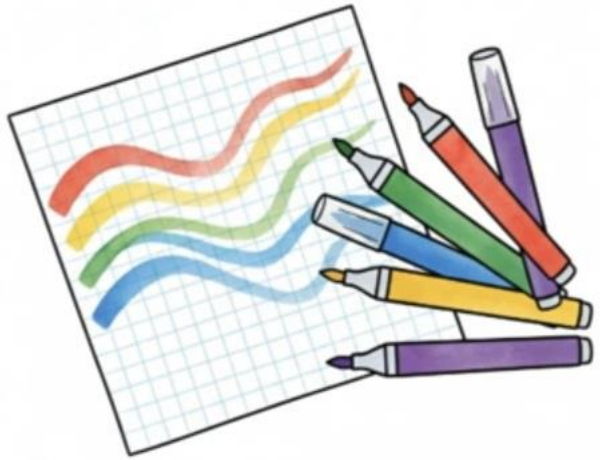
جامعة الملك سعود
King Saud University

Tabu Search for Graph Coloring

By:

Mohammed Mahdy
446910613
Mohammed Ewida
446910614



Under the supervision of:

Prof. Manar Hosny

Problem Statement

Methodology

Algorithm

Dataset & Parameters

Results & Discussion

Problem Statement

How to schedule exams with no conflicts?

Graph Coloring Problem (GCP)

Assigning a 'color' to each point in a graph so that no two connected points share the same color.

The Objective

Finding the Chromatic Number is an NP-hard problem, making it practically impossible to find the perfect solution for large graphs in reasonable time.

Problem Components

Input

An undirected graph G = (V, E) with vertices and edges.

Output

A valid coloring function that assigns a color to each vertex.

Constraint

Adjacent vertices MUST have different colors.

Objective

Minimize the total number of colors used (k).

Methodology

Greedy Algorithm: A constructive heuristic. It is simple, extremely fast, but often suboptimal.

Tabu Search: An improvement metaheuristic. It is complex and slow, but aims for near-optimal quality.

Greedy Algorithm

Step 1: Order

Order all the vertices in a list.

Step 2: Assign

Assign the first available color to the first vertex.

Step 3: Iterate

Move to the next vertex in the list.

Step 4: Color

Assign the first color that doesn't conflict with its neighbors.

Step 5: New Color

If no existing color works, add a new one.

Step 6: Repeat

Repeat until all vertices are colored.

Pros: Very high speed and simple to implement.

Cons: Highly sensitive to the initial ordering and gets stuck in suboptimal solutions.

How can a search be smarter to avoid getting trapped?

Tabu Search

Tabu Search doesn't start over. It starts with a solution (from the Greedy algorithm) and intelligently refines it.

The Core of Tabu Search

- Tabu List: A short-term memory (FIFO queue) to forbid recent moves.
- Aspiration: Overrides tabu rules for exceptionally good moves.
- Sampling: Randomly samples moves for greater efficiency.
- Reduction: Iteratively tries to find a solution with one fewer color.

Tabu List

A short-term memory of recent moves that forbids undoing a change, preventing it from getting stuck.

Aspiration Criterion

Allows a 'tabu' move if it leads to a better solution than ever seen before. Helps escape local optima.

Algorithm

TS Core Algorithm

The implementation uses an **Iterative Color Reduction Strategy**.

1. **Start:** Begin with the k0 coloring solution provided by the Greedy algorithm.
2. **Attempt Reduction:** Set the target number of colors to k = k0 - 1. Re-assign all vertices with colors > k to random colors in the new, smaller range {1, ..., k - 1}.
3. **Search:** Run the Tabu Search local search mechanism for a maximum of **T** iterations to try and find a feasible solution (where conflicts F(C) = 0).
4. **Repeat:**
   - **If a feasible solution is found:** The solution is saved. The algorithm repeats step 2, attempting to find a solution with k-1 colors.
   - **If no feasible solution is found (timeout):** The search fails for k colors. The algorithm terminates.
5. **Result:** The best feasible solution (the one with the minimum k) found during the entire process is returned.

