Applying AI search algorithms to solve the N-Queens problem

INTRODUCTION AI

TEAM MEMBERS

BÙI ĐĂNG DƯƠNG
TÔ HIẾN HẢI ĐĂNG
NGUYỄN BẢO TÀI
PHAN ĐĂNG VŨ

Instructor: Nguyễn Trọng Nghĩa

Business AI Lab

Content

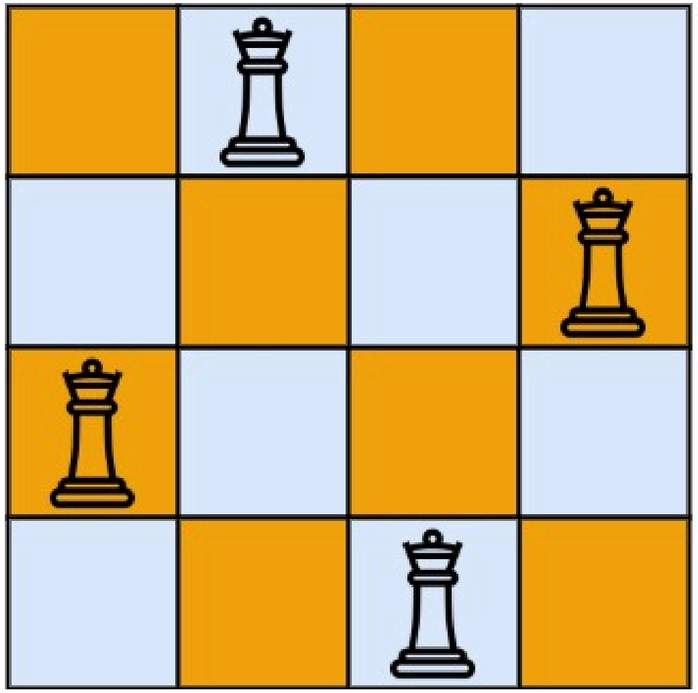
- Real-world problem
- Problem modeling
- Search algorithms
- Algorithm evaluation

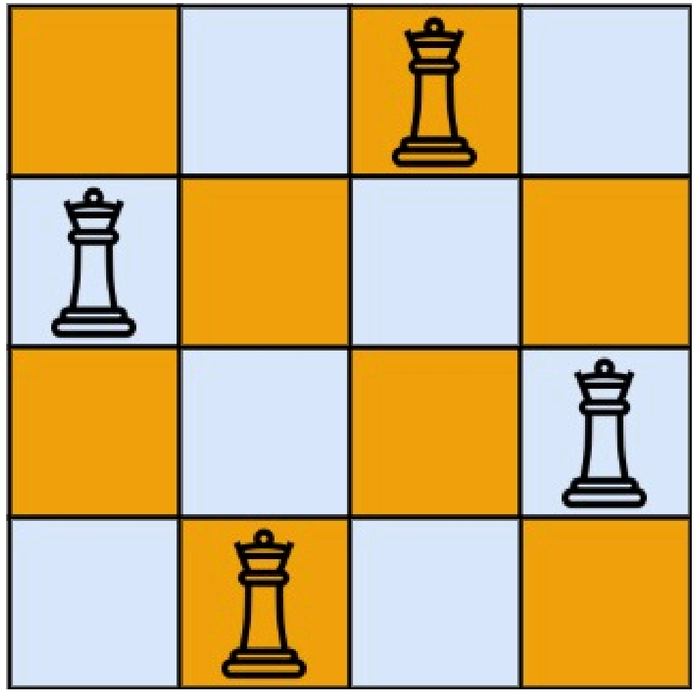
Real-world problem

- In high-security fields such as border management, forest resource protection, or critical infrastructure surveillance, the placement of observation stations — such as watchtowers — plays a crucial role. Our goal is to determine how, with a limited number of towers, we can maximize visibility, cover the entire area, and, most importantly, eliminate any potential blind spots.
- In practice, even a small mistake in the arrangement can lead to serious consequences: not only wasting resources (manpower, construction costs) but also significantly reducing the overall security efficiency of the system.
- This problem can be modeled similarly to the N-Queens problem — a classic example of positional optimization in computational problem-solving

