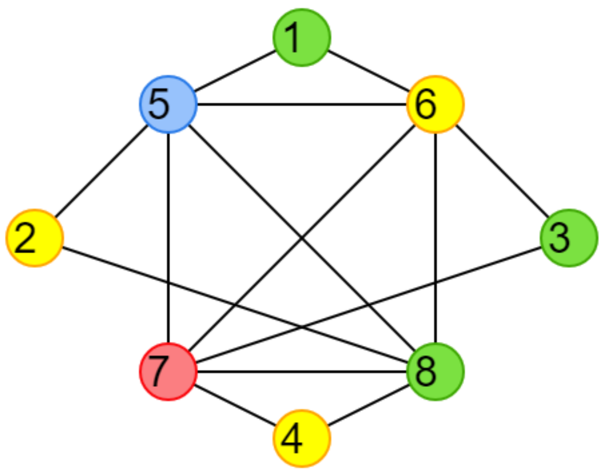


Figure 12: Optimal coloring of G_3

This is a non-optimal solution. We can see that the subgraph created by vertices $\{5, 6, 7, 8\}$ is complete and thus we need at least 4 colors. This graph is *4-colorable* however, meaning $\chi^*(G_3) = 4$. Thus, we can see that heuristic D does not always give us an optimal solution. The optimal coloring of G_3 is shown in Figure 12.

Proposition 9. *Heuristic A and heuristic B yield an upper bound of* $\Delta + 1$.

To show this, we will create another example that satisfies conditions (1) and (2) from above. After doing some research into bipartite graphs, we learned that *crown graphs* are excellent at showing how bad greedy heuristics can be, as shown and defined by Kordecki and Łyczkowska-Hanćkowiak [8].

Definition 5 (Crown Graph)**.** A crown graph $CR_n = (V, E)$ is an undirected graph such that $U = \{u_1, u_2, \ldots, u_i\}$ and $W = \{w_1, w_2, \ldots, w_j\}$ where $V = U \cup W$, $U \cap W = \emptyset$, and $|U| = |W|$. There is an edge from u_i to w_j whenever $i \neq j$. A crown graph can also be viewed as a complete bipartite graph where the edges of a perfect matching have been removed.

We constructed a simple crown graph, $G_3 = (V, E)$. We've let $|V| = 10$ for this example.

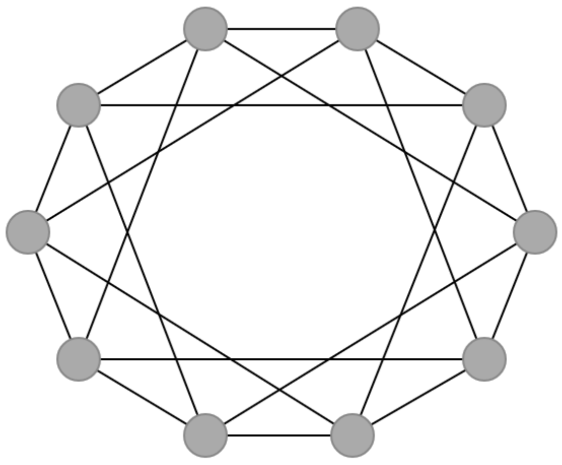


Figure 13: Uncolored original graph G_4

We can see that $\Delta(G_4) = 4$. We can also see $d_{G_4}(v) = 4$ for all $v \in V$. Thus, in both heuristics A and B, the algorithm would pick an order arbitrarily. We can show using this crown graph the worst-case scenario of these heuristics.

