# Artificial Intelligence Lab Report 3

**Anuj Saha**
*202351010*
*BTech CSE*
*IIIT, Vadodara*

**G. Nikhil**
*202351037*
*BTech CSE*
*IIIT, Vadodara*

**Divyanshu Ghosh**
*202351036*
*BTech CSE*
*IIIT, Vadodara*

***Abstract*—This report presents the implementation and evaluation of uniform random k-SAT problem generation and non-classical search algorithms for 3-SAT solving. We developed a generator creating instances with specified clause length $k$, clauses $m$, and variables $n$, ensuring distinct literals per clause. For solving, we implemented Hill-Climbing with random restarts, Beam Search (widths 3, 4), and Variable Neighborhood Descent with three neighborhoods. Two heuristics guide the search: h1 (unsatisfied clauses count) and h2 (unsatisfied clauses plus satisfaction penalty). Experiments across $m/n \in \{2, 4, 6\}$ for $n = 10, 20$ reveal VND's superiority (80% solved rate at $m/n = 4$) and Beam-4's efficiency for easy problems (1.6 average steps). Penetrance analysis shows VND's exploration efficiency (0.0022) despite higher per-step cost. The study highlights algorithm trade-offs and the satisfiability phase transition near $m/n \approx 4 - 6$.**

***Index Terms*—k-SAT, 3-SAT, Local Search, Hill-Climbing, Beam Search, Variable Neighborhood Descent, Heuristic Functions**

## 1. Introduction

The Boolean Satisfiability Problem (SAT) is a key problem in artificial intelligence, computational complexity, and optimization. The k-SAT version, where every clause consists of exactly $k$ literals, is a narrow enough problem that provides a controlled environment, to observe the behavior of an algorithm as the density of constraints vary. This laboratory exercise will involve generating uniform random k-SAT instances and using them to evaluate local search approaches to solve 3-SAT.

Uniform random k-SAT generation allows us to obtain well-characterized distributions of problems so that we can systematically benchmark algorithms. The generator has to be careful that clauses do not repeat variables and it has to assign litter polarity uniformly to obtain instances which represent the default fixed clause length model.

As for the solver component, we use three non-classical search algorithms: Hill-Climbing with random restarts, Beam Search with configurable beam widths, and Variable Neighborhood Descent (VND) with deliberate exploration of the neighborhoods. Each of these is a different type of local search: greedy ascent, population-based exploration, and systematic neighborhood expansion.

We utilize two heuristic functions to drive the search. The first (h1) measures how many clauses are unsatisfied, offering a straightforward distance-to-goal function. The second function (h2) includes a penalty for satisfied clauses with fewer than three true literals, punishing weaker solutions.

The experimental design systematically varies the clause-to-variable ratio $m/n$ across easy ($m/n = 2$), moderate ($m/n = 4$), and hard ($m/n = 6$) regimes for $n = 10, 20$ variables. Performance evaluation focuses on three metrics: solved rate, computational steps, and penetrance (search efficiency relative to explored space). The results provide critical insights into algorithm behavior across the satisfiability phase transition and highlight practical trade-offs in local search design.

## 2. Problem Statement

The assignment has two main tasks to achieve: First, to write a uniform random k-SAT generator that accepts k (clause length), m (number of clauses), and n (number of variables) requirements that would generate a valid instance of k-SAT where each clause has exactly k distinct variables with a randomly chosen positive or negative literal; and second, to implement three local search algorithms to compare the performance in terms of solved rate, efficiency of computation, and search penetrance for solving 3-SAT problems with different densities in constraints.

### 2.1. k-SAT Generation Requirements

The generator must satisfy:

- Each clause contains exactly $k$ *distinct* variables from $\{1, 2, \ldots, n\}$
- Independent random polarity assignment ($+1$ or $-1$) for each literal
- Uniform sampling across all valid k-SAT formulas of size $(k, m, n)$
- No duplicate variables within individual clauses

### 2.2. 3-SAT Solver Requirements

For $k = 3$, implement:

- **Hill-Climbing**: Greedy local search with random restarts/ (100 attempts, 1000 flips maximum)
- **Beam Search**: Fixed-width beam (widths 3, 4) exploring single-variable flip successors.
- **Variable Neighborhood Descent**: Three neighborhoods ($k = 1, 2, 3$ simultaneous flips)

We design two heuristic functions and conduct experiments for $m/n \in \{2, 4, 6\}$ with $n \in \{10, 20\}$, reporting:

- **Solved Rate**: Percentage of satisfiable instances successfully solved.
- **Computational Steps**: Total iterations or state evaluations.
- **Penetrance**: $\frac{\text{steps}}{\text{estimated explored space size}}$

## 3. Methodology

### 3.1. Uniform Random k-SAT Generation

The k-SAT generator produces uniform random instances following Algorithm 1.

