# Lab Assignment: Solving Uniform Random 3■SAT — Hill-Climbing, Beam Search, VND

Author: Tishu Verma (replace with your names and roll numbers)

## Abstract

This report describes implementations and an experimental study of local-search algorithms applied to uniform random 3■SAT instances. We implement three metaheuristics — Hill■Climbing (HC), Beam Search (BS) with beam widths 3 and 4, and Variable■Neighborhood■Descent (VND) using three neighborhood functions — and evaluate two heuristic functions. Performance is measured by **penetrance**, defined here as the proportion of instances solved within a given iteration/time budget, together with average number of flips/iterations and time to solution.

## Introduction

Local-search methods are common tools for large random SAT instances. This lab compares variant local-search strategies on uniform random 3■SAT (k=3) instances across a sweep of clause counts `m` and variable counts `n`. The goal is to measure how algorithm design choices (search strategy, neighborhoods, heuristic) affect penetrance and efficiency.

## Problem Definition

Given multiple uniform random 3■SAT instances (fixed clause length k=3) generated for combinations of `n` (variables) and `m` (clauses), implement algorithms to attempt to satisfy each instance and report:

- Penetrance (success rate): fraction of instances solved within a fixed budget.

- Average flips/iterations for successful runs.

- Runtime statistics (optional depending on environment).

Algorithms to implement:

- Hill■Climbing (restarts allowed)

- Beam Search with beam widths 3 and 4

- Variable■Neighborhood■Descent (VND) with three neighborhood functions

Heuristics to compare (two variants):

1. **Greedy break-count heuristic** (minimize number of currently unsatisfied clauses after a flip) — classic GSAT-like choice.

2. **Score with tabu■like tie■breaker**: use break-count, but break ties by the count of occurrences of variable in unsatisfied clauses (higher occurrence favored), or use random tie■break.

We measure penetrance across several random instances per (m,n) pair and compare algorithms and heuristics.

## Definitions and Metrics

- **Clause-to-variable ratio (α)** = m / n. Phase transition around certain α values (for k=3 around α ≈ 4.26) often affects difficulty.

- **Penetrance** (in this report): the percentage (or fraction) of instances solved within `max_flips` flips or `max_time` seconds.

- **Average flips to success**: average number of flips among successful runs.

- **Instance budget**: number of random instances generated per (n,m) pair (e.g., 50 or 100).

## Neighborhood Functions for VND

VND cycles through neighborhood structures; for 3■SAT we use three neighborhoods defined for a current full assignment:

1. **N1 — Single■flip neighborhood:** all assignments that differ by flipping a single variable.

2. **N2 — Double■flip neighborhood:** all assignments that differ by flipping two distinct variables simultaneously (useful to escape local minima of N1).

3. **N3 — Clause■guided neighborhood:** choose an unsatisfied clause at random and flip one of the three variables in it (also effectively a restricted single flip but focused on unsatisfied clauses). N3 can be implemented as a small targeted neighborhood.

The VND algorithm tries to improve current solution by exploring N1; if no improvement it moves to N2; if improvement found it restarts from N1, etc.

## Algorithm Descriptions (High Level)

### Hill■Climbing (HC)

- Start with random assignment for all n variables.

- Repeat up to `max_flips`:

- If assignment satisfies all clauses → success.

- Evaluate candidate flips (usually single■variable flips) using heuristic (e.g. break-count) and pick the best; if multiple, tie-break with heuristic 2.

- Perform flip and continue.

- Optionally allow `restarts` — after `max_flips` without success, reinitialize and try again up to `max_restarts`.

### Beam Search (BS)

- Maintain a beam of B best assignments (beam width B = 3 or 4).

- Initialize beam with B random assignments.

- For each iteration up to `max_iters`:

- Expand each beam member by generating its neighbor assignments (single■flip neighbors or small neighborhood set).

- Evaluate and select top B distinct assignments for the next beam using the heuristic score (lower number of unsatisfied clauses preferred).

- If any assignment satisfies all clauses → success.

### Variable■Neighborhood■Descent (VND)

- Start with random assignment.

- For neighborhood index i = 1..3:

- Apply local search restricted to Ni until no improvement is found or a small subbudget is exhausted.

- If improvement is found, go back to i = 1. Otherwise increment i.

- If global `max_flips` reached or satisfied, stop.

## Pseudocode

(Pseudocode included in the document for each algorithm — see the code section for concrete Python implementations.)

## Python Implementation

... (Full Python code as given in the report) ...