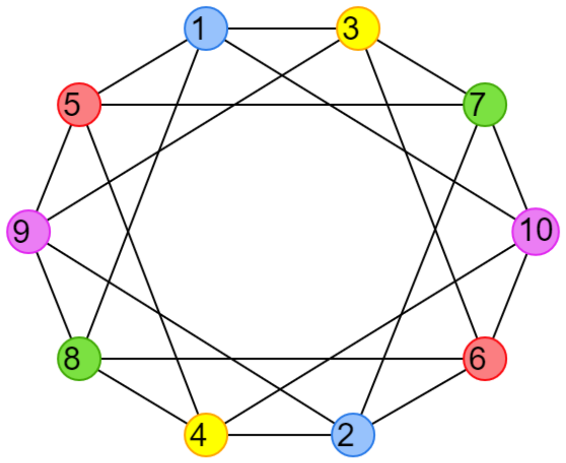


Figure 14: Worst-case coloring of G_4

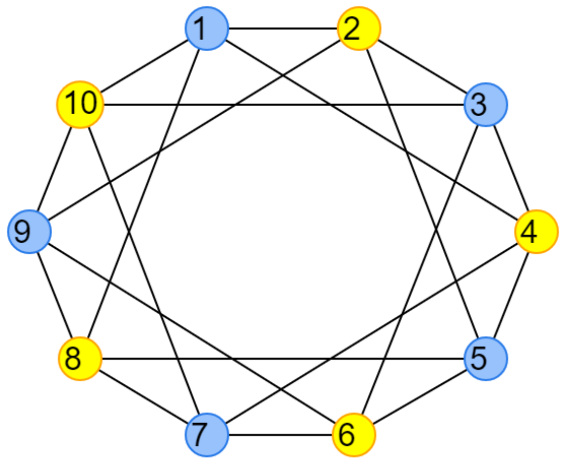


Figure 15: Optimal coloring of G_4

As shown in Figure 14, using either heuristic with this ordering, we get $\chi(G_4) = 5$. This gives us $\frac{|V|}{2}$ colors. This is the worst-case for a crown graph. See Johnson [6]. This graph also demonstrates the upper bounds for these heuristics.

Let's take a look at how this looks in general. Let $P = (V, E)$ be a simple, complete graph. We can see that $\chi(H) = \Delta(H) + 1$. This is very easy to see in a *complete* graph, where all vertices are connected to every other vertex. This means all vertices have degree $|V| - 1$. Thus, every time we color a node, a new color is needed. And since we have $\Delta(P) = |V| - 1$ and $|V|$ vertices, we will need $\Delta(P) + 1$ colors. This is stated in *Brooks' Theorem*.

Theorem 1 (Brooks' Theorem)**.** *For any connected undirected graph G with maximum degree Δ, the chromatic number of G is at most Δ unless G is a complete graph or an odd cycle, in which case the chromatic number is $\Delta + 1$.*

The proof of Brook's Theorem is shown by Lovász [10]. Overall, heuristics can produce undesirable results compared to the optimal solution. In graph G_4, in the worst case, some heuristics produce $\chi(G_4) = 5$ when $\chi^*(G_4) = 2$ as it is a bipartite graph. This is shown in Figure 15.

5 Results

Our main goal was to compare these heuristics to find out which heuristic was most effective in efficiently approximating the vertex coloring problem. we wanted to compare both quality (number of colors used) and efficiency (running time). For this study, we decided to omit the maximal independent set heuristic (heuristic D) as it was so different from the other 3 as well as including another NP-hard problem. We implemented the three heuristics in Java and ran simulations according to the following:

- Graphs were randomly generated based on the *Erdős–Rényi* model, given in Erdős and Rényi [3]. This model generates a random graph with approximately p percent of edges.
- We analyzed graphs with $10\%, 20\%, \ldots, 90\%$ edges. We called this the type of the graph. This gave us both sparse and dense graphs.
- Simulations were run on graphs with 5 different amount of vertices: 50, 100, 250, 500, 1000.

We used statistical analysis to compare the number of colors used and the running time of each heuristic based on edge percentage (type) and then also looked at how this changed as the number of vertices changed.

