Graph Coloring - NP Hard Problem

Neeraj Bhatt (30), Vivek Rai (58)

October 28, 2025

1 Problem Statement

We will write a program to implement the graph coloring problem and compare three different approaches: 1. brute force, 2. Greedy, 3. A heuristic, in terms of cost and time, on data of various sizes and plot graph.

Definition 1 (Graph Coloring Problem)**.** *Given an undirected graph, G=(V,E) and a positive integer k, the graph coloring problem asks whether it is possible to assign one of k different colors to each vertex in V such that no two adjacent vertices (i.e. vertices connected by an edge in E) share the same color.*

Graph coloring problem Figure:

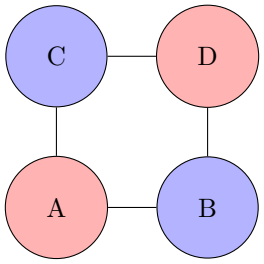

Figure 1: Graph Coloring Example (Chromatic Number = 2)

In this graph, we used two colors: Red and Blue. The goal is to assign colors to the vertices in such a way that no two connected vertices (i.e., vertices with an edge between them) share the same color. In this example, the chromatic number is **2**, as only two colors are needed to color the graph.

The graph coloring problem is NP-hard and the proof of the graph coloring problem being NP-hard has been taken from [3]

2 Brute Force Algorithm (BF)

We want the chromatic number $\chi(G)$, i.e., the minimum number of colors needed to color a graph G. For each value of k = 1, 2, 3,..., n (where n is the number

of vertices), we will check for all different assignments of k colors if it is possible to color the graph so that no adjacent vertices have the same color.

Algorithm 1 Brute-force Graph Coloring

```
1: function ChromaticNumber(G)
2:     for k = 1 to |V(G)| do
3:         for all colorings C with k colors do
4:             if no edge (u, v) has C[u] = C[v] then
5:                 return k
6:             end if
7:         end for
8:     end for
9: end function
```

Time Complexity : $O(k^n)$,where k is the number of colors and n is the nodes.

3 Greedy Algorithm

A simple greedy algorithm colors the vertices of a graph one by one, assigning the smallest available color that has not been used by it's neighbors. We can always color graph G with $\Delta + 1$ colors, where Δ is the maximum degree of a graph by a simple greedy algorithm, $\chi(G) \leq \Delta(G) + 1$ [2].

4 Heuristic Algorithm (Heu.)

For heuristic approach we have well known strategies like vertex-ordering by saturation degree (DSATUR) to produce solutions that are closer to optimal. The DSATUR algorithm for graph coloring was introduced by Brélaz [1].
Time Complexity : $O(n^2)$

5 Linear Program

Variables:

$$x_{vc} \in \{0,1\} \qquad \forall v \in V, c \in \{1,\ldots,k\}$$
$$y_c \in \{0,1\} \qquad \forall c \in \{1,\ldots,k\}$$

Objective:

$$\min \sum_{c=1}^{k} y_c$$

Constraints:

$$\sum_{c=1}^{k} x_{vc} = 1 \qquad \forall v \in V$$

$$x_{uc} + x_{vc} \leq 1 \qquad \forall (u, v) \in E, \forall c$$

$$x_{vc} \leq y_c \qquad \forall v \in V, \forall c$$

6 Experimental Setup

We generate graphs using the Erdos-Renyi model and the Barabasi-Albert model from networkx library. Small graphs for running Brute Force and Large Graphs for Greedy and Heuristic.

Erdos–Renyi (G(n,p)): Creates homogeneous random graphs where each edge exists with probability p.

Barabasi–Albert (G(n,m)): Generates scale free networks through preferential attachment, where new vertices preferentially connect to well-connected existing vertices.

Parameter	**Values/Range**
Graph Type	Erdos–Renyi
Graph Size(nodes)	4–11
Edge probabilities	[0.1, 0.3, 0.5, 0.7]
Number of Instances	23 small graphs
Performance Metrics	Colors used, Computation time

Table 1: small-scale experimental parameters

Parameter	**Values**
Graph Types	Erdos–Renyi, Barabasi-Albert
Graph Size	50,100,200,300,400,500,600,700,800,900,1000
Edge Probabilities	[0.1, 0.3, 0.5, 0.7]
Attachment Parametners	[2, 3, 5]
Number of Instances	77 large graphs
Performance Metrics	Colors used, Computation time

Table 2: Large-scale experimental parameters

All experiments were conducted on google colab with the following specifications:

Component	Specification
Platform	Google Colab (free-tier)
CPU	Intel Xeon Processor @ 2.20GHz
RAM	12.7 GB
Storage	107.7 GB
Python Version	3.10.12

Table 3: System configuration for experiments

6.1 Comparison 1: Comparison of Greedy and Heuristic with brute force

We compared Greedy and DSATUR (heuristic) against Brute Force on small graphs to establish performance baselines and computational limits.

Graph Size	Brute Force Feasible	Avg. Time
$n \leq 10$	Yes	< 1s
$n = 11 - 14$	Marginal	$1 - 30$s
$n = 15$	Boundary	$1 - 5$min
$n \geq 16$	No	> 10min

Table 4: Brute force computational feasibility

Break Point: $n = 15$ vertices was identified as the practical limit for brute force computation.