Problem Formulation

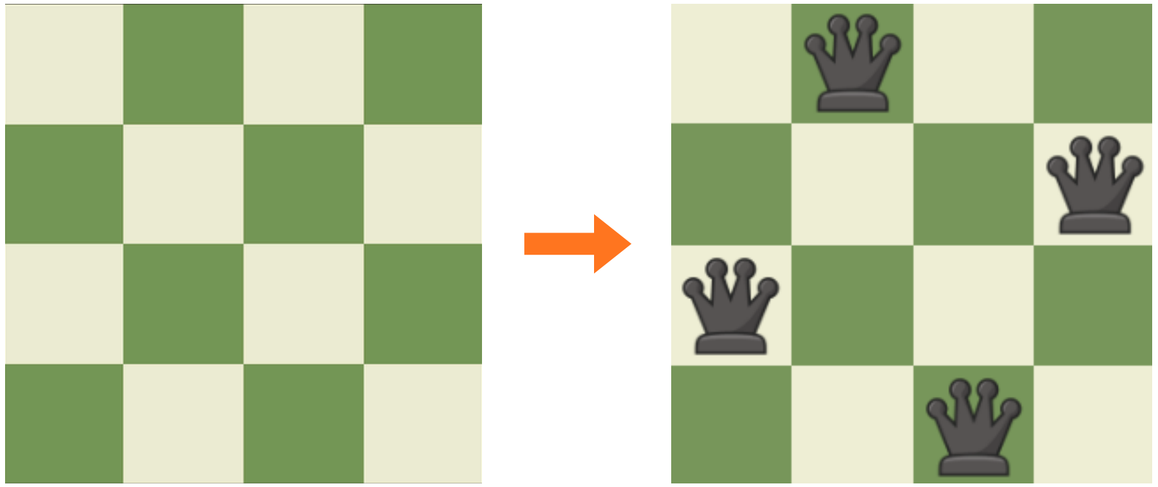
- N-Queens is the problem of placing N queens on an NxN chessboard such that no two queens attack each other.
- Example with $N = 4$





Problem Formulation

- **State space:** Any arrangement of k queens (0≤k≤N) on an N×N grid such that no two queens attack each other.
- **Initial state:** An empty N×N board with no queens.
- **Action:** Place a queen on the next empty file in a safe position (not blocked by other queens).
- **Transition model:** From a state with k queens, taking a valid action will create a new state with k+1 queens.
- **Goal state:** Successfully place N queens on the board such that no queens are in the same row, column, or diagonal.
- **Cost:** Each action (placing 1 queen) costs 1.
- **The total cost to reach the goal is N.**