TS Core Mechanism

- **Neighborhood Moves:** A "move" consists of selecting a vertex v that is currently in a conflict and changing its color c to a new color c'.
- **Tabu List (Short-Term Memory)**
  - **What it is:** A simple FIFO (First-In-First-Out) queue of a fixed size L (e.g., L=10).
  - **How it works:** When a vertex v is moved from color c, the pair (v, c) is added to the list.
  - **Purpose:** It forbids the search from reversing this move (i.e., moving v back to c) for the next L iterations. This **prevents cycling** and forces the search to explore new regions.
- **Aspiration Criterion (Dynamic)**
  - **Purpose:** To override the Tabu List and prevent good moves from being blocked.
  - **How it works:** A tabu-forbidden move *is* accepted if it results in a solution with a lower conflict count than *any solution previously seen* at that conflict level.

TS Intensification & Diversification

- **Intensification Strategies**
  - Instead of evaluating all possible neighborhood moves (which is computationally expensive), the algorithm randomly samples R candidate moves (e.g., R=50).
  - It then applies a **first-improvement** strategy, accepting the *first* sampled move that reduces the conflict count and is not tabu (or satisfies aspiration).
- **Diversification Strategies**
  - **Tabu List:** Forces exploration by forbidding recent moves.
  - **Randomized Sampling:** Introduces stochastic variation, preventing deterministic patterns.
  - **Iterative Color Reduction:** The outer loop naturally forces the algorithm to explore new, more constrained solution spaces.

Dataset & Parameters

Experimental Setup

- **Environment:** Python 3.x, Jupyter, NetworkX, NumPy, and Matplotlib.
- **Hardware:** Intel Core i7-14700KF, 32.0 GB RAM.
- **Datasets:** used standardized **DIMACS Benchmark Instances** to ensure generalizable and comparable results.

