DEPARTMENT OF COMPUTER AND INFORMATION SYSTEMS ENGINEERING
BACHELORS IN COMPUTER SYSTEMS ENGINEERING
Course Code and Title: CS-323 Artificial Intelligence
Complex Engineering Problem
TE Batch 2023, Fall Semester 2025
TERM PROJECT
Grading Rubric

Group Members:

Student No.	Name	Roll No.
S1	**Zain Waseem**	**099**
S2	**Sufyan Ali**	**134**
S3	**Moiz Haider**	**137**

CRITERIA AND SCALES				**Marks Obtained**		
				S1	**S2**	**S3**
Criterion 1: Implementation and performance of search method 1. (CPA-1, CPA-2, CPA-3) **[4 marks]**						
1	2	3	4			
The algorithm is incorrectly implemented, produces wrong outputs, or fails to execute on test boards.	The algorithm is partially implemented, produces limited or inconsistent results.	The algorithm is correctly implemented and produces mostly correct outputs with reasonable efficiency.	The algorithm is correctly and efficiently implemented, produces accurate results consistently, and demonstrates clear understanding of the AI search technique used.			
Criterion 2: Implementation and performance of search method 2. (CPA-1, CPA-2, CPA-3) **[4 marks]**						
1	2	3	4			
The CSP formulation or backtracking implementation is incorrect or incomplete.	The CSP model is partially correct but produces limited or inaccurate outputs.	The CSP formulation is correct, producing mostly accurate results with acceptable efficiency.	The CSP solver is accurately and efficiently implemented, with well-defined constraints and consistent results showing strong understanding of CSP concepts.			
Criterion 3: Comparison, evaluation, and analysis of both methods (CPA-3) **[4 marks]**						
1	2	3	4			
No meaningful comparison or analysis; results not discussed or evaluated.	Basic comparison made but lacks depth, clarity, or quantitative support.	Clear comparison with relevant metrics (e.g., success rate, runtime) and reasonable interpretation.	Comprehensive and insightful comparison using quantitative and qualitative metrics, with critical discussion on performance, efficiency, and limitations.			
Criterion 3: Adherence of report to the given format and requirements. **[4 marks]**						
1	2	3	4			
The report does not contain the required information and is formatted poorly.	The report contains the required information only partially but is formatted well.	The report contains all the required information but is formatted poorly.	The report contains all the required information and completely adheres to the given format.			
Criterion 4: Individual and team contribution. (CPA-2) **[4 marks]**						
1	2	3	4			
The student did not contribute meaningfully to project tasks.	The student contributed partially to assigned tasks with limited collaboration.	The student contributed adequately and met assigned goals.	The student demonstrated active participation, initiative, and significant contribution beyond expectations.			

Final Score = ______________________ Teacher's Signature: ____________________

Table of Contents

1. Problem Representation 3

1.1 Variables 3

1.2 Domain 3

1.3 State Representation: 3

1.4 Constraints 3

1.5 State Space Size 4

2. Search Algorithms Implemented 4

Logic: 4

2.2 CSP Backtracking with MRV + Forward Checking 5

Logic: 5

3. Additional Features Implemented 6

1. Conflict-Highlighting Visualization 6

2. Generalized N-Queens (Dynamic N) 6

4. Graphical Comparision 6

5. Performance Discussion 8

5.1 Hill-Climbing 8

5.2 CSP (Backtracking + MRV + Forward Checking) 8

5.3 Suggestions for Improvement 9

6. Test Case Screenshots 9

8. Generative AI Use Declaration 11

Instructions 11

1. Problem Representation

The N-Queens problem requires placing **N queens on an N×N chessboard** such that:

- No two queens are in the same **row**
- No two queens are in the same **column**
- No two queens are on the same **diagonal**

1.1 Variables

Each column is treated as a variable:

X0, X1, X2, …, X(N–1)

1.2 Domain

For each variable Xi, domain is:

Di = {0, 1, 2, …, N–1}

1.3 State Representation:

A state is stored as a **list**:

state = [row_of_col0, row_of_col1, …]

