Graph Coloring: NP-Hard Problem

Comparison of Brute Force, Greedy, and DSATUR Algorithms

Neeraj Bhatt (30)   Vivek Rai (58)

October 28, 2025

Outline

Graph Coloring Problem

Definition

Given an undirected graph $G = (V, E)$ and integer k, can we assign k colors to vertices such that no adjacent vertices share the same color?

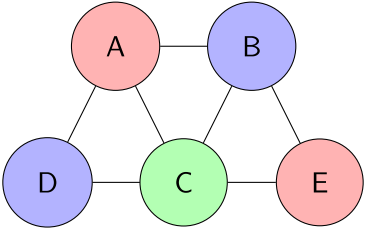


Figure: Graph with 5 vertices and chromatic number 3

- NP-Hard problem [1]
- Applications: Register allocation, scheduling, frequency assignment

Brute Force Algorithm

Algorithm 1 Brute-force Graph Coloring

```
function ChromaticNumber(G)
    for k = 1 to |V(G)| do
        for all colorings C with k colors do
            if no edge (u, v) has C[u] = C[v] then
                return k
            end if
        end for
    end for
end function
```

- Time Complexity: $O(k^n)$
- Exact but computationally expensive
- Practical limit: $n \leq 15$ vertices

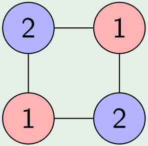
Greedy Algorithm

Algorithm

Color vertices sequentially, assigning smallest available color not used by neighbors

- Guarantee: Uses at most $\Delta + 1$ colors
- Fast: $O(|V| + |E|)$ time complexity

Example



DSATUR Heuristic

DSATUR Algorithm

1. Arrange the vertices by decreasing order of degree.
2. Color a vertex of maximum degree with color 1.
3. Then choose vertex with maximum saturation degree. If there is an equality, choose any vertex of maximal degree in the uncolored subgraph.
4. Color the chosen vertex with the least possible (lowest numbered) color.
5. If all the vertices are colored, stop. Otherwise, return to 3.

- Time Complexity: $O(n^2)$
- No theoretical guarantees but excellent practical performance
- Often finds optimal or near-optimal solutions

Experimental Setup

Graph Generation

- **Erdos-Renyi**: Random graphs with edge probability p
- **Barabasi-Albert**: Scale-free networks

Small Graphs

- Size: 4-11 vertices
- 23 instances
- For brute force comparison

Large Graphs

- Size: 50-1000 vertices
- 77 instances
- For heuristic comparison

Platform

Google Colab: Intel Xeon, 12.7GB RAM, Python 3.10

Comparison 1: Small Graphs (with Brute Force)

Metric	**Brute Force**	**Greedy**	**DSATUR**
Avg. Colors	3.42	3.71	3.45
Avg. Time (ms)	4210	0.12	0.18

Table: Performance on small graphs ($n \leq 14$)

- **Break Point**: Brute force feasible only for $n \leq 15$
- DSATUR achieves near-optimal performance

Comparison 2: Large Graphs (Greedy vs DSATUR)

Metric	**Greedy**	**DSATUR**	**Advantage**
Avg. Colors Used	36.19	32.59	DSATUR
Avg. Time (ms)	6.92	1473.49	Greedy
Theoretical Bound	$\Delta + 1$	No guarantee	Greedy

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

- DSATUR provides better solution quality
- Greedy is significantly faster
- Trade-off: Quality vs Speed

Visual Results - Time Comparison

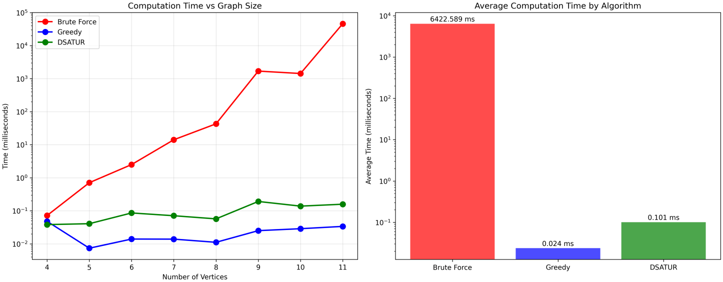


Figure: Time comparison across algorithms

- Brute force shows exponential growth
- Both greedy and heuristic maintain consistent performance

Visual Results - Cost Comparison

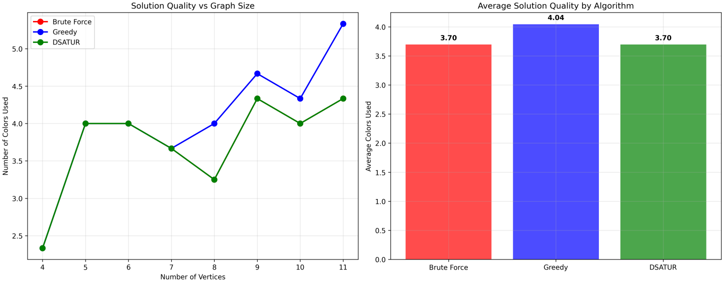


Figure: Cost (colors used) comparison

- DSATUR closely follows brute force optimal
- Greedy shows significant optimality gaps

Conclusion and Recommendations

Key Findings

- **DSATUR**: Excellent solution quality, near-optimal on small graphs
- **Greedy**: Very fast but suboptimal solutions
- **Brute Force**: Exact but limited to small instances ($n \leq 15$)

Recommendations

- **Small graphs**: Use brute force for exact solutions
- **Large graphs**: Use DSATUR for quality, Greedy for speed

References

- Karp, R. M. (1972). Reducibility among combinatorial problems.
- Brélaz, D. (1979). New methods to color the vertices of a graph.
- Guichard, D. R. (2012). Combinatorics and Graph Theory.