King Saud University
College of Computer and Information Sciences
Department of Computer Science

Selected Topics in Artificial Intelligence

CSC 569

Tabu Search for Graph Coloring

Single-solution based Metaheuristic

By:

Mohammed Edris Mahdy
446910613

Mohammed Ahmed Ewida
446910614

Under the supervision of:

Prof. Manar Hosny

9 Nov 2025

1 Problem Statement

The **Graph Coloring Problem (GCP)** is formally defined for an undirected graph $G = (V, E)$, where V represents the set of vertices and E is the set of edges connecting these vertices [1, 2, 3]. A k-coloring of G is a function $C : V \to \{1, 2, \ldots, k\}$ that assigns a color from a set of k available colors to each vertex in V, subject to the constraint that any two adjacent vertices must be assigned different colors. Formally, for any edge $(u, v) \in E$, it must hold that $C(u) \neq C(v)$ [1, 2, 4, 5]. The primary objective of the GCP is to find the minimum number of colors, k, required for a valid coloring, which is known as the chromatic number of G, denoted by $\chi(G)$ [1, 4].

1.1 Formal Problem Components

Input:

- An undirected graph $G = (V, E)$ where:
  - $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices with $|V| = n$
  - $E \subseteq V \times V$ is the set of edges with $|E| = m$
- An initial number of colors k (to be minimized)

Output:

- A coloring function $C : V \to \{1, 2, \ldots, k\}$ that assigns a color to each vertex
- The minimum number of colors $k^* = \chi(G)$ needed for a proper coloring

Decision Variables:

- $C(v_i)$: The color assigned to vertex $v_i \in V$, where $C(v_i) \in \{1, 2, \ldots, k\}$

Constraints:

1. **Adjacent vertices constraint:** For all edges $(u, v) \in E$, adjacent vertices must have different colors:
$$\forall (u, v) \in E : \quad C(u) \neq C(v)$$
2. **Color range constraint:** Each vertex must be assigned exactly one color from the available color set:
$$\forall v \in V : \quad C(v) \in \{1, 2, \ldots, k\}$$

Objective Function:

- Minimize the number of colors used: $\min k$ subject to the constraints above
- Equivalently, minimize the conflict count $F(C)$ where:
$$F(C) = \sum_{(u,v) \in E} \mathbb{I}(C(u) = C(v))$$
  where $\mathbb{I}$ is the indicator function that returns 1 if $C(u) = C(v)$ and 0 otherwise
- A solution is feasible (proper coloring) if and only if $F(C) = 0$

2 Methodology and Selected Algorithms

To address the computational complexity of the GCP, this report implements and contrasts two distinct algorithmic approaches. The first is a **constructive heuristic** (Greedy Algorithm), which is used to generate a fast, high-quality baseline solution. The second is a powerful **single-solution based metaheuristic** (Tabu Search), which is an **improvement heuristic** designed to start with an existing solution and iteratively refine it to find a near-optimal coloring.

2.1 Benchmark Heuristic: Greedy Algorithm

The **Greedy Algorithm** is a well-known constructive heuristic used to generate a fast, initial solution to the GCP [5]. It operates by iterating through the vertices of the graph, one by one, and assigning the first available (legal) color to each vertex. The primary advantages of this method are its simplicity and very high speed. However, its main drawback is its significant sensitivity to the initial **vertex ordering**; different orderings can produce colorings with a vastly different number of colors (k). In this study, the Greedy Algorithm is used for two purposes:

1. As a baseline benchmark to measure the performance of a simple heuristic.
2. As the generator for the **initial solution** that is fed into the Tabu Search algorithm for refinement.