**Algorithm 1** Uniform Random k-SAT Generation

```
Require: $k$: clause length, $m$: number of clauses, $n$: number of variables
Ensure: Formula: list of $m$ clauses, each with $k$ literals
1: Initialize empty formula $F = []$
2: for $i = 1$ to $m$ do
3:     Sample $k$ distinct variables $V \subset \{1, 2, \ldots, n\}$
4:     For each $v \in V$: assign polarity $p_v \in \{-1, +1\}$ randomly
5:     Create clause $C = [p_1 v_1, p_2 v_2, \ldots, p_k v_k]$
6:     $F \leftarrow F \cup \{C\}$
7: end for
8: return $F$
```

The algorithm ensures uniformity through without-replacement variable sampling and independent polarity assignment. For $k = 3$, $n = 10$, $m = 20$, output includes clauses such as $[2, -4, 3], [-2, 3, -5]$, representing $x_2 \lor \neg x_4 \lor x_3$ and $\neg x_2 \lor x_3 \lor \neg x_5$.

### 3.2. Local Search Algorithms

**3.2.1. Hill-Climbing with Random Restarts.** Hill-Climbing performs greedy ascent from random initial assignments, evaluating all single-variable flip neighbors and selecting the best improvement.The algorithm explores the local neighborhood exhaustively at each step, comparing the heuristic values of all possible single-bit flips before committing to a move. When multiple flips yield equal improvement, one is selected randomly to avoid bias. Random restarts help escape local optima by reinitializing the search from fresh starting points. (Algorithm 2).

**Algorithm 2** Hill-Climbing with Random Restarts

```
Require: Formula $F$, variables $n$, heuristic $h$, max_tries=100, max_flips=1000
Ensure: Assignment $A$, steps, solved status
1: $steps \leftarrow 0$, $best\_A \leftarrow \emptyset$
2: for $t = 1$ to max_tries do
3:     $A \leftarrow$ random assignment $[False, True]^n$
4:     for $f = 1$ to max_flips do
5:         $steps \leftarrow steps + 1$
6:         if $h(F, A) = 0$ then
7:             return $A$, $steps$, true
8:         end if
9:         $best\_h \leftarrow h(F, A)$, $best\_vars \leftarrow []$
10:        for $v = 1$ to $n$ do
11:            Temporarily flip $A[v]$
12:            if $h(F, A) < best\_h$ then
13:                $best\_h \leftarrow h(F, A)$, $best\_vars \leftarrow [v]$
14:            else if $h(F, A) = best\_h$ then
15:                $best\_vars.append(v)$
16:            end if
17:            Revert $A[v]$
18:        end for
19:        if $best\_h < h(F, A)$ then
20:            Select random $v \in best\_vars$, flip $A[v]$
21:        else
22:            break (local optimum)
23:        end if
24:    end for
25: end for
26: return $\emptyset$, $steps$, false
```

**3.2.2. Beam Search.** Beam Search maintains a fixed-size set of promising assignments, generating $n$ single-flip successors per beam element and retaining the top $b$ by heuristic value (Algorithm 3).

