COMPUTER SCIENCE AND DATA ANALYTICS

Course: CSCI 6511 Artificial Intelligence

Project 2

Project title: **CSP: N-Queens**

Students: Laman Khudadatzada, Namig Planov

Instructor: **Amrinder Arora**

Baku 2026

N-Queens CSP Project Report

Namig Planov and Laman Khudadatzada

Project Code

The complete source code for this project:
https://github.com/NamiqPlanov/P2-Team3-AI

Introduction

The **N-Queens problem** is a classic puzzle where the goal is to place n queens on an n x n chessboard such that no two queens attack each other.

This project implements a **Constraint Satisfaction Problem (CSP)** solution for N-Queens using both **Backtracking with heuristics** and **Min-Conflicts iterative repair**.

The program can read an initial board from a file or generate a random one. It automatically chooses the suitable algorithm depending on board size (n <= 50 => CSP Backtracking, n > 50 => Min-Conflicts).

Problem Formulation

The N-Queens problem is modeled as a CSP:

- **Variables:** Each row on the chessboard (0 to n-1).
- **Domains:** Columns where a queen can be placed (0 to n-1).
- **Constraints:** No two queens can share the same column or diagonal.

This CSP model allows the use of backtracking with heuristics or iterative repair to find solutions efficiently.

Algorithms

a. CSP Backtracking

For smaller boards (n <= 50), **CSP Backtracking** is used.
Key components:

- **Minimum Remaining Values (MRV):** Selects the row with the fewest remaining valid columns.
- **Least Constraining Value (LCV):** Orders columns that minimize conflicts with other queens.
- **AC-3 (Arc Consistency):** Propagates constraints to reduce domains before trying values.

Steps:

1. Initialize domains for each row.
2. Reduce domains using AC-3.
3. Select unassigned row using MRV.
4. Order possible columns with LCV.
5. Assign queen and propagate constraints recursively until solution is found.

n = 11:

```
Number of lines (10-1000): 16

Solving N-Queens for n = 16

Algorithm          | Heuristics                     | Conflicts  | Time(ms)    | Mem(KB)     | Solved | Steps
------------------------------------------------------------------------------------------------------------
CSP Backtracking   | MRV + LCV + AC3                | 0          | 174.34      | 799.02      | Yes | 18

Solution (column index per row):
[15, 13, 11, 3, 5, 12, 1, 9, 0, 2, 14, 8, 10, 7, 4, 6]

Board visualization:
. . . . . . . . . . . . . . . Q
. . . . . . . . . . . . . Q . .
. . . . . . . . . . . Q . . . .
. . . Q . . . . . . . . . . . .
. . . . . Q . . . . . . . . . .
. . . . . . . . . . . . Q . . .
. Q . . . . . . . . . . . . . .
. . . . . . . . . Q . . . . . .
Q . . . . . . . . . . . . . . .
. . Q . . . . . . . . . . . . .
. . . . . . . . . . . . . . Q .
. . . . . . . . Q . . . . . . .
. . . . . . . . . . Q . . . . .
. . . . . . . Q . . . . . . . .
. . . . Q . . . . . . . . . . .
. . . . . . Q . . . . . . . . .
```

---

b. Min-Conflicts (Iterative Repair)

For larger boards (n > 50), **Min-Conflicts** is used:

- Start with a random assignment of queens.
- Identify rows where queens are in conflict.

- Move conflicted queens to positions that minimize conflicts.
- Repeat up to a maximum number of steps (by default it is 200,000).

This algorithm scales well because it avoids full backtracking and focuses only on conflicts.

n = 333:

```
Number of lines (10-1000): 333

Solving N-Queens for n = 333

Algorithm          | Heuristics                      | Conflicts  | Time(ms)    | Mem(KB)     | Solved | Steps
---------------------------------------------------------------------------------------------------------------
Min-Conflicts      | Iterative Repair + Random TieBreak | 0          | 217.27      | 27.81       | Yes | 230

Solution (column index per row):
[99, 76, 195, 235, 281, 197, 143, 43, 234, 332, 226, 156, 293, 125, 142, 112, 274, 278, 275, 78, 191, 81, 315, 128, 140, 18,
 286, 185, 13, 220, 187, 229, 310, 55, 301, 105, 26, 158, 237, 212, 252, 122, 267, 228, 213, 146, 15, 254, 150, 41, 294, 103
, 263, 104, 2, 259, 232, 222, 86, 218, 246, 215, 147, 83, 170, 121, 320, 48, 163, 231, 202, 116, 89, 153, 6, 328, 186, 5, 16
5, 204, 127, 49, 131, 137, 50, 148, 32, 178, 92, 256, 180, 9, 74, 3, 151, 164, 66, 284, 262, 133, 37, 52, 248, 313, 182, 20,
 73, 203, 244, 14, 230, 107, 31, 63, 12, 324, 209, 33, 65, 95, 11, 47, 108, 304, 139, 268, 270, 206, 130, 201, 249, 173, 287
, 241, 114, 132, 253, 311, 271, 61, 64, 319, 174, 16, 233, 327, 118, 261, 106, 302, 102, 314, 129, 211, 177, 290, 326, 240,
0, 123, 273, 285, 176, 266, 29, 59, 307, 17, 1, 330, 300, 57, 10, 62, 214, 184, 192, 79, 325, 205, 272, 117, 23, 331, 145, 3
16, 297, 309, 250, 21, 149, 70, 238, 243, 223, 42, 159, 264, 155, 56, 53, 305, 190, 22, 210, 82, 321, 162, 60, 247, 292, 152
, 27, 109, 7, 216, 289, 227, 189, 8, 24, 90, 19, 51, 288, 225, 93, 282, 115, 242, 296, 317, 91, 69, 322, 329, 255, 308, 58,
160, 279, 318, 217, 251, 45, 183, 80, 207, 46, 113, 199, 269, 75, 119, 100, 77, 124, 168, 299, 257, 141, 239, 35, 44, 157, 2
60, 200, 94, 303, 208, 291, 258, 283, 97, 136, 101, 306, 71, 30, 161, 154, 188, 194, 196, 40, 312, 85, 34, 144, 179, 68, 166
, 36, 193, 198, 54, 98, 181, 323, 67, 111, 280, 219, 236, 120, 175, 169, 221, 38, 298, 171, 295, 172, 134, 4, 277, 126, 87,
39, 72, 276, 84, 25, 265, 88, 245, 138, 96, 28, 224, 167, 110, 135]

Board not printed (n too large).
```