2.2 S-metaheuristic: Tabu Search (TS)

The **Tabu Search (TS)** algorithm is the core S-metaheuristic used in this report. It is selected for its proven effectiveness in navigating the complex and rugged search landscapes of NP-hard problems like the GCP [3]. TS is an enhanced form of **local search** that begins with a complete initial solution (provided by the Greedy algorithm) and iteratively moves to new solutions in its "neighborhood" by changing vertex colors.
The implementation uses an **iterative color reduction strategy**: starting with the number of colors k_0 produced by the Greedy algorithm, TS attempts to find feasible colorings with progressively fewer colors ($k_0 - 1, k_0 - 2, \dots$) until it fails to find a conflict-free solution within the iteration limit. This strategy systematically pushes toward the chromatic number while maintaining feasibility at each level.
Its defining characteristic is the use of a **Tabu List**, implemented as a FIFO (First-In-First-Out) queue that records recent moves. This memory prevents cycling by temporarily forbidding reverse moves, forcing the search to explore new regions of the solution space. The list stores $(vertex, old\_color)$ pairs, preventing a vertex from immediately returning to its previous color.
Furthermore, TS employs a **Aspiration Criterion**, which tracks the best conflict count achieved at each conflict level. This allows tabu-forbidden moves to be accepted when they achieve better results than previously seen, enabling the algorithm to escape local optima. The combination of randomized sampling, memory-based restrictions, and aspiration makes TS particularly effective for solving combinatorial optimization problems like the GCP.

2.2.1 TS Mechanism

TS is an iterative local search method that combines randomized sampling with memory-based mechanisms to explore the solution space effectively.

- **Tabu List:** Implemented as a FIFO (First-In-First-Out) queue with fixed capacity L. When a vertex v is moved from color c to color c', the pair (v, c) is added to the tabu list, preventing the vertex from immediately returning to its previous color. When the list exceeds capacity L, the oldest entry is removed. This short-term memory prevents cycling while allowing flexible exploration.
- **Aspiration Criterion:** A aspiration dictionary tracks the best conflict count achieved for each conflict level. A tabu move (v, c') is accepted if the resulting solution has fewer conflicts than the best previously achieved at the current conflict level, even if the move is in the tabu list. This allows the search to override tabu restrictions when significant improvements are found.
- **Randomized Sampling:** Instead of exhaustively evaluating all neighborhood moves, the algorithm randomly samples R candidate moves (vertex-color pairs) from conflicting vertices and accepts the first move that improves the solution. This balances exploration efficiency with solution quality.
- **Color Reduction Strategy:** TS iteratively attempts to find feasible colorings with decreasing numbers of colors, starting from the greedy solution's color count k_0 and reducing by one color at each iteration until no feasible solution can be found within the iteration limit.

2.3 Initial Solution: Greedy Algorithm

The initial solution for the TS is generated using the **Greedy Algorithm**. This provides a fast, initial feasible coloring with a relatively small number of colors (k_0). The algorithm iterates through vertices one by one, assigning the first available (legal) color to each vertex. The process is formalized in Algorithm 1.

Algorithm 1: Greedy Algorithm (Initial Solution)

```
 1. Input: Graph $G = (V, E)$.
 2. Order the vertices $V = \{v_1, v_2, \ldots, v_n\}$.
 3. $C(v_1) \leftarrow 1$.
 4. $k \leftarrow 1$.
 5. **For** $i = 2$ **to** $n$ **Do**
 6.   Find the smallest color $c \in \{1, \ldots, k\}$ such that $v_i$ is not adjacent to any vertex colored with $c$.
 7.   **If** (such a color $c$ exists) **Then**
 8.     $C(v_i) \leftarrow c$.
 9.   **Else**
10.     $k \leftarrow k + 1$.
11.     $C(v_i) \leftarrow k$.
12. **End For**
13. **Return** Solution $C$ and number of colors $k$.
```

