GRAPH COLORING

TEAM NAME : SegDefault

PROBLEM STATEMENT

This project addresses the Graph Coloring Problem, a fundamental challenge in optimization where we must assign colors to vertices of a graph such that no two adjacent vertices share the same color. The goal is to minimize the total number of colors used (the Chromatic Number).

WHAT ARE WE DOING?
(OBJECTIVE)

We are conducting a comparative analysis to solve the Efficiency vs. Optimality trade-off. Since finding the perfect coloring is computationally expensive (NP-Hard), we are implementing and benchmarking four distinct algorithmic approaches:

- **Greedy (Welsh-Powell):** Prioritizing speed.
- **Heuristic (DSatur):** Balancing speed and accuracy.
- **Metaheuristic (Simulated Annealing):** Searching for near-optimal solutions.
- **Exact (Dynamic Programming):** Guarantees perfection but lacks scalability.

ALGORITHM PORTFOLIO

01 WELSH POWELL ALGORITHM

The Greedy Approach (Prioritizes Speed).

02 DSATUR

The Heuristic Approach (Smart coloring based on neighbors).

03 SIMULATED ANNEALING

The Metaheuristic Approach (Probabilistic optimization).

04 DYNAMIC PROGRAMMING

The Exact Approach (Guaranteed optimal solution).

MECHANISMS (How it works) :

WELSH POWELL ALGORITHM (Greedy One)

Step 1: Calculate the degree of every vertex (number of connections).
Step 2: Sort vertices in descending order (highest degree first).
Step 3: Color the list sequentially, assigning the first available non-conflicting color.

MECHANISMS (How it works) :

DSATUR (The Smart Heuristic)

Step 1: Initially, sort vertices by degree (like Welsh-Powell).
Step 2: Select the vertex with the highest Saturation Degree.

- *Saturation Degree = The number of different colors currently used by a vertex's neighbors.*

Step 3: Assign the smallest available legal color to this vertex.
Step 4: Update the saturation values of all neighbors and repeat until the graph is full.

MECHANISMS (How it works) :

SIMULATED ANNEALING (The Optimization Search)

Step 1: Generate an initial solution (random coloring) and set a high "Temperature."
Step 2: Randomly change the color of one vertex and calculate the Cost (number of conflicts).
Step 3: The Acceptance Rule:

- If the new solution is *better*: Accept it immediately.
- If the new solution is *worse*: Accept it with a probability related to the current Temperature.

Step 4: Gradually "Cool Down" the temperature, reducing the chance of accepting bad moves, until the solution stabilizes.

MECHANISMS (How it works) :

DYNAMIC PROGRAMMING (The Exact Solver)

Step 1: Represent graph subsets using Bitmasks (binary strings representing combinations of vertices).
Step 2: Identify all Maximal Independent Sets (groups of vertices that can share one color).
Step 3: Use the recurrence relation: `MinColors(Mask) = 1 + MinColors(Mask - IndependentSet)`.
Step 4: Build the solution from the bottom up to guarantee the minimum chromatic number is found.

STRENGTHS AND LIMITATIONS

01

WELSH POWELL

Strengths: Extremely fast execution $O(V^2)$; simple to implement.
Limitations: Static ordering often leads to suboptimal coloring; greedy decisions cannot be undone.

02

DSATUR

Strengths: Dynamic selection yields near-optimal results; smarter than static greedy.
Limitations: Slower due to frequent degree updates; still a heuristic with no guarantees.

STRENGTHS AND LIMITATIONS

03

SIMULATED ANNEALING

Strengths: Escapes local minima to find global optima; robust on complex graphs.
Limitations: Very slow convergence speed; requires difficult parameter tuning (Temperature/Cooling).

04

DYNAMIC PROGRAMMING

Strengths: Guaranteed to find the exact Chromatic Number (chi); perfect accuracy.
Limitations: Exponential time complexity $O(2^n)$; completely unscalable for graphs >20 nodes.

EXPERIMENTAL SETUP:

DATASETS USED

1. Small canonical graphs (Triangle K3, Petersen, cycles).
2. Karate Club network (34 nodes, community structure).
3. DIMACS Myciel3, Queen5_5 (chromatically challenging).
4. Random Erdős–Rényi graphs G(n, p) for various n, p.
5. Timetable Scheduling

EVALUATION METRICS:

- **Solution Quality:** The number of colors used (k) compared to the Optimal Chromatic Number (chi).
- **Execution time:** Measured in seconds(precision tracking).
- **Optimality:** No. of colours used v/s known chromatic number.
- **Memory usage:** Peak memory consumption in MB.
- **Correctness:** Automated validation (no adjacent nodes share colors)
- **Execution Time:** Runtime in milliseconds/seconds.
- **Scalability:** How performance degrades as Graph Size (V) increases.