Example for N=8:

[0, 4, 7, 5, 2, 6, 1, 3]

1.4 Constraints

Two queens conflict if:

Same row:
state[i] == state[j]

Same diagonal:
abs(state[i] - state[j]) == abs(i - j)

1.5 State Space Size

For N queens:

Total possible states = $\mathbf{N^n}$
(Each column can choose N rows)

2. Search Algorithms Implemented

We implemented two approaches:

1. **Hill-Climbing with Random Restarts** (Local Search)
2. **CSP Backtracking with MRV + Forward Checking** (Complete Search)

1.1 Hill-Climbing with Random Restarts:

Logic:

- Start from a **random state**
- At each step, move to the **neighbor with the lowest conflicts**
- If stuck in local minimum → **restart randomly**
- Stop when either:
    - Zero-conflict solution found
    - Max restarts reached

Pseudocode:

```
function HILL-CLIMBING(max_restarts):

  for restart in 1 to max_restarts:

    state ← RANDOM_STATE()

  loop:  neighbor ← BEST_NEIGHBOR(state)

      if neighbor.conflicts ≥ state.conflicts:

        break     # local minimum

     state ← neighbor
```

```
        if state.conflicts == 0:
            return state, success=True
    return last_state, success=False
```

2.2 CSP Backtracking with MRV + Forward Checking

Logic:

- Use **backtracking search**
- Select variable using **MRV (Minimum Remaining Values)**
- Use **Forward Checking** to prune inconsistent future values
- Guaranteed to find solution if one exists

Pseudocode:

```
function BACKTRACK(assignment, domains):
    if assignment is complete:
        return assignment
    var ← variable with smallest domain (MRV)
    for each value in domain[var]:
        if value is consistent:
            new_assignment ← assign(var, value)
            new_domains ← FORWARD_CHECK(domains, var, value)
            result ← BACKTRACK(new_assignment, new_domains)
            if result ≠ failure:
                return result
    return failure
```

3. Additional Features Implemented

1. Conflict-Highlighting Visualization

A custom board display was implemented where queens involved in conflicts are automatically shown in **red**, while safe queens appear in **black**.
The function checks row and diagonal conflicts for each queen and colors them accordingly.

Why useful?

- Quickly shows whether a solution is valid
- Helps understand where Hill-Climbing gets stuck
- Makes screenshots and analysis clearer

2. Generalized N-Queens (Dynamic N)

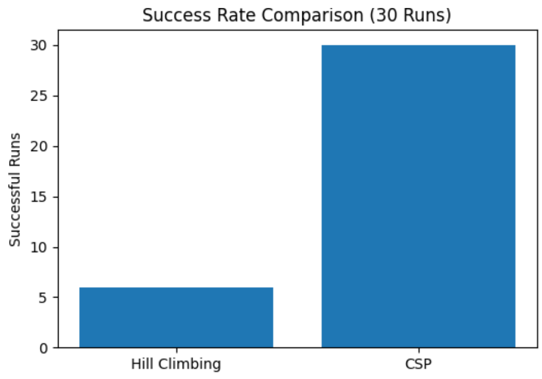
The solver was extended to handle **any board size N**, not just 8-Queens.
A single global variable N controls the entire system: generation, visualization, HC, CSP, and all experiments.

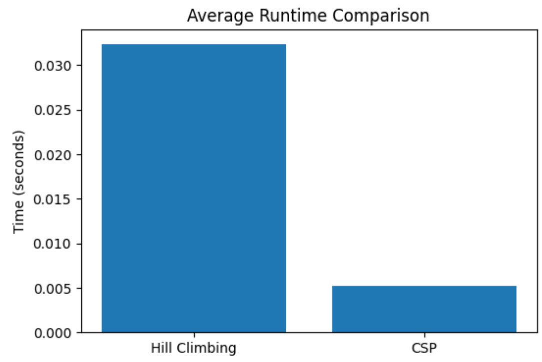
Why useful?