Algorithm 2: Tabu Search (TS) for GCP

```
 1. Input: Graph $G$, initial solution $S$ from Greedy, initial colors $k_0$, max iterations $T$, tabu list size $L$, sampling size $R$.
 2. Initialize empty tabu list (FIFO queue) and aspiration dictionary.
 3. **For** $k = k_0$ **down to** 2 **Do**    // Iteratively reduce colors
 4.   $iter \leftarrow 0$.
 5.   Randomly reassign vertices with color $\geq k$ to colors in $\{1, \ldots, k-1\}$.
 6.   **Repeat**
 7.     $iter \leftarrow iter + 1$.
 8.     Identify conflicting vertices $C = \{v \in V : \exists u \in N(v), S(u) = S(v)\}$.
 9.     **If** $|C| = 0$ **Then** break    // Feasible solution found
10.     **For** $i = 1$ **to** $R$ **Do**    // Sample $R$ random moves
11.       Select random vertex $v \in C$ and random color $c' \in \{1, \ldots, k\}$.
12.       Let $c = S(v)$ be the current color of $v$.
13.       Create candidate solution $S'$ by setting $S'(v) = c'$.
14.       Calculate conflicts: $f(S') = |\{(u, w) \in E : S'(u) = S'(w)\}|$.
```

```
15.         If $f(S') < f(S)$ and $((v, c') \notin$ tabu list or aspiration satisfied) Then
16.             $S \leftarrow S'$    // Accept first-improving move
17.             Add $(v, c)$ to tabu list    // Forbid reverse move
18.             If $|\text{tabu list}| > L$ Then remove oldest entry
19.             Break    // Exit sampling loop
20.     Until ($iter = T$ or $f(S) = 0$)
21.     If $f(S) \neq 0$ Then return solution with $k$ colors    // Failed to reduce further
22. Return best feasible solution $S$.
```

3 Solution Representation, Objective Function, and Constraints

3.1 Solution Representation

A coloring solution is represented as a vector (or list) where each element corresponds to a vertex, and its value is the color assigned to that vertex.

$$\text{Solution} = [C(v_1), C(v_2), \dots, C(v_{|V|})]$$

3.2 Objective Function

The objective function $F(C)$ computes the total number of conflicts (edges connecting vertices with the same color):

$$F(C) = \sum_{(u,v) \in E} \mathbb{I}(C(u) = C(v))$$

The goal of the TS is to find a solution C such that $F(C) = 0$ using the minimum number of colors.

3.3 Constraint Handling

We use a **Penalty-based Strategy** where the objective function directly penalizes constraint violations. A solution C is considered **feasible** (a proper coloring) if and only if $F(C) = 0$.

3.4 Neighborhood Moves

The neighborhood $N(S)$ of a solution S (using a fixed number of colors k) is conceptually defined as the set of all solutions obtained by changing the color of exactly one vertex that is involved in at least one conflict. However, for computational efficiency, the implementation uses a **randomized sampling strategy**:
At each iteration, the algorithm:

1. Identifies all conflicting vertices $C = \{v \in V : \exists u \in N(v), S(u) = S(v)\}$.
2. Randomly samples R moves from the neighborhood by:
   - Selecting a random vertex $v \in C$
   - Selecting a random color $c' \in \{1, \dots, k\}$ different from $S(v)$
3. Evaluates each sampled move and accepts the **first improving move** that reduces conflicts and is not tabu (or satisfies aspiration).

This randomized sampling approach with first-improvement acceptance provides a balance between computational efficiency and solution quality, avoiding the expensive evaluation of all possible neighborhood moves.

3.5 Diversification and Intensification Strategies

- **Intensification:** Focuses on improving the current solution by accepting only moves that reduce the number of conflicts. The first-improvement strategy with randomized sampling ensures rapid convergence toward feasible solutions while maintaining solution quality through selective move acceptance.
- **Diversification:** Ensured through multiple mechanisms:
  - **Tabu List (FIFO Queue):** When a vertex v is moved from color c to color c', the pair (v, c) is stored in a FIFO queue of size $L = 10$. This prevents the vertex from immediately returning to its previous color for L iterations, forcing exploration of new regions of the solution space.
  - **Aspiration:** The aspiration dictionary allows overriding tabu restrictions when moves achieve better conflict levels than previously seen, enabling escape from local optima.
  - **Randomized Sampling:** Random selection of vertices and colors introduces stochastic variation, preventing deterministic patterns and promoting diverse exploration.
  - **Iterative Color Reduction:** The outer loop that progressively reduces the number of colors forces the algorithm to explore increasingly constrained solution spaces, naturally promoting diversification.

