N-Queen Problem

Problem description:

In the N-Queen problem, we are given an NxN chessboard and
we have to place N number of queens on the board in such a
way that no two queens attack each other. A queen will attack
another queen if it is placed in horizontal, vertical or diagonal points in its
way.

BFS Steps to Solve N-Queens Problem:

1-Initialize:

- Create a queue.
- Add an empty state [ ] to the queue.

2-Dequeue a State:

- Remove the first state from the queue.
- Determine the current row by the number of placed queens.

3-Check for Solution:

- If the number of queens equals **N**: The state is a valid solution,Store it,If it is the first solution, save it.

4-Generate Next States:

- If the solution is not complete:Try placing a queen in each column of the current row,Check if the position is safe

5-Enqueue Safe States:

- For each safe position:Create a new state,Add it to the queue.

6-Repeat:

- Continue until the queue becomes empty.

7-Finish : Return the first solution, total number of solutions, and execution time.

DFS Backtracking Steps to Solve N-Queens Problem

1-Initialize:

- Start the timer.
- Reset the solutions counter.
- Create a board initialized with `-1` for all rows.

2-Start Backtracking**:**

- Begin from the first row (row = 0).

3-Check for Complete Solution:

- If the current row equals N:A valid solution is found,Increase the solutions count,If it is the first solution, store it.Return to explore other possibilities.

4-Try All Columns:

- For the current row:Try placing the queen in each column.

5-Safety Check:

- For each column:Check if the position is safe (no conflicts).

6-Place Queen and Recurse:

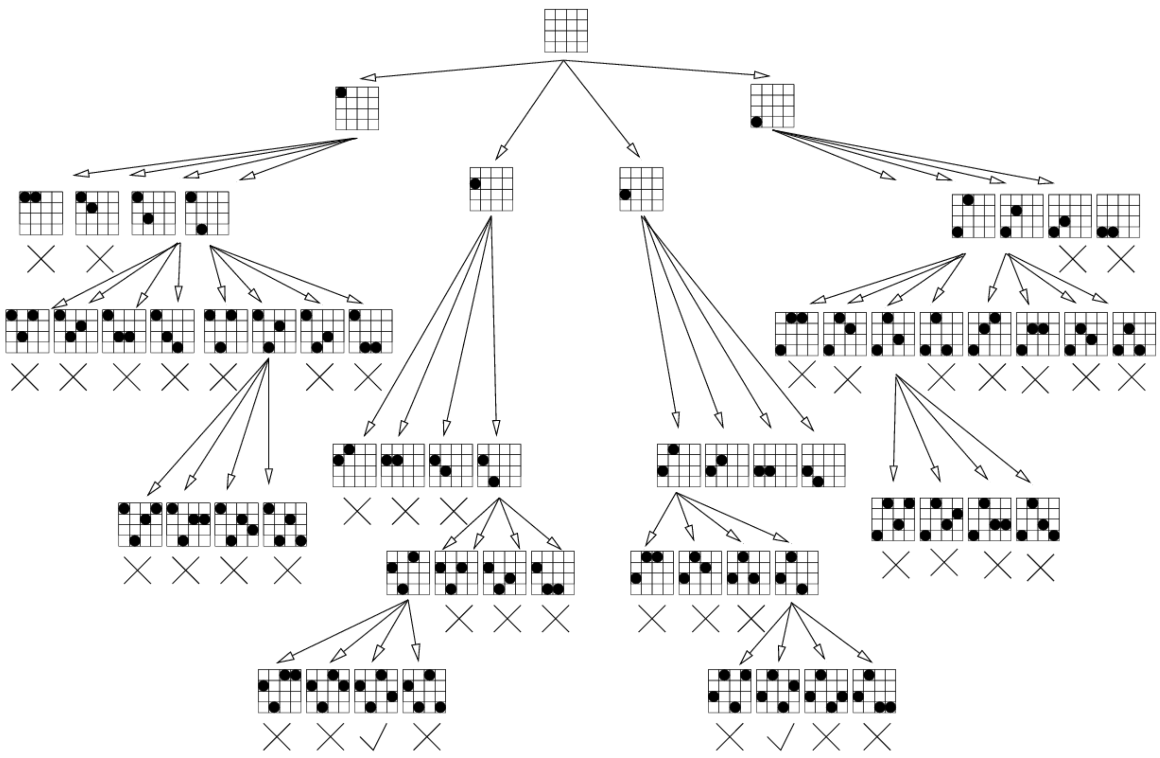
- If safe: Place the queen on the board,Recursively call backtracking for the next row.