6.2 Comparison 2: Comparison of Greedy and Heuristic

This comparison evaluates the performance of simply greedy coloring against the more sophisticated DSATUR heuristic on large graphs.

Metric	Greedy	DSATUR	Advantage
Average Colors Used	36.19	32.59	DSATUR
Average Time (ms)	6.92	1473.49	Greedy
Theoretical Bound	$\Delta + 1$ colors	No guarantee	Greedy
Consistency	Moderate	High	DSATUR

Table 5: Performance comparison on large graphs ($n \geq 50$)

6.3 Results

Result 1

Metric	Brute Force	Greedy	DSATUR
Average Colors	3.6956521739130435	4.043478260869565	3.6956521739130435
Average Time (ms)	6422.588959984158	0.023603439331054688	0.10103764741317085

Table 6: Average time and color table for small graphs.

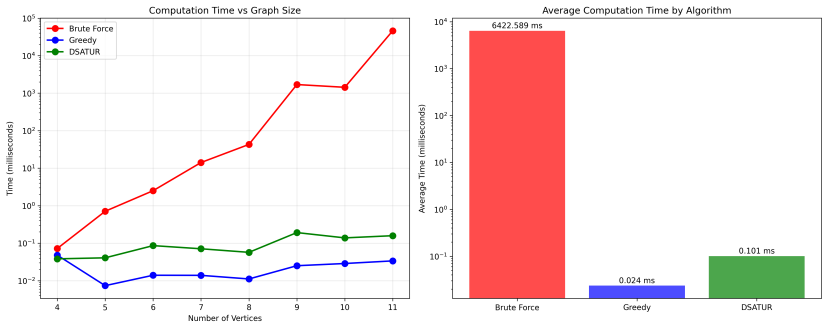

Figure 2: Time comparison of Greedy and Heuristic with Brute force.

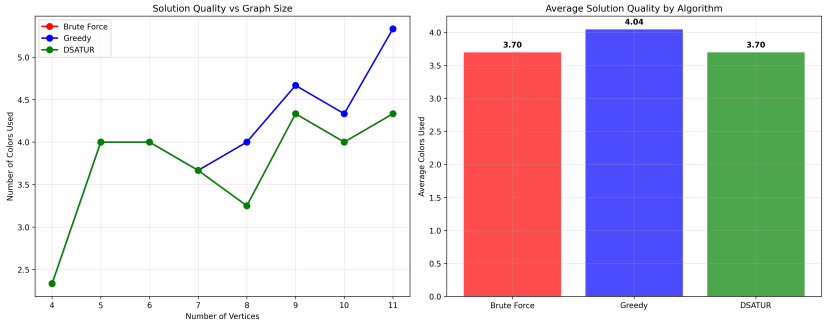

Figure 3: Cost comparison of Greedy and Heuristic with Brute force.

Result 2

Metric	Greedy	DSATUR
Average Colors	36.1948051948052	32.5974025974026
Average Time (ms)	6.920576095581055	1473.4906345218808

Table 7: Average time and color table.

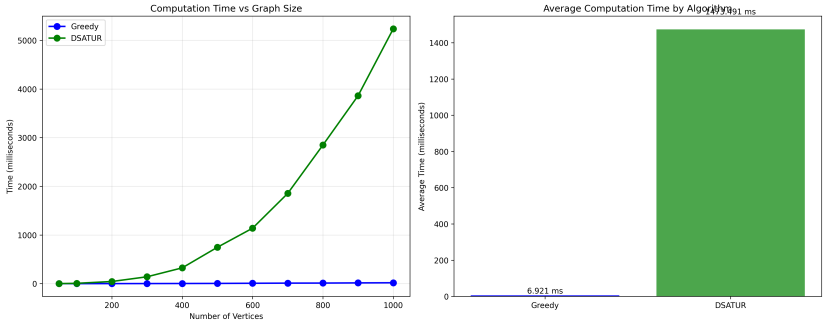

Figure 4: Time comparison between Greedy and DSATUR

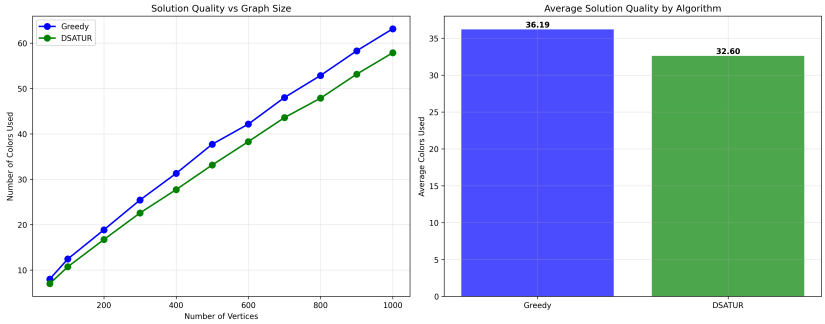

Figure 5: Cost comparison between Greedy and DSATUR

References

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

[2] David Guichard. *An Introduction to Combinatorics and Graph Theory*, pages 120–121. Springer, 2012.

[3] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.