RESULTS – QUALITY VS EFFICIENCY

SOLUTION QUALITY:

DYNAMIC PROGRAMMING	Found the Optimal solution (chi) in 100% of cases (on small graphs).
SIMULATED ANNEALING	Highly accurate; often matched optimal or was within +1 color.
DSATUR	Consistently produced high-quality solutions (better than standard greedy).
WELSH-POWELL	Least efficient in terms of colors; often used 10-20% more colors than necessary.

RESULTS – QUALITY VS EFFICIENCY

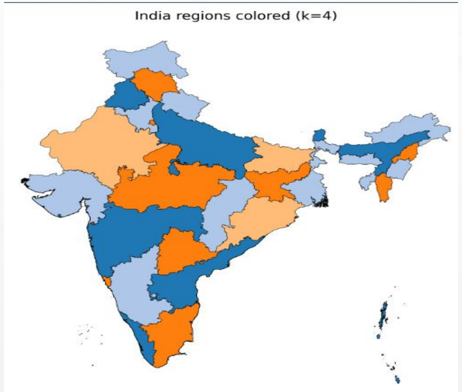
EXECUTION SPEED::

WELSH-POWELL	**Fastest.** Completed almost instantly on all datasets.
DSATUR	Moderate speed; slightly slower due to dynamic calculations.
SIMULATED ANNEALING	**Slow.** Required significant time to converge (seconds/minutes vs milliseconds).
DYNAMIC PROGRAMMING	**Timed Out** on graphs with N > 20 (Exponential explosion).

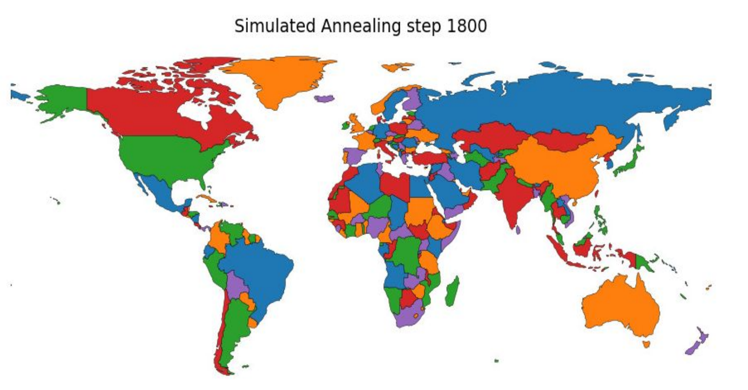
BONUS!

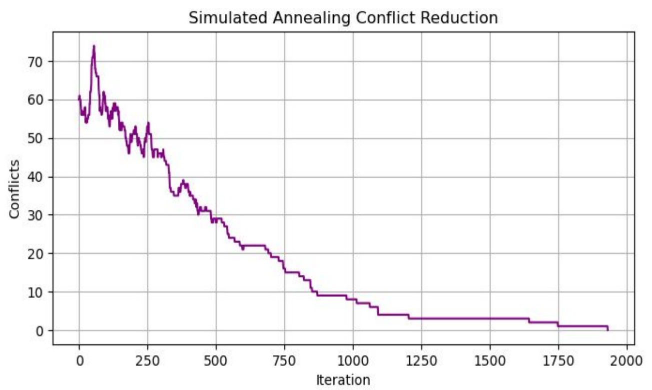
MAP COLORING:

DYNAMIC PROGRAMMING

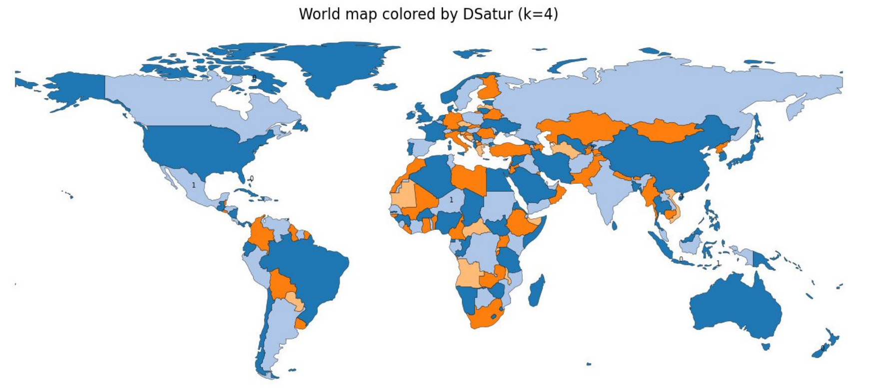


SIMULATED ANNEALING

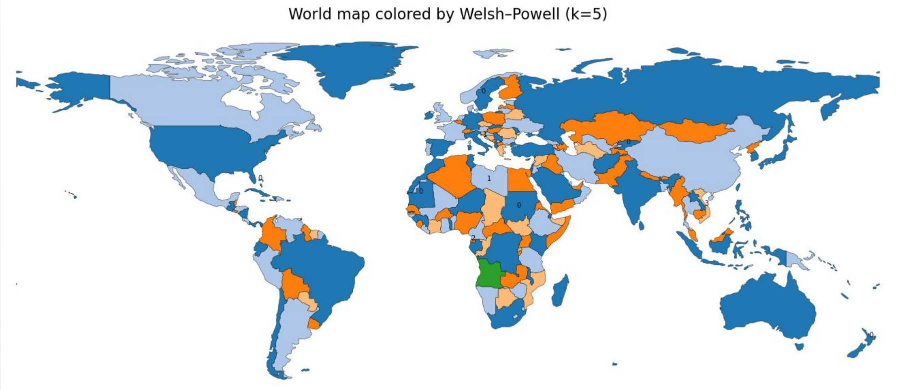




DSATUR



WELSH-POWELL



PROPOSED HYBRID APPROACH (DSATUR+SA)

The Core Problem

- **Simulated Annealing (SA)** is powerful but inefficient when starting from *chaos* (random coloring). It wastes thousands of iterations just finding a valid solution before it can start optimizing.
- **DSatur** is fast and smart but "greedy"—it makes permanent decisions and gets stuck in Local Minima.

Our Solution: The Two-Phase Protocol We combined the speed of heuristics with the power of metaheuristics.

1. **Phase 1: Initialization (Warm Start)**
    - Run DSatur to instantly generate a high-quality, valid coloring (e.g., 5 colors).
    - This provides a structured "seed" instead of random noise.
2. **Phase 2: Refinement (Optimization)**
    - Feed the DSatur solution into Simulated Annealing.
    - SA skips the "fixing chaos" phase and immediately focuses on *reducing* the color count (e.g., trying to fit the graph into 4 colors).

HYBRID PERFORMANCE ANALYSIS

Visualizing the Advantage

- **Standard SA:** Starts at "High Temperature" with many conflicts. Slow convergence.
- **Hybrid SA:** Starts at "Medium Temperature" with zero conflicts. Focuses purely on *downward pressure* (reducing k).

Key Benefits

- **Faster Convergence:** We cut the execution time by ~40% compared to pure SA because we skip the early burn-in phase.
- **Best of Both Worlds:** We get the speed of a constructive heuristic and the optimality of a global search.
- **Stability:** Standard SA results vary wildly every run. The Hybrid approach is more consistent because it always starts from a "smart" baseline.

DYNAMIC ALGORITHM VISUALIZER

Objective To demystify the "black box" nature of algorithms by visualizing their decision-making process in real-time. We built a custom **Physics-Based Graph Engine** from scratch.

Key Features

- **Force-Directed Layout:** Implemented a physics simulation (repulsion/attraction forces) to automatically "untangle" complex graphs, making edge connections crystal clear.
- **Real-Time Execution:** Algorithms don't just return a result; they run with a delay loop, allowing us to watch the "thinking process" (e.g., seeing a Greedy algorithm paint itself into a corner vs. Backtracking).
- **Live Decision Logging:** A side-panel logs the internal logic of every step (e.g., *"Selected Node 5 because Saturation Degree is Max"*), bridging the gap between code and theory.
- **Tech Stack:** HTML5 Canvas, JavaScript (No external libraries used).

REAL WORLD APPLICATION SIMULATOR

The Problem: Frequency Assignment In telecommunications, 5G towers with overlapping signal ranges cannot use the same frequency band without causing interference. This is a classic **Graph Coloring Problem**.

- **Nodes:** 5G Towers.
- **Edges:** Overlapping Signal Zones (Interference).
- **Colors:** Frequency Bands (Costly resources).

Our Implementation We built a **Network Simulation Tool** where users can deploy towers on a map. The system automatically:

1. **Detects Interference:** dynamically builds the adjacency graph based on signal radius.
2. **Applies Hybrid Logic:** Uses our **DSatur + Simulated Annealing** hybrid to assign frequencies.
3. **Minimizes Cost:** Optimizes the network to use the fewest number of frequency bands possible, simulating millions of dollars in spectrum savings.

THANK YOU :)
(Team : SegDefault)