- Demonstrates scalability of both algorithms
- Allows testing performance on N = 8, 10, 12, 16, etc.
- Shows how solution difficulty changes with board size

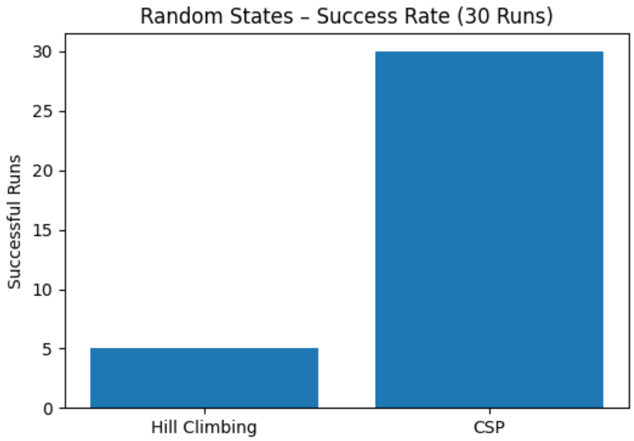
4. Graphical Comparision

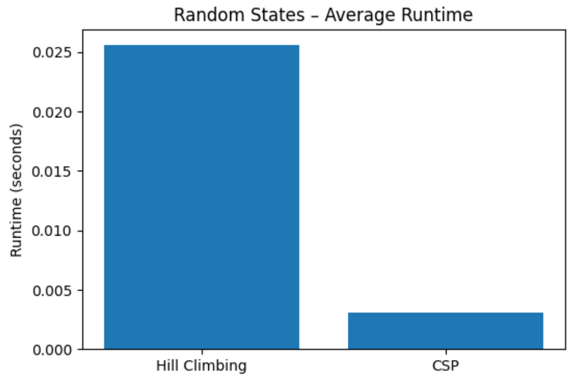
Performance graphs:



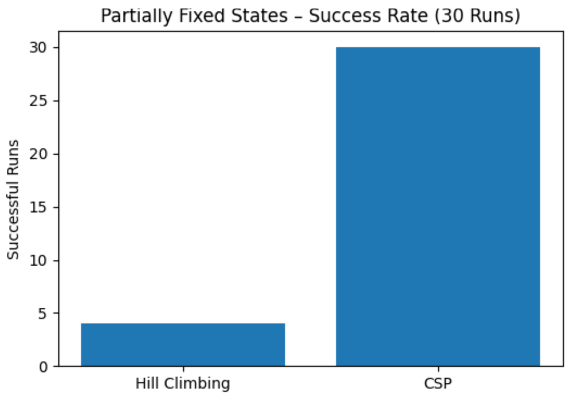


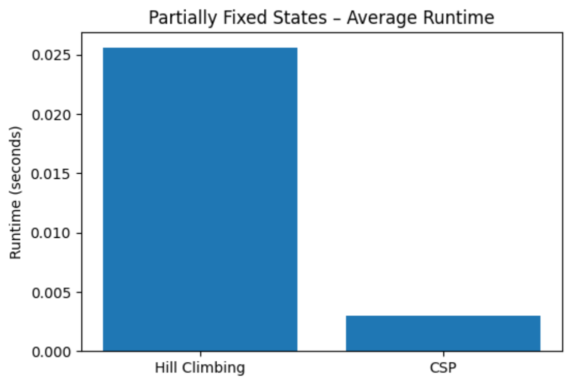
RANDOM STATES GRAPHS:



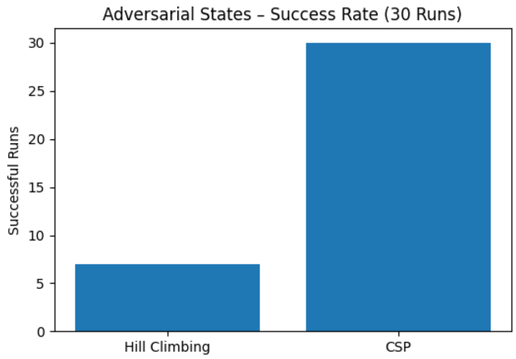


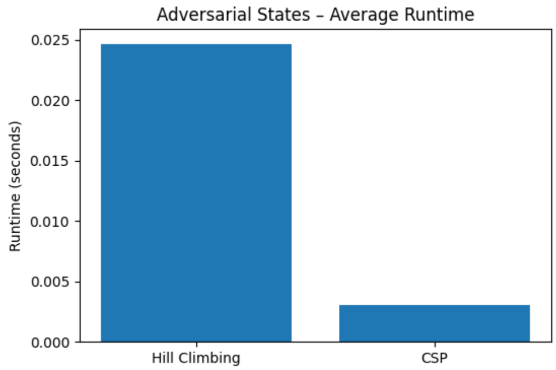
PARTIALLY FIXED STATES GRAPHS:





ADVERSARIAL STATES GRAPHS:





5. Performance Discussion

5.1 Hill-Climbing

Hill-Climbing is extremely **fast** and performs well on **random initial states**.
However, because it is a **local search** technique, it often gets stuck in:

- **Partially fixed states** (where early constraints reduce flexibility)
- **Adversarial states** (many queens conflicting intentionally)

It is an **incomplete algorithm**:
even if a solution exists, Hill-Climbing may fail to reach it, regardless of the number of steps or restarts.

Strengths:

- Very fast
- Easy to implement
- Works well for small N

Weaknesses:

- Local minima
- Plateaus
- Not guaranteed to find solution

5.2 CSP (Backtracking + MRV + Forward Checking)

The CSP solver is a **complete algorithm**, meaning it **always finds a solution** if one exists.
It performs consistently across:

- **Random states**
- **Partially fixed states**
- **Adversarial states**

It uses MRV and Forward Checking, which reduce the search space and avoid exploring impossible branches.

Strengths:

- Guaranteed to find a valid solution
- Works on all test categories
- Robust and reliable

Weaknesses:

- Higher runtime than Hill-Climbing
- Less scalable for very large N
- More computationally expensive

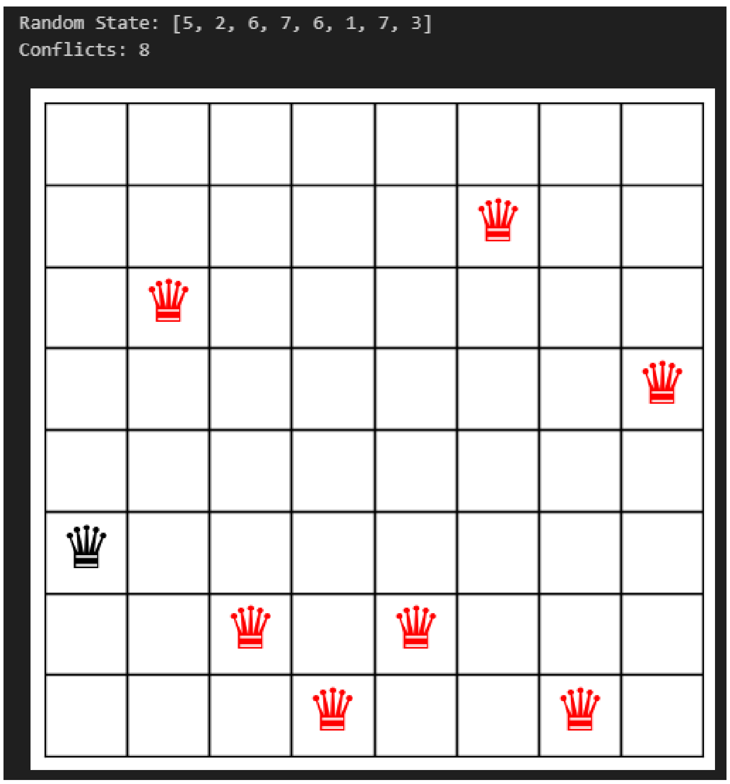
5.3 Suggestions for Improvement

Several enhancements can improve the solver's performance and robustness:

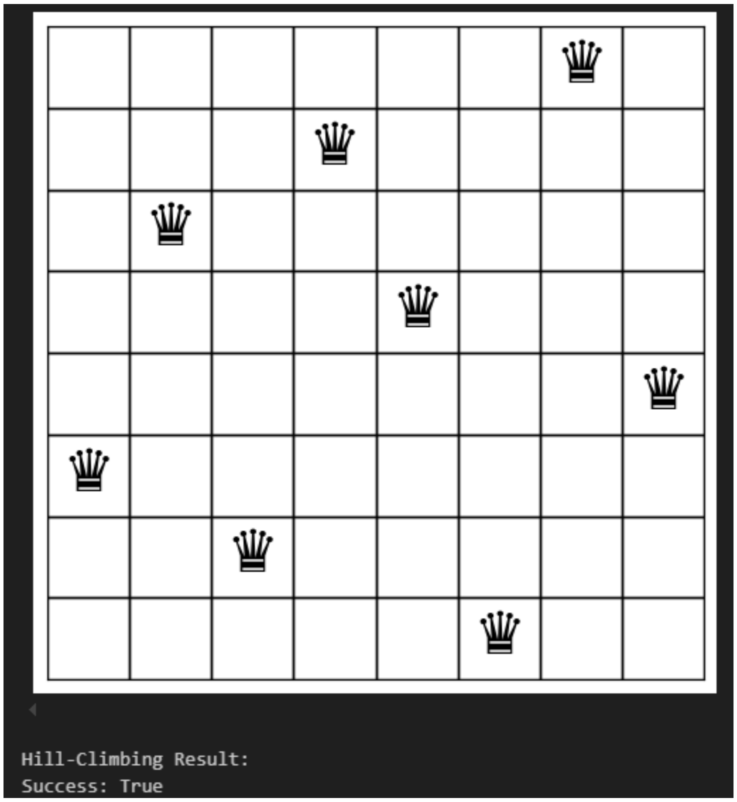
1. **Simulated Annealing**
   Helps Hill-Climbing escape local minima by allowing "worse" moves early on.
2. **Min-Conflicts Heuristic**
   Ideal for large N; often solves huge boards (N > 1000) in milliseconds.
3. **Forward Checking + Arc-Consistency (AC-3)**
   Strengthens CSP pruning and reduces backtracking.
4. **Parallel Random Restarts**
   Running multiple Hill-Climbing attempts in parallel significantly increases success rate.
5. **Hybrid Solver (HC + CSP)**
   Use Hill-Climbing to reduce conflicts, then switch to CSP for a guaranteed finish.

6. Test Case Screenshots

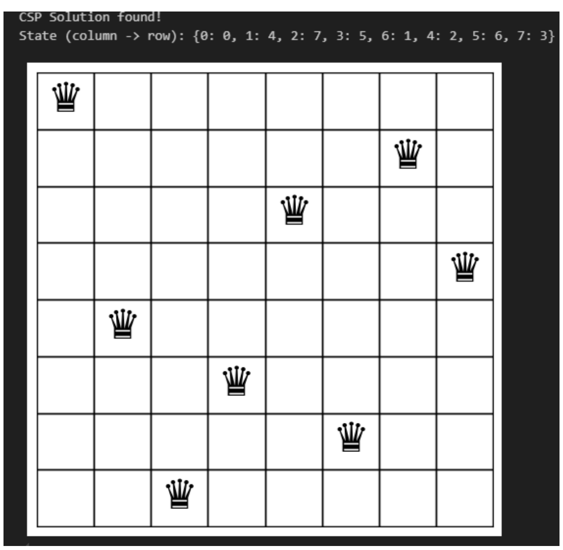
Random board before solving



Hill-Climbing solution board



CSP solution board



8. Generative AI Use Declaration

I used Generative AI tools (ChatGPT) **only** for:

- Debugging assistance
- Code structuring
- Writing explanations
- Report formatting
- Little bit for psuedocode also

Instructions

- Please execute the code blocks one by one only for first time
- Git hub repo link https://github.com/moiz-haider11/AI-CEP-8Queens-problem