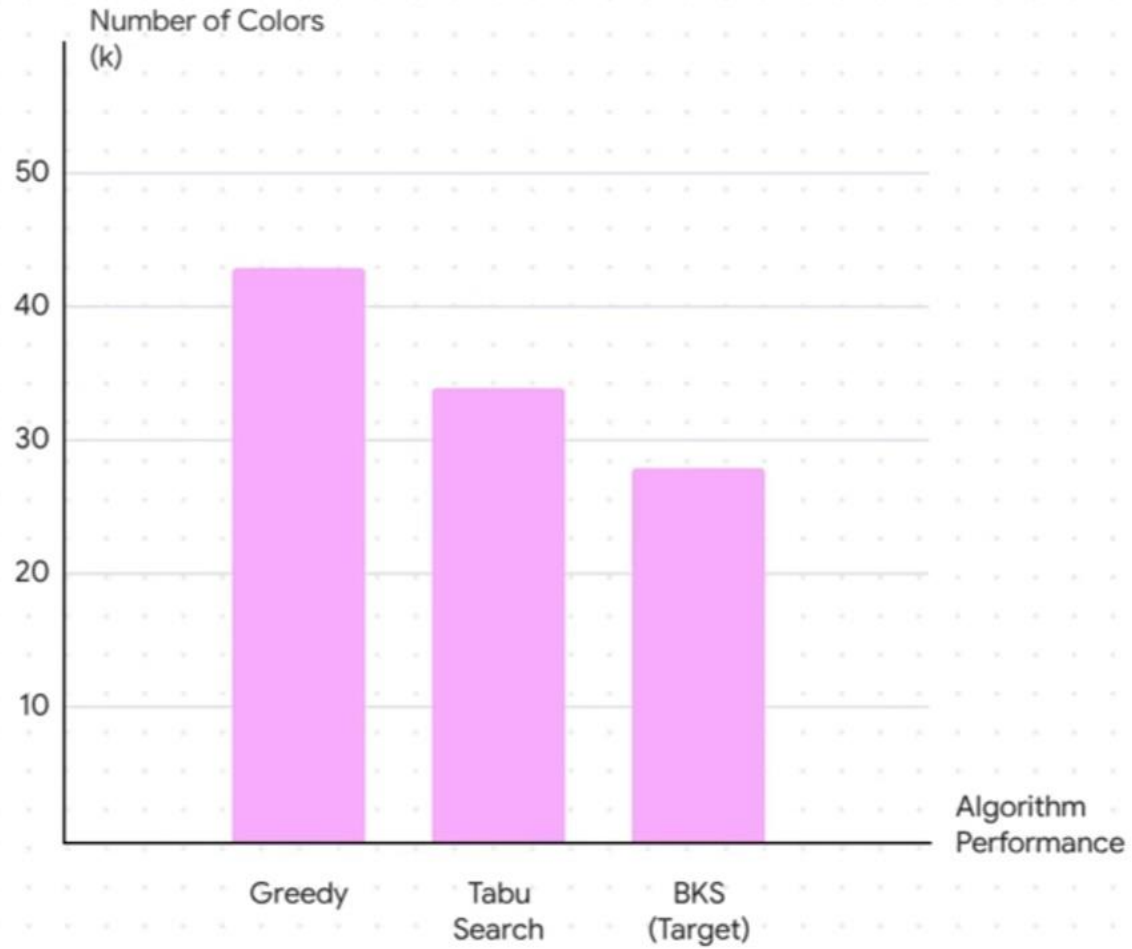
DIMACS

Instance	Vertices	Edges	Best Known Solution (Colors)
dsjc250.5	250	31,336	28
dsjc500.9	500	224,874	126
dsjc1000.5	1000	249,826	85

TS Parameters

Parameter (Factor)	Levels Tested	Optimal Value	Rationale
Tabu List Size (L)	3, 5, 10, 30, 50	10	Provides effective cycle prevention without being overly restrictive[60].
Random Sampling Size (R)	30, 40, 50	50	Balances exploration breadth with computational efficiency[62].
Max Iterations (T)	3k, 5k, 7k, 9k	7,000	Sufficient time for convergence per color while maintaining reasonable total runtime[64].

Results & Discussion



Tabu Search gets *significantly closer* to the optimal solution than Greedy for instance dsjc250.5.

21.42%

But this dramatic improvement in quality comes at a staggering cost...

40,000x

Instance	Algorithm	Avg. Colors	Avg. Time (s)	Deviation
dsjc250.5	Greedy	43	0.008	53.57%
dsjc250.5	Tabu Search	35	344.97	21.42%
dsjc1000.5	Greedy	127	0.129	49.40%
dsjc1000.5	Tabu Search	126	3969.29	47.05%

> TS finds **significantly better** solutions than Greedy, but at a **high computational cost**.

Evaluation Matrix

Performance was evaluated using four key metrics:

1. **Solution Quality (k):** The final (minimum) number of colors achieved in a feasible solution.
2. **Percent Deviation (%):** How far the solution is from the Best-Known Solution (BKS).

   Deviation(%) = (Obtained Colors – BKS) / BKS * 100

3. **Robustness:** The standard deviation (sigma) of colors found over 5 independent runs.
4. **Computational Effort:** The average run time in seconds.

Discussion and Analysis

The results highlight a clear trade-off:

- **Superior Solution Quality:**
  - Tabu Search **consistently and significantly outperformed** the Greedy algorithm in all three benchmark instances.
  - It demonstrates the power of TS to escape the local optima that trap the simple Greedy heuristic. For dsjc250.5, TS cut the deviation from BKS by more than half (from 53.57% down to 21.42%).

- **The "Time vs. Quality" Compromise:**
  - This superior quality comes at a **massive increase in computational time**.
  - The Greedy algorithm completed all instances in fractions of a second (e.g., 0.129s).
  - In contrast, Tabu Search required significant time, scaling up to 3969 seconds (approx. 66 minutes) for the largest instance.

Limitations and Conclusion

- **Limitations:**
  - A **considerable gap to the Best Known Solution (BKS) remains** (e.g., 21.42% deviation for dsjc250.5).
  - The relative improvement of TS diminished on denser graphs (e.g., dsjc1000.5).
  - This is a basic TS implementation. State-of-the-art solvers use more advanced mechanisms (like long-term memory) to achieve better results.
- **Conclusion:**
  - The results validate the hypothesis that a single-solution metaheuristic (Tabu Search) can find **significantly better solutions** for the GCP than a simple constructive heuristic (Greedy).
  - This improvement, however, comes at a **very high computational cost**.

When to choose a fast solution over the best one?

Thank You