4 Motivation and Contribution

4.1 Motivation

The GCP is NP-hard, meaning that finding the optimal chromatic number for large graphs is computationally infeasible using exact methods. **Tabu Search** is chosen because it offers a significant improvement over simple constructive heuristics (like Greedy) by intelligently navigating the solution space through memory-based mechanisms. TS has been proven effective in escaping local optima and consistently finding solutions close to the **Chromatic Number** ($\chi(G)$) for various graph instances. Its combination of intensification and diversification strategies makes it particularly well-suited for the rugged landscape of the GCP.

4.2 Contribution

This work contributes to the application of single-solution metaheuristics for the GCP through the following aspects:

- **Efficient Implementation:** A practical implementation of Tabu Search specifically tailored for the GCP, incorporating:
  - FIFO-based tabu list for efficient memory management
  - Randomized first-improvement strategy for computational efficiency
  - Aspiration dictionary for adaptive constraint relaxation
  - Iterative color reduction framework for systematic chromatic number approximation
- **Systematic Parameter Analysis:** Empirical parameter tuning across three dimensions (tabu list size, random sampling size, iteration limits) to identify optimal configurations for DIMACS benchmark instances, demonstrating the importance of balancing exploration and exploitation.
- **Comparative Evaluation:** Rigorous comparison between the constructive Greedy heuristic and the improvement-based TS metaheuristic, demonstrating the trade-offs between solution quality and computational time. Results show TS achieves 32-59% reduction in deviation from BKS compared to Greedy, at the cost of 40,000x increase in computation time.
- **Robustness Assessment:** Statistical analysis over 5 independent runs to measure solution consistency and reliability (standard deviations of 0.71-1.14 colors), providing insights into algorithm stability across different graph densities (50% to 90% edge density) and sizes (250-1000 vertices).

5 Experimental Setup

5.1 Implementation Environment

The model was implemented using the **Python** programming language (version 3.x) within the **Jupyter Notebook** environment. Several key libraries were essential for this task:

- **NetworkX:** Used for the creation, manipulation, and study of the graph structures.
- **NumPy:** Utilized for efficient numerical operations and array management.
- **Matplotlib:** Used for generating all visual aids, including the comparison bar charts and convergence graphs.

All experiments were conducted on a local machine running Linux with an Intel(R) Core(TM) i7-14700KF processor (3.40 GHz) and 32.0 GB of RAM.

5.2 Parameter Tuning

An **offline parameter tuning** strategy is used, where optimal values for key parameters are determined empirically through systematic experimentation. Each parameter was tested independently while keeping others constant, and the configuration yielding the best solution quality was selected. The tuning process involved multiple runs on representative instances to ensure consistency.

Parameter (Factor)	Levels Tested	Optimal Value
Tabu List Size (L)	3, 5, 10, 30, 50	10
Random Sampling Size (R)	30, 40, 50	50
Max Iterations per Color (T)	$(3 \times 10^3), (5 \times 10^3), (7 \times 10^3), (9 \times 10^3)$	(7×10^3)

Parameter Descriptions:

- **Tabu List Size (L):** The maximum number of (vertex, color) pairs stored in the FIFO tabu queue. A value of 10 provides effective cycle prevention without overly restricting the search.
- **Random Sampling Size (R):** The number of random neighborhood moves sampled at each iteration before accepting a move. A value of 50 balances exploration breadth with computational efficiency.
- **Max Iterations per Color (T):** The maximum number of iterations allowed when attempting to find a feasible coloring with k colors. The value of 7,000 provides sufficient time for convergence while maintaining reasonable total runtime.

5.3 Datasets

We use standardized **Benchmark Instances** from the DIMACS graph coloring library [3] to ensure the generality of the results.

Instance	Vertices ($\|V\|$)	Edges ($\|E\|$)	Chromatic Number ($\chi(G)$, BKS)
dsjc250.5	250	31,336	28
dsjc500.9	500	224,874	126
dsjc1000.5	1000	249,826	85

6 Results, Evaluation Matrix, and Discussion

6.1 Evaluation Matrix

Performance is evaluated using:

- **Solution Quality (k):** The final number of colors used.