Implementation Notes

- **Initial Board Input:** p2_n-queen.txt (optional). If missing, the program asks for n and generates a random board.
- **Conflict Counting:** Function conflicts(board, row, col) checks column and diagonal attacks.
- **Memory & Performance Tracking:** Uses tracemalloc and time.perf_counter() to measure memory and execution time.
- **Board Visualization:** Only displayed for n <= 50 to avoid huge output.

```
Board visualization:
. . . . . . . . . . . . . . . Q
. . . . . . . . . . . . . Q . .
. . . . . . . . . . . Q . . . .
. . . Q . . . . . . . . . . . .
. . . . . Q . . . . . . . . . .
. . . . . . . . . . . . Q . . .
. Q . . . . . . . . . . . . . .
. . . . . . . . . Q . . . . . .
Q . . . . . . . . . . . . . . .
. . Q . . . . . . . . . . . . .
. . . . . . . . . . . . . . Q .
. . . . . . . . Q . . . . . . .
. . . . . . . . . . Q . . . . .
. . . . . . . Q . . . . . . . .
. . . . Q . . . . . . . . . . .
. . . . . . Q . . . . . . . . .
```

Testing and Validation

- test.py validates algorithms for different board sizes.
- Measures:
  - Total conflicts
  - Execution time
  - Peak memory usage
  - Steps taken
- Results for small boards include full board output, but large boards skip visualization for readability.

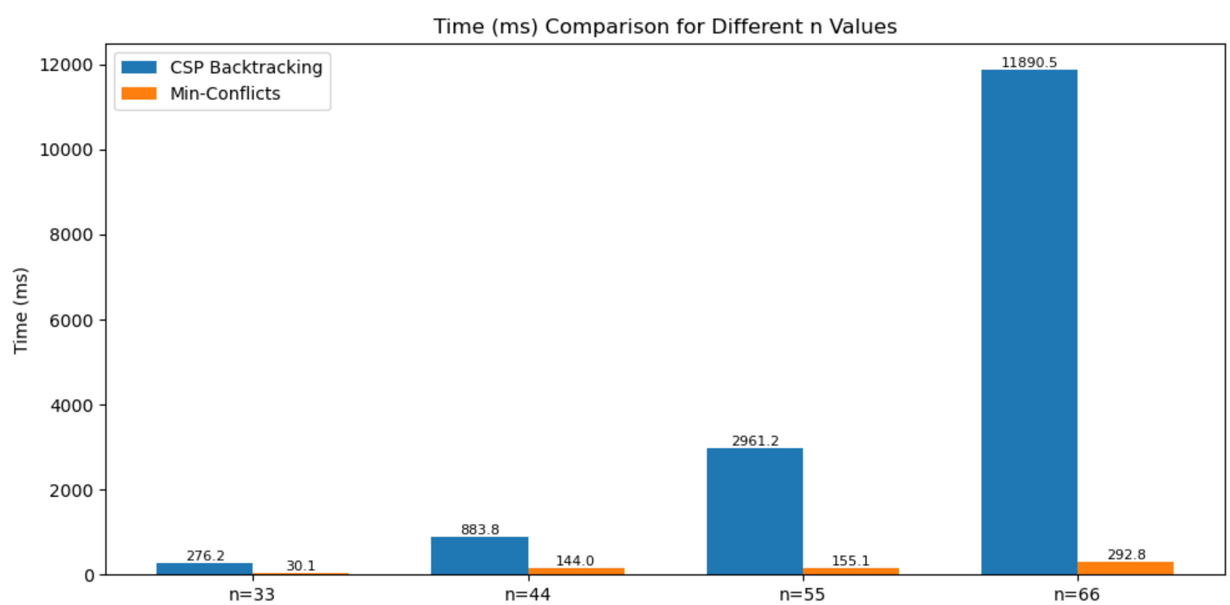
Example result table for n = 23:

Algorithm	Conflicts	Time(ms)	Mem(KB)	Solved	Steps
CSP Backtracking	0	912.59	2995.83	Yes	27
Min-Conflicts	0	2.83	1.97	Yes	168

Example result table for n = 200:

Algorithm	Conflicts	Time(ms)	Mem(KB)	Solved	Steps
CSP Backtracking	Skipped	0.00	0.00	Skipped	–
Min-Conflicts	0	86.55	12.57	Yes	232

Visualization of 4 different test cases:



Observations and Notes

- **CSP Backtracking** is slower on large boards due to exponential growth in possibilities.
- **Min-Conflicts** performs well even for n = 1000 but may not guarantee a solution if max steps are reached.
- Memory usage scales linearly with n for both algorithms.
- AC-3 significantly reduces the search space for small boards.

Conclusion

This project successfully implements two CSP approaches for the N-Queens problem:

- Backtracking with heuristics and AC-3 for small boards.
- Min-Conflicts iterative repair for large boards.