Initial State

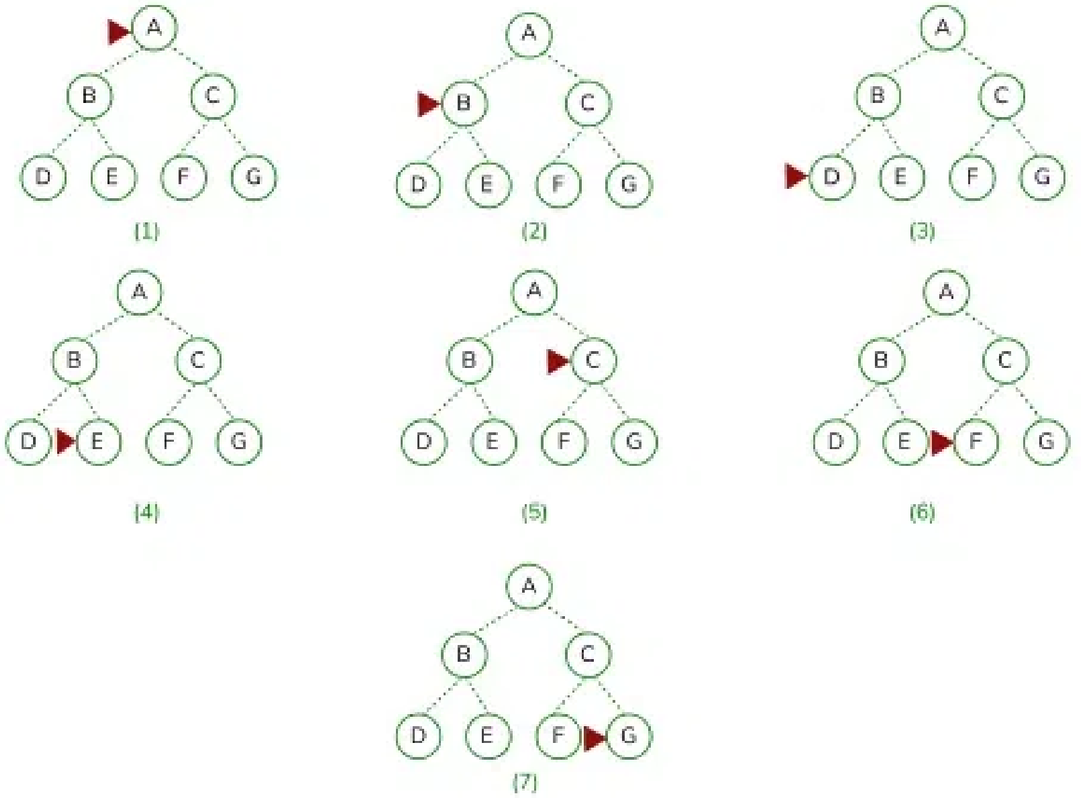
Goal State

Algorithms

Depth-First Search

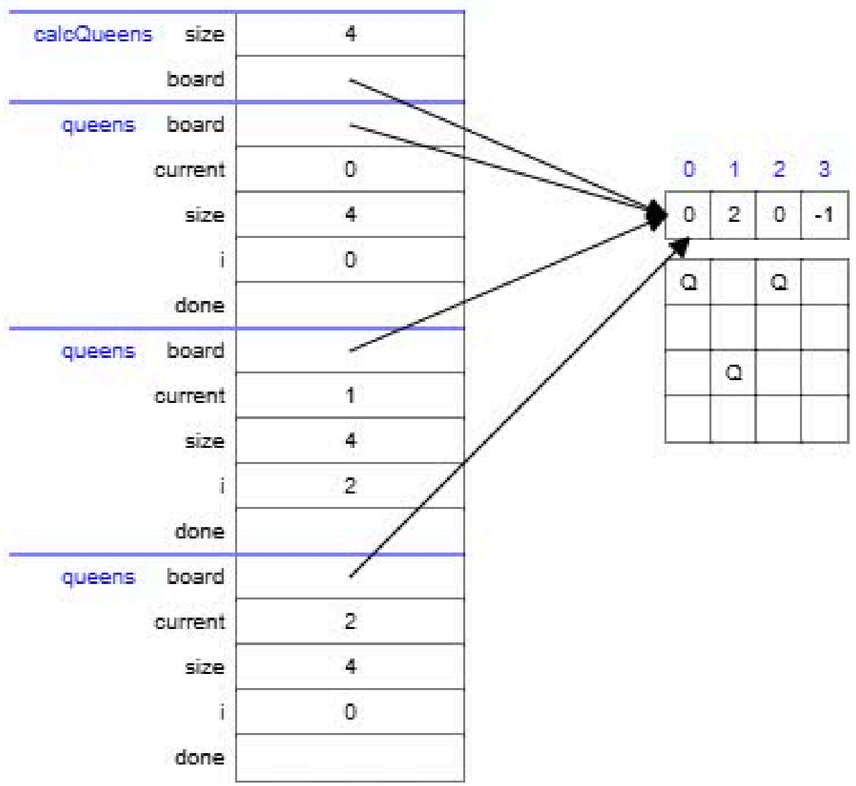
- This method focuses on depth, meaning it will go through a branch and if it does not find a solution, it will go back to the previous solution and branch out to other branches to continue finding solutions, and continue until all nodes are visited.