**Algorithm 3** Beam Search

```
Require: Formula $F$, variables $n$, beam width $b$, heuristic $h$, max_steps=1000
Ensure: Assignment $A$, steps, solved status
1: Initialize beam with $b$ random assignments
2: for $step = 1$ to max_steps do
3:     for $A \in$ current beam do
4:         if $h(F, A) = 0$ then
5:             return $A$, $step$, true
6:         end if
7:     end for
8:     $successors \leftarrow \emptyset$
9:     for $A \in$ current beam do
10:        for $v = 1$ to $n$ do
11:            $A' \leftarrow$ copy of $A$ with $v$ flipped
12:            $successors.append(A')$
13:        end for
14:    end for
15:    Sort $successors$ by $h(F, A')$ ascending
16:    Update beam $\leftarrow$ top $b$ successors
17: end for
18: return $\emptyset$, max_steps, false
```

#### 3.2.3. Variable Neighborhood Descent.

VND systematically explores neighborhoods of increasing size $k = 1, 2, 3$, where neighborhood $k$ contains all assignments differing in exactly $k$ variables simultaneously.(Algorithm 4).

**Algorithm 4** Variable Neighborhood Descent

```
Require: Formula $F$, variables $n$, heuristic $h$, max_tries=10
Ensure: Assignment $A$, steps, solved status
 1: $k_{max} \leftarrow 3$, $steps \leftarrow 0$
 2: for $t = 1$ to max_tries do
 3:     $A \leftarrow$ random assignment $[False, True]^n$
 4:     $k \leftarrow 1$
 5:     while $k \leq k_{max}$ do
 6:         $steps \leftarrow steps + 1$
 7:         if $h(F, A) = 0$ then
 8:             return $A$, $steps$, true
 9:         end if
10:         $best\_h \leftarrow h(F, A)$, $best\_flips \leftarrow \emptyset$
11:         for each combination $C \subset \{1, \ldots, n\}$ with $|C| = k$ do
12:             Temporarily apply all flips in $C$ to $A$
13:             if $h(F, A) < best\_h$ then
14:                 $best\_h \leftarrow h(F, A)$, $best\_flips \leftarrow C$
15:             end if
16:             Revert flips in $C$
17:         end for
18:         if $best\_h < h(F, A)$ then
19:             Apply flips in $best\_flips$ to $A$
20:             $k \leftarrow 1$ (improvement found)
21:         else
22:             $k \leftarrow k + 1$ (expand neighborhood)
23:         end if
24:     end while
25: end for
26: return $\emptyset$, $steps$, false
```

### 3.3. Heuristic Functions

Two heuristic functions guide the search process:

**h1 - Unsatisfied Clauses (Direct Distance):**

$$h_1(F, A) = \sum_{C \in F} \mathbb{I}\left[\bigwedge_{l \in C} \neg eval(l, A)\right] \tag{1}$$

**h2 - Weighted Satisfaction Penalty:**

$$\begin{aligned} h_2(F, A) = \sum_{C \in F} \Bigg\{ & \mathbb{I}\left[\bigwedge_{l \in C} \neg eval(l, A)\right] \\ & + 0.1 \times (3 - |\{l \in C : eval(l, A) = True\}|) \\ & \times \mathbb{I}\left[\bigvee_{l \in C} eval(l, A)\right] \Bigg\} \end{aligned} \tag{2}$$

where $eval(l, A)$ evaluates literal $l$ under assignment $A$, and $\mathbb{I}[\cdot]$ denotes the indicator function. h2 penalizes clauses satisfied with fewer than three true literals, favoring more robust solutions.

### 3.4. Literal Evaluation

Literal evaluation handles both positive and negated variables:

**Algorithm 5** Literal Evaluation

```
Require: Literal $l$, assignment $A$
Ensure: Truth value of $l$ under $A$
1: $var \leftarrow |l|$ (absolute variable index)
2: $val \leftarrow A[var]$ (variable assignment)
3: if $l < 0$ then
4:     $val \leftarrow \neg val$ (negation)
5: end if
6: return $val$
```

### 3.5. Experimental Design

We generated 5 instances for each parameter combination:

- $n \in \{10, 20\}$ (number of variables)
- $m \in \{2n, 4n, 6n\}$ (easy, moderate, hard constraint density)
- Algorithms: Hill-Climbing, Beam-3, Beam-4, VND
- Heuristics: h1, h2

Performance metrics include:

- **Solved Rate**: $\frac{\text{successful solutions}}{\text{total instances}} \times 100\%$
- **Steps**: Total iterations or state evaluations until convergence
- **Penetrance**: $P = \frac{\text{steps}}{N}$, where $N$ estimates explored space:
  - Hill-Climbing: $N = \text{steps} \times n$
  - Beam-$b$: $N = \text{steps} \times b \times n$
  - VND: $N = \text{steps} \times \frac{n^3}{18}$ (average $\binom{n}{k}$ for $k = 1, 2, 3$)