7-Backtrack :

- After returning from recursion:Remove the queen (reset position),Try the next column.

8-Finish:

- After exploring all possibilities:Return the first solution, total number of solutions, and execution time.



Genetic Algorithm Steps to Solve N-Queens Problem :

1-Initialize Parameters:

- Define board size N.
- Set population size, number of generations, crossover probability, and mutation probability.

2-Generate Initial Population:

- Create a population of random chromosomes.
- Each chromosome represents a possible queen placement.

3-Evaluate Fitness:

- Calculate the fitness of each chromosome.
- Fitness equals the number of diagonal conflicts (lower is better).

4-Select Parents:

- Randomly select a small group from the population.
- Choose the best individuals as parents (tournament selection).

5-Apply Crossover:

- With probability Pc, perform crossover to produce a child.
- Otherwise, copy one parent directly.

6-Apply Mutation:

- With probability Pm, swap two genes in the chromosome.

7-Create New Population :

- Keep the best solution (elitism).
- Fill the rest of the population with new children.

8-Update Best Solution

- Compare the current best solution with the global best.
- Update if a better one is found.

9-Check Termination:

- If fitness equals 0, a valid solution is found.Or stop after reaching the maximum number of generations.

10-Finish:

- Return the best solution, its fitness, number of generations, and execution time.

Hill Climbing Steps to Solve N-Queens Problem:

1-Initialize:

- Start the timer.
- Generate a random initial state (or use a given start state).

2-Evaluate Current State:

- Calculate the heuristic value (number of conflicts) of the current state.

3-Generate Neighbors:

- Generate all neighboring states by moving one queen to another column in the same row.

4-Select Best Neighbor:

- Evaluate the heuristic of each neighbor.
- Choose the neighbor with the lowest heuristic value.

5-Check for Improvement:

- If the best neighbor is not better than the current state:Stop the algorithm (local optimum reached).

6-Move to Better State:

- CoIf a better neighbor exists:Update the current state,Update the current state.

7-Repeat :

- Continue searching until no better neighbor is found.

8-Finish:

- Return the final state, number of conflicts, and execution time.

Time Complexity &Space Complexity of N-Queens Algorithms

Algorithm	Time Complexity	Space Complexity
DFS Backtracking	O(N!)	O(N)
Hill Climbing	$O(I \times N^2)$	O(N)
Genetic Algorithm	O(G × N × P)	O(P × N)
BFS	O(N!)	O(N!)

Experimental Methodology for N-Queens Algorithms

1-**Objective**:

- Evaluate the performance of four algorithms for solving the N-Queens problem: BFS, DFS Backtracking, Hill Climbing, Genetic Algorithm.
- Metrics used:Execution Time,Solutions Count,Conflicts / Fitness,Conflicts / Fitness,Generations / Iterations (for GA and Hill Climbing).

2- Problem Setup:

- Tested on different board sizes: N = 4, 8, 12
- Each algorithm starts from a random initial state to provide a realistic assessment.
- Experiments repeated several times for each N to reduce randomness effects (especially for Hill Climbing and Genetic Algorithm).

3- Algorithm Execution & Data Collection:

- For each algorithm:

  1-BFS / DFS: generate all possible solutions and validate them.

  2-Hill Climbing: search for a better solution by moving queens to reduce conflicts.

