ALGORITHMS & COMBINATORIAL OPTIMIZATION

M-Coloring Problem using Backtracking

Systematic constraint satisfaction, state-space tree traversal, and optimal graph coloring strategies.

Computer Science • Graph Theory

FOUNDATIONS

What is the M-Coloring Problem?

Given an undirected graph **G = (V, E)** and an integer **m**, assign colors to all vertices such that no two adjacent vertices share the same color.

- **Decision Problem:** Can graph G be colored using at most m colors?
- **Optimization Form:** Find the minimum m (Chromatic Number χ(G)).
- **NP-Complete:** Decision variant is NP-Complete for m ≥ 3.



MATHEMATICAL FORMULATION

Adjacency & Validity Rule

FUNDAMENTAL CONSTRAINT

$$c(u) \neq c(v) \quad \forall \ (u, v) \in E$$

Where $c(v) \in \{1, 2, \ldots, m\}$ represents the color assigned to vertex v.

Valid State

Every neighboring vertex has a distinct assigned color.

Invalid State (Conflict)

Two adjacent vertices share the exact same color label.

SEARCH PARADIGM

Why Backtracking Beats Brute Force

Pruning Dead Ends

Abandons invalid partial assignments immediately.

State-Space Traversal

Explores depth-first through vertex assignments.

Systematic Rollback

Reverts assignments on constraint violations.

EXECUTION PIPELINE

Step-by-Step Backtracking Routine

01

Base Case Check

If all V vertices colored, return true.

02

Try Colors 1..M

Iterate candidate colors for vertex v.

03

Safety Validation

Verify no adjacent node shares color.

04

Recurse or Reset

Recurse to v+1; reset color on failure.

HELPER FUNCTION

Checking Valid Colors: isSafe()

```
bool isSafe(int v, int graph[V][V], int color[], int c) {
  // Check all adjacent vertices
  for (int i = 0; i < V; i++) {
    if (graph[v][i] == 1 && color[i] == c)
      return false;
  }
  return true;
}
```

Input Parameters

Current vertex v, adjacency matrix, assignment state, proposed color c.

Time Complexity

O(V) checks against row v in adjacency matrix.

CORE ALGORITHM

The Recursive Solver: graphColoringUtil()

```
bool graphColoringUtil(int v, int m, int color[]) {
  if (v == V) return true; // Solved all vertices

  for (int c = 1; c <= m; c++) {
    if (isSafe(v, graph, color, c)) {
      color[v] = c;
      if (graphColoringUtil(v + 1, m, color))
        return true;
      color[v] = 0; // Backtrack on failure
    }
  }
  return false;
}
```

Assignment Step

Optimistically assigns color c to vertex v upon safety check.

Backtracking Step

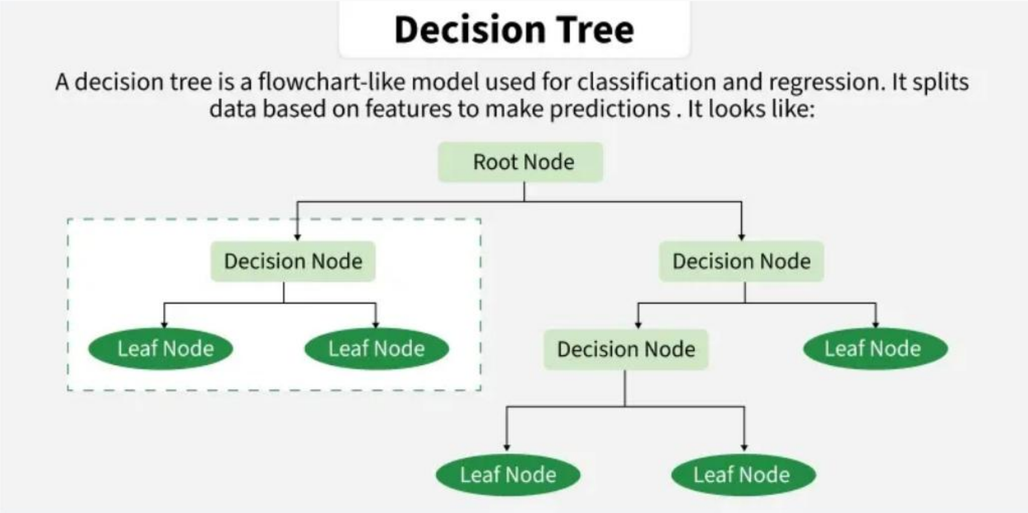
color[v] = 0 resets state when subtree yields no feasible solutions.

Termination

Returns true when depth reach V, confirming validity.

VISUALIZATION

State-Space Tree & Pruning



Branch & Bound Effect

The state-space tree possesses a branching factor of **m** and depth of **V**.

Pruned Subtrees

Subtrees rooted at conflicting states are pruned instantly, avoiding m^(V - d) redundant evaluations.

PERFORMANCE METRICS

Computational Complexity

$O(m^V)$

Worst-Case Time

Upper bound when entire state space is explored.

$O(V)$

Auxiliary Space

Recursion stack frame and color array storage.

Practical Runtime: Significantly faster than $O(m^V)$ on average due to aggressive pruning of invalid adjacency states early in traversal.

TECHNIQUE TRADEOFFS

Comparative Paradigm Analysis

Method	Optimality	Worst-Case Time	Key Advantage
Backtracking	**Guaranteed Exact**	`O(m^V)`	Simple, exact constraint satisfaction
Greedy Coloring	**Sub-optimal / Approx**	`O(V^2 + E)`	Extremely fast on massive graphs
Welsh-Powell	**Heuristic Upper Bound**	`O(V log V + E)`	Orders vertices by degree first

PRACTICAL USES

Real-World Applications

Register Allocation

Assigning CPU registers to program variables via interference graphs.

Frequency Assignment

Allocating non-interfering wireless cellular frequencies to towers.

Timetable Scheduling

Preventing time-slot clashes between concurrent exams or meetings.

Map Coloring (4-Color Theorem)

Distinguishing contiguous geographic regions with bounded palettes.

KEY TAKEAWAY

Exact Solutions via Systematic Pruning

Backtracking transforms an intractable brute-force exponential search into an effective exact solver by dynamically eliminating invalid subtrees.

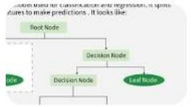
Depth-First Traversal | Early Invalidation | Optimal State Rollback

Image Sources



https://ona-book.org/gitbook/_main_files/figure-html/workfrance-additional-1.png

Source: ona-book.org



https://media.geeksforgeeks.org/wp-content/uploads/20251216110915386263/decision_tree.webp

Source: www.geeksforgeeks.org