## 4. Results

### 4.1. Instance Generation Validation

The k-SAT generator successfully produced uniform random instances. For $k = 3$, $n = 10$, $m = 20$, sample output includes:

```
[[-4, 2,  1 ],[-3, -5, -1],[-3,  1,  2],
[-3, -1, -4 ],[-5, -3,  4],[-5, -1, -2],
[ 2,  1,  5 ],[ 4,  2,  3],[ 2, -3, -1],
                [ 5,  1, -3]]
```

Each clause contained exactly 3 distinct variables with random polarities, satisfying all uniformity requirements.

### 4.2. Solved Rate Performance

Table 1 presents solved rates across problem densities. All algorithms achieve 100% success for easy problems ($m/n = 2$). At moderate density ($m/n = 4$), performance diverges significantly.

TABLE 1. Solved Rate Analysis

| Algorithm | $n = 10$ (m/n=4) | $n = 20$ (m/n=4) | Hard (m/n=6) |
|---|---|---|---|
| Hill-Climbing (h1) | 100% | 80% | ≤20% |
| Hill-Climbing (h2) | 100% | 80% | ≤20% |
| Beam-3 (h1) | 100% | 40% | ≤20% |
| Beam-3 (h2) | 60% | 20% | ≤20% |
| Beam-4 (h1) | 100% | 20% | ≤20% |
| Beam-4 (h2) | 80% | 20% | ≤20% |
| VND (h1) | 100% | 80% | ≤20% |
| VND (h2) | 100% | 60% | ≤20% |

VND and Hill-Climbing maintain 100% success for $n = 10$, $m/n = 4$, while Beam-3 h2 degrades to 60%. For $n = 20$, $m/n = 4$, VND h1 achieves 80% versus Beam-3 h1's 40%. The $m/n = 6$ regime shows $\leq 20\%$ success across algorithms, indicating the satisfiability phase transition.

## 4.3. Computational Efficiency

Table 2 shows average steps for solved instances at different difficulty levels. Beam-4 requires fewest steps for easy problems while Hill-Climbing needs most.

TABLE 2. Computational Steps Analysis

| Algorithm | Easy (m/n=2) | Mod. (m/n=4) | Hard (m/n=6) |
|---|---|---|---|
| Hill-Climbing (h1) | 3.8 | 12.4 | 11.0 |
| Hill-Climbing (h2) | 2.6 | 15.6 | 14.0 |
| Beam-3 (h1) | 2.2 | 5.6 | 3.0 |
| Beam-3 (h2) | 2.2 | 5.5 | 3.0 |
| Beam-4 (h1) | 1.6 | 3.6 | – |
| Beam-4 (h2) | 1.8 | 4.0 | 3.0 |
| VND (h1) | 2.6 | 8.2 | 3.0 |
| VND (h2) | 3.8 | 9.2 | 4.0 |

For $n = 20$, $m/n = 4$, Hill-Climbing requires 31-37.5 steps versus Beam-3's 4-5.5 steps. Unsolved instances hit limits: 1000 steps (Beam), 300-1000 steps (Hill-Climbing), 55-155 steps (VND).

## 4.4. Penetrance Analysis

Table 3 shows search efficiency relative to explored space. Hill-Climbing exhibits highest penetrance due to linear neighborhood size ($n$). VND shows lowest penetrance from exploring $\mathcal{O}(n^3)$ combinations.

TABLE 3. Penetrance Analysis ($P = \frac{\text{STEPS}}{N}$)

| Algorithm | $n = 10$ | $n = 20$ |
|---|---|---|
| Hill-Climbing | 0.1000 | 0.0500 |
| Beam-3 | 0.0333 | 0.0167 |
| Beam-4 | 0.0250 | 0.0125 |
| VND | 0.0180 | 0.0022 |

Lower penetrance indicates more efficient search relative to total space explored. Beam-4's intermediate penetrance (0.0125-0.0250) balances exploration breadth and focus.