  3-Genetic Algorithm: use population, crossover, and mutation to reach the best solution.

- Recorded for each run:

  1-Solution: final queen positions

  2-Conflicts / Fitness: number of conflicts or fitness value.

  3-Execution Time in seconds.

  4-Solutions Count: total solutions found (BFS and DFS).

  5-Generations: number of generations (GA).

---

1. N = 4

Algorithm	Total Solutions	Conflicts/Fitness	Generations	Execution Time (s)
Hill Climbing	-	0	-	0.0–0.001
BFS&Dfs	2	-	-	0.0
Genetic Algorithm	-	0	1	0.001

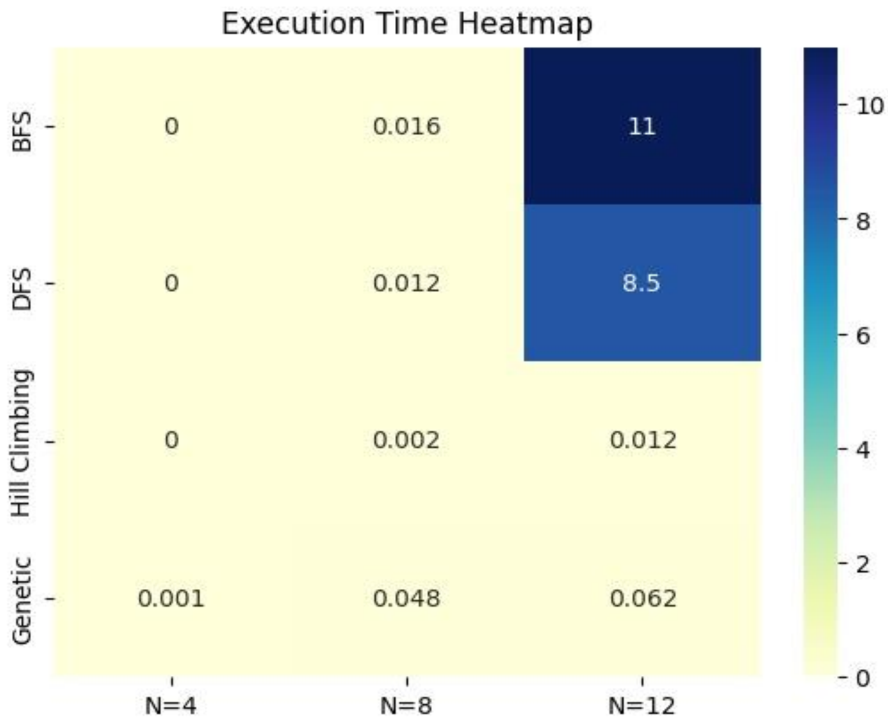
2. N = 8

Algorithm	Total Solutions	Conflicts/Fitness	Generations	Execution Time (s)
Hill Climbing	-	0–2	-	0.001–0.003
DFS	92			0.009–0.012
BFS	92	-	-	0.016–0.019
Genetic Algorithm	-	0–1	6–50	0.006–0.048

3. N = 12

Algorithm	Total Solutions	Conflicts/Fitness	Generations	Execution Time (s)
Hill Climbing	-	1–2	-	0.008–0.012
DFS	14200			8.5–8.6
BFS	14200	-	-	10.6–11.0

Genetic Algorithm	-	0–2	1–2	0.047–0.062



---

5 Analysis & Conclusion

- **DFS / BFS:** Excellent for finding all correct solutions for small N, but impractical for large N due to time and memory.
- **Hill Climbing**: Fast and effective for medium-size boards, but may get stuck in local optimal.
- **Genetic Algorithm:** Balances speed and solution quality, suitable for large boards; performance improves with population size and number of generations.

Recommendation:

- **N ≤ 8**: DFS or BFS are sufficient.

- **N > 8:** Use Hill Climbing or Genetic Algorithm for faster and effective solutions.