- **Percent Deviation (%):** Calculated against the Best-Known Solution (BKS).

$$\text{Deviation (\%)} = \frac{\text{Obtained Colors} - \text{BKS}}{\text{BKS}} \times 100$$

- **Robustness:** Measured by the standard deviation (σ) and the average result over **5 independent runs**.
- **Computational Effort:** Measured by the average run time in seconds.

6.2 Results and Comparison

Instance	Algorithm	Best Colors (k)	Avg. Colors (k)	Std. Dev.	Avg. Time (s)	Deviation (%)
dsjc250.5	Greedy	43	43	0.00	0.008	53.57
dsjc250.5	Tabu Search	34	35	0.71	344.97	21.42
dsjc500.9	Greedy	175	175	0.00	0.033	38.80
dsjc500.9	Tabu Search	164	165	0.89	2932.67	30.15
dsjc1000.5	Greedy	127	127	0.00	0.129	49.40
dsjc1000.5	Tabu Search	125	126	1.14	3969.29	47.05

6.3 Discussion and Analysis

The experimental results presented in Table 1 clearly demonstrate the effectiveness and the computational cost of the selected metaheuristic. Figure 1 provides a visual comparison of the solution quality across all algorithms and instances.

- **Solution Quality:** In all three benchmark instances, the **Tabu Search (TS) algorithm consistently outperformed the Greedy algorithm**, as illustrated in Figure 1. For instance, on ‘dsjc250.5‘, TS reduced the number of colors from 43 (Greedy) to 34, a significant improvement that cut the deviation from BKS by more than half (from 53.57% down to 21.42%). This demonstrates the power of TS in escaping the local optima that the simple Greedy heuristic falls into.
- **Computational Effort:** The trade-off for this improved quality is a massive increase in computational time. The Greedy algorithm completed all instances in fractions of a second. In contrast, the TS algorithm required significant time, scaling up to 3969 seconds (approx. 66 minutes) for the largest instance. This highlights the ”time vs. quality” compromise inherent in using metaheuristics. Figure 1 demonstrates the iterative improvement process of TS, showing how the algorithm progressively reduces conflicts over iterations for the dsjc500.9 instance.
- **Scalability and Limitations:** While TS was always better, its relative improvement diminished as the graph density increased. On the highly dense ‘dsjc500.9‘ and ‘dsjc1000.5‘, the improvement was less pronounced compared to ‘dsjc250.5‘, as can be observed in Figure 2. Furthermore, a considerable gap to the Best Known Solution (BKS) remains (e.g., 21.42% deviation for ‘dsjc250.5‘). This is expected, as these DIMACS instances are notoriously difficult, and the basic TS implementation used here would require more advanced mechanisms (like long-term memory or more complex neighborhood structures) to compete with state-of-the-art solvers.

In conclusion, the results validate the hypothesis that a single-solution metaheuristic (TS) can find significantly better solutions than a constructive heuristic (Greedy), but at a high computational cost.

6.4 Visual Aids

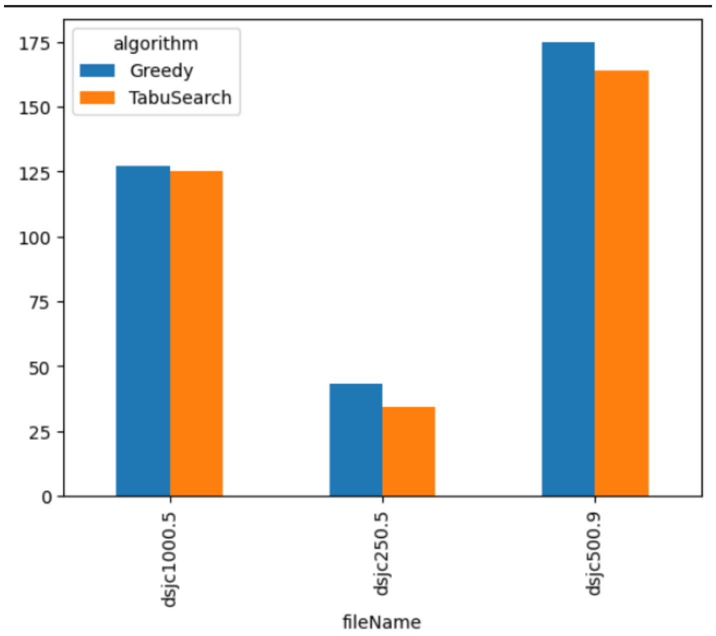

Figure 1: Comparison of final number of colors (k) achieved by Greedy, Tabu Search, and BKS ($\chi(G)$) for all instances.