- In programming, this algorithm is called backtracking.



Algorithms

Depth-First Search



Comment

```python
def dfs_nqueens(n):
    solutions = []

    def is_safe(board, row, col):
        for r in range(row):
            c = board[r]
            if c == col or abs(c - col) == abs(r - row):
                return False
        return True

    def dfs(board, row):
        if row == n:
            solutions.append(board[:])
            return
        for col in range(n):
            if is_safe(board, row, col):
                board[row] = col
                dfs(board, row + 1)
                board[row] = -1

    dfs([-1] * n, 0)
    return solutions
```

DFS / Backtracking

Advantages: Guaranteed to find a solution if it exists (completeness). Low memory requirements.

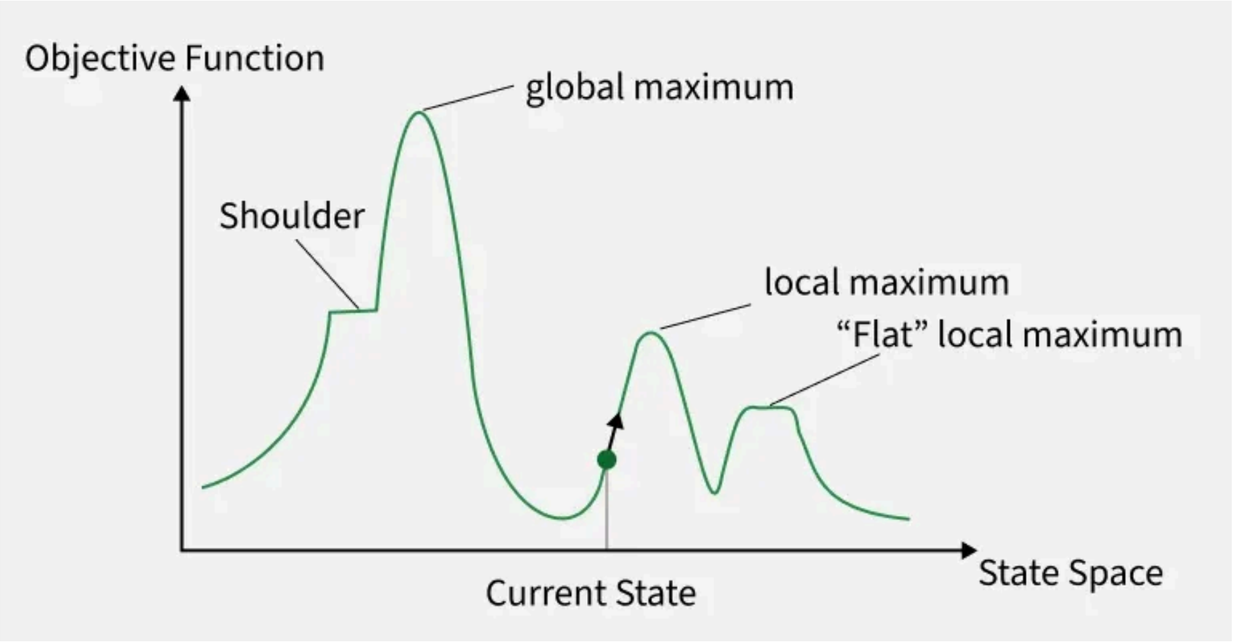
Disadvantages: Very slow when N is large. Due to the need to “blindly” traverse a huge search tree, the running time can increase exponentially.

Practical significance: A reliable, foundational method for small-scale problems (N < 20). It is suitable when an exact solution is required and time is not a top priority.