## 4.5. Heuristic Comparison

h1 consistently outperforms h2 in solved rate, particularly for Beam Search (100% vs 60% for $n = 10$, $m/n = 4$). However, h2 reduces steps in some cases (2.6 vs 3.8 for Hill-Climbing, $n = 10$, $m/n = 2$). The satisfaction penalty in h2 benefits easy problems but misguides dense constraint landscapes.

## 5. Discussion

The findings from the experiments show distinctive behaviors of the algorithm in different regimes of problem difficulty:

- In the case of an easy problem ($m/n = 2$), all algorithms demonstrate a corresponding perfect (100%) success rate, yet Beam-4 outperforms the others in efficiency, averaging 1.6 steps vs 3.8 steps for Hill-Climbing. This shows an advantage for beam search when solutions are easily found in the state space.
- Moderate constraint density ($m/n = 4$), showed pronounced divergence in performance. VND and Hill-Climbing maintained high success rates (80-100%) while Beam Search degraded significantly for larger problems ($n = 20$: 20-40% success). As the state space dimensionality increases, the sensitivity to problem dimensionality suggests that the fixed beam width becomes insignificant.
- The $m/n = 6$ regime clearly demonstrates the satisfiability phase transition, with all algorithms achieving $\leq 20\%$ success rates. This aligns with theoretical predictions, that the 3-SAT instances become unsatisfiable beyond the critical ratio $\alpha_c \approx 4.26$, confirming the generator produces representative problem distributions.
- Penetrance analysis provides insight into exploration efficiency. Despite higher per-step computational cost, VND exhibits the lowest penetrance (0.0022 for $n = 20$), reflecting effective navigation of complex landscapes through systematic neighborhood expansion. The $\mathcal{O}(n^3)$ neighborhood exploration proves more efficient than Hill-Climbing's linear search despite examining more states per iteration.
- A heuristic comparison demonstrates the robustness of h1 for all densities and serves as a reliable distance-to-goal measure. The h2 penalty term provides marginal benefit for easy problems by advantage, however, it increases failure rates at denser constraints given a sub-optimal 0.1 weight. These trade-offs could potentially be balanced by adaptive weighting schemes.
- The implementation demonstrates that non-classical search is an effective technique for solving 3-SAT problems, and produces a reproducible benchmark for k-SAT problems that is necessary for developing algorithms. The phase transition behavior observed here can also have consequences for real-world constraint satisfaction problems, where density plays a vital role in the solvability of the problem.

## 6. Conclusion

In this laboratory assignment, we designed a uniform random k-SAT generator and a complete evaluation framework for local search algorithms. Our generator produces well-characterized instances that meet all uniformity requirements, while the implementations of local search solvers exhibit different algorithm behaviors across the satisfiability spectrum.

Key findings include:

- **VND superiority** for moderate difficulty ($m/n = 4$) achieving 80-100% success rates.
- **Beam-4 efficiency** for easy problems ($m/n = 2$) requiring minimal computational steps.
- **h1 reliability** across all constraint densities versus h2's selective benefits.
- **Low penetrance** of VND (0.0022) highlighting exploration efficiency despite higher cost per step.
- **Phase transition** near $m/n \approx 4 - 6$ confirming theoretical predictions.

The systematic experimental design gives a robust performance characterization across algorithm types and problem regimes. Future work could explore adaptive beam widths, dynamic locality selection strategies, and learned heuristic functions to improve performance across the entire satisfiability landscape. Furthermore, hybrid methods that combine local searching with organized methods may solve over-constrained instances around the phase transition.

The implementation establishes a solid foundation for continued research in constraint satisfaction, with applications extending beyond 3-SAT to broader AI planning, scheduling, and optimization domains.

The complete implementation is available at: https://github.com/AnujSaha0111/CS307_Lab_Submissions/tree/main/Submission_3

## References

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2016.

[2] D. Khemani, *A First Course in Artificial Intelligence*. New Delhi, India: McGraw Hill Education, 2013.

[3] H. H. Hoos and T. Stützle, *Search Algorithms for Engineering Optimization*. Berlin, Germany: Springer, 2005.

[4] D. Mitchell, "Random 3-satisfiability: Phase transition and complexity," *Journal of Physics A: Mathematical and General*, vol. 38, no. 36, pp. R187–R200, 2005.