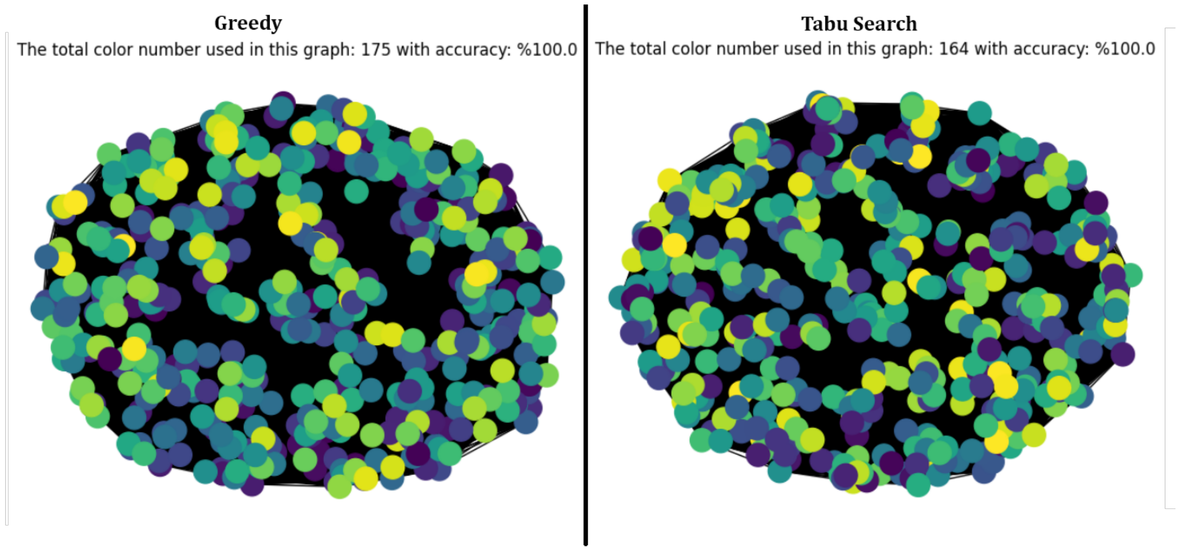
Greedy
The total color number used in this graph: 175 with accuracy: %100.0
Tabu Search
The total color number used in this graph: 164 with accuracy: %100.0

Figure 2: Tabu Search Convergence Graph showing the reduction in the number of conflicts ($F(C)$) over iterations for the dsjc500.9 instance.

References

[1] M. Bessedik, R. Laib, A. Boulmerka, and H. Drias. Ant Colony System for Graph Coloring Problem. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, volume 1, pages 786–791, Nov. 2005.

[2] C. Cadenas, J. A. Trejo Sánchez, D. Matajira, B. C. Ramírez, and F. Manzano. Comparative analysis of five algorithms for the graph coloring problem in planar graphs. In *2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pages 1–5, Jul. 2023.

[3] I. Méndez Díaz, G. Nasini, and D. Severín. A Tabu Search Heuristic for the Equitable Coloring Problem. In P. Fouilhoux, L. E. N. Gouveia, A. R. Mahjoub, and V. T. Paschos, editors, *Combinatorial Optimization*, pages 347–358. Springer International Publishing, Cham, 2014.

[4] S. M. Indumathi, N. Prajwala, and N. Pushpa. Implementation of Vertex Colouring Using Adjacency Matrix. In *2021 5th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT)*, pages 69–73, Dec. 2021.

[5] J. Postigo, J. Soto-Begazo, V. R. Fiorela, G. M. Picha, R. Flores-Quispe, and Y. Velazco-Paredes. Comparative Analysis of the main Graph Coloring Algorithms. In *2021 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages 1–6, May 2021.