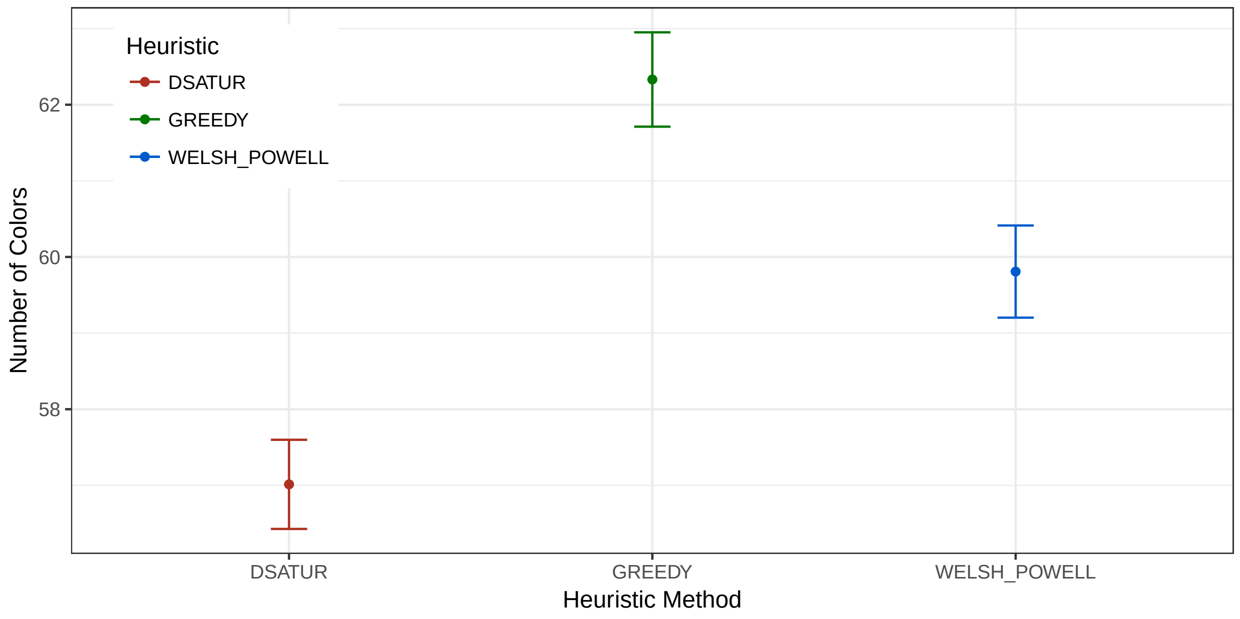


Figure 16: Number of colors used for the three heuristics of graphs with 50, 100, 250, 500, & 1000 vertices

Based on computational data of the amount of colors used, as shown in Figure 16, we found:

- Heuristic A (Greedy) uses on average 2.52 colors more than Heuristic B (Welsh-Powell).
- Heuristic B (Welsh-Powell) uses on average 2.79 colors more than Heuristic C (DSATUR).

This led us to the following partial order based on colors used, where $X \preceq Y$ means heuristic X uses significantly more colors than heuristic Y.

$$\textbf{Greedy} \preceq \textbf{Welsh-Powell} \preceq \textbf{DSATUR}$$

As vertices increased, the partial order defined above stays consistent. Heuristic C (DSATUR) always used the least colors while Heuristic A (Greedy) always used the most.

Based on computational data of the running time, as shown in Figure 17, we found:

- Welsh-Powell is significantly slower than Greedy.
- DSATUR is significantly slower than Welsh-Powell.

This led us to the following partial order based on running time, where $X \preceq Y$ means heuristic X uses significantly less time than heuristic Y.

$$\textbf{Greedy} \preceq \textbf{Welsh-Powell} \preceq \textbf{DSATUR}$$

As the edge percentage increases:

- Heuristic C (DSATUR) running time increases.
- Heuristic A (Greedy) and Heuristic B (Welsh-Powell) running time increases until 50% edges then decreases until 90% edges.
- The difference in running time between DSATUR and Greedy as well as DSATUR and Welsh-Powell grows larger and larger.

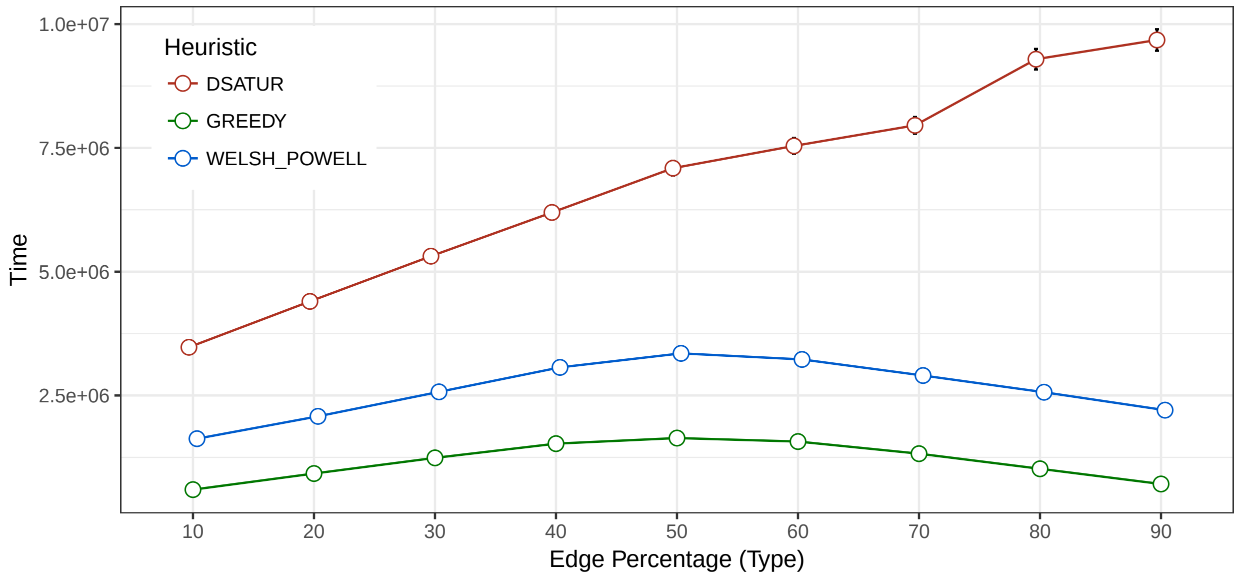


Figure 17: Running time for each edge percentage of graphs with 50, 100, 250, 500, & 1000 vertices

6 Conclusion

Although we were not able to compare the coloring of the heuristics to the optimal coloring, we were able to determine which heuristics gave the closest to the optimal solution. We found that Heuristic C (DSATUR) produced the lowest number of colors. However, DSATUR also utilized the most time especially as the edge percentage of the graph grew. This is undesirable, because in large graphs, this defeats our purpose of using heuristics to efficiently approximate the VCP.

Based on our overall results, we would suggest using Heuristic B (Welsh-Powell) for most graphs. This is because the extra time needed from Heuristic A (Greedy) to Heuristic B (Welsh-Powell) to achieve better accuracy is not as significant compared to the extra time needed from Heuristic B (Welsh-Powell) to Heuristic C (DSATUR).

7 References

[1] A. S. Asratian, T. M. Denley, and R. Häggkvist. *Bipartite graphs and their applications*, volume 131. Cambridge University Press, 1998.

[2] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[3] P. Erdős and A. Rényi. On random graphs i. *Publ. Math. Debrecen*, 6:290–297, 1959.

[4] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[5] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical computer science*, 1(3):237–267, 1976.

[6] D. S. Johnson. Worst case behavior of graph coloring algorithms. In *Proc. 5th SE Conf. on Combinatorics, Graph Theory and Computing*, pages 513–528, 1974.

[7] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[8] W. Kordecki and A. Łyczkowska-Hanćkowiak. Greedy online colouring with buffering. *arXiv preprint arXiv:1601.00252*, 2016.

[9] R. E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM (JACM)*, 22(1): 155–171, 1975.

[10] L. Lovász. Three short proofs in graph theory. *Journal of Combinatorial Theory, Series B*, 19(3): 269–271, 1975.

[11] B. M. Moret. The theory of computation. Technical report, Addison-Wesley, Reading, Mass., 1998.

[12] P. C. Sharma and N. S. Chaudhari. A new reduction from 3-sat to graph k-colorability for frequency assignment problem. *Int. J. Comp. Applic*, pages 23–27, 2012.

[13] J. P. Spinrad and G. Vijayan. Worst case analysis of a graph coloring algorithm. *Discrete Applied Mathematics*, 12(1):89–92, 1985.

[14] D. J. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.