Algorithms

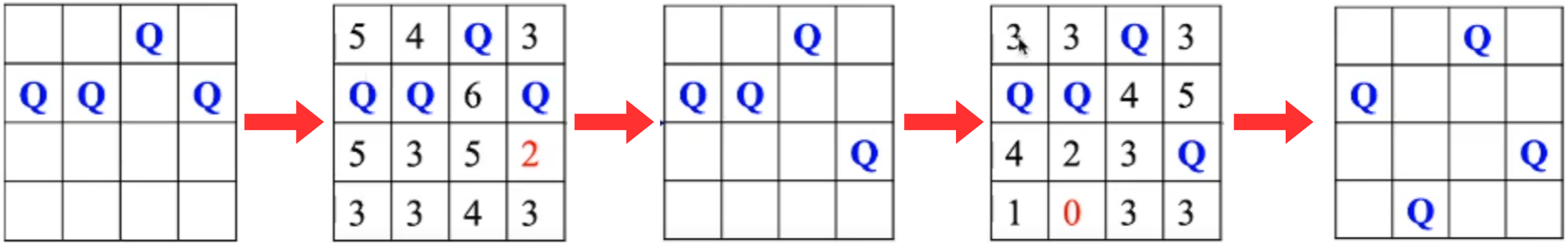
Hill-Climbing

- This is a local search technique in artificial intelligence to find solutions.
- Works by starting from an initial solution and continually moving to better neighboring solutions until there are no better solutions left, like a mountain climber working his way to the top.

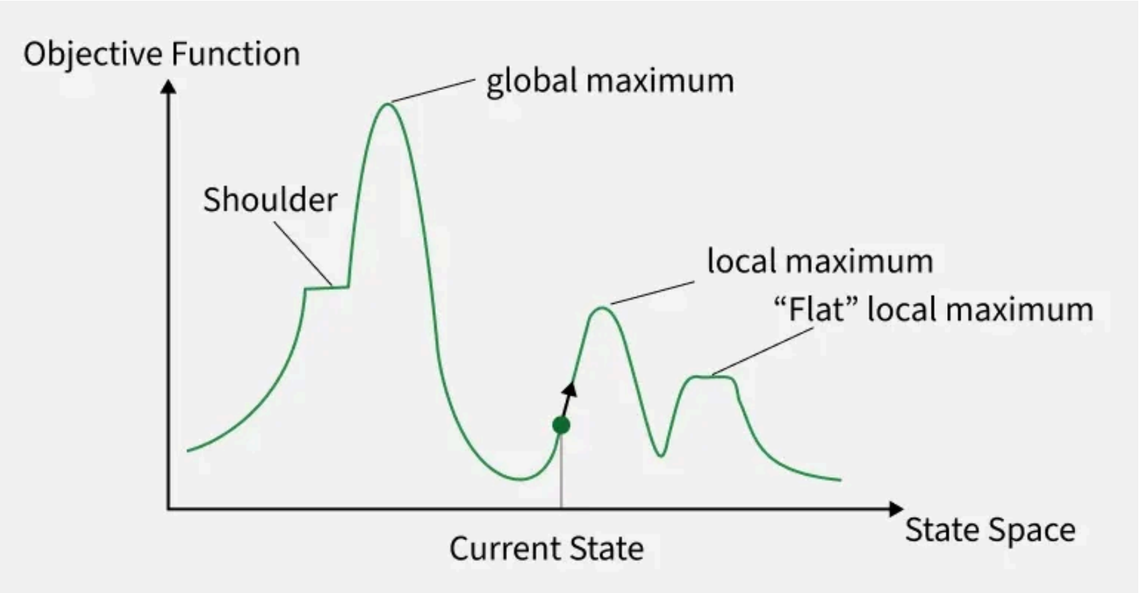


Algorithms

Hill-Climbing



Comment



Hill Climbing

Advantages: Extremely fast with minimal memory requirements (O(N)). Usually produces near-instant results.

Disadvantages: No guarantee of finding a solution. Low success rate due to frequent getting stuck at local extrema

Practical significance: Ideal for systems that need a “good enough” answer in real time. In the watchtower problem, it can be used to quickly generate an initial, albeit imperfect, solution, or run multiple iterations with different starting points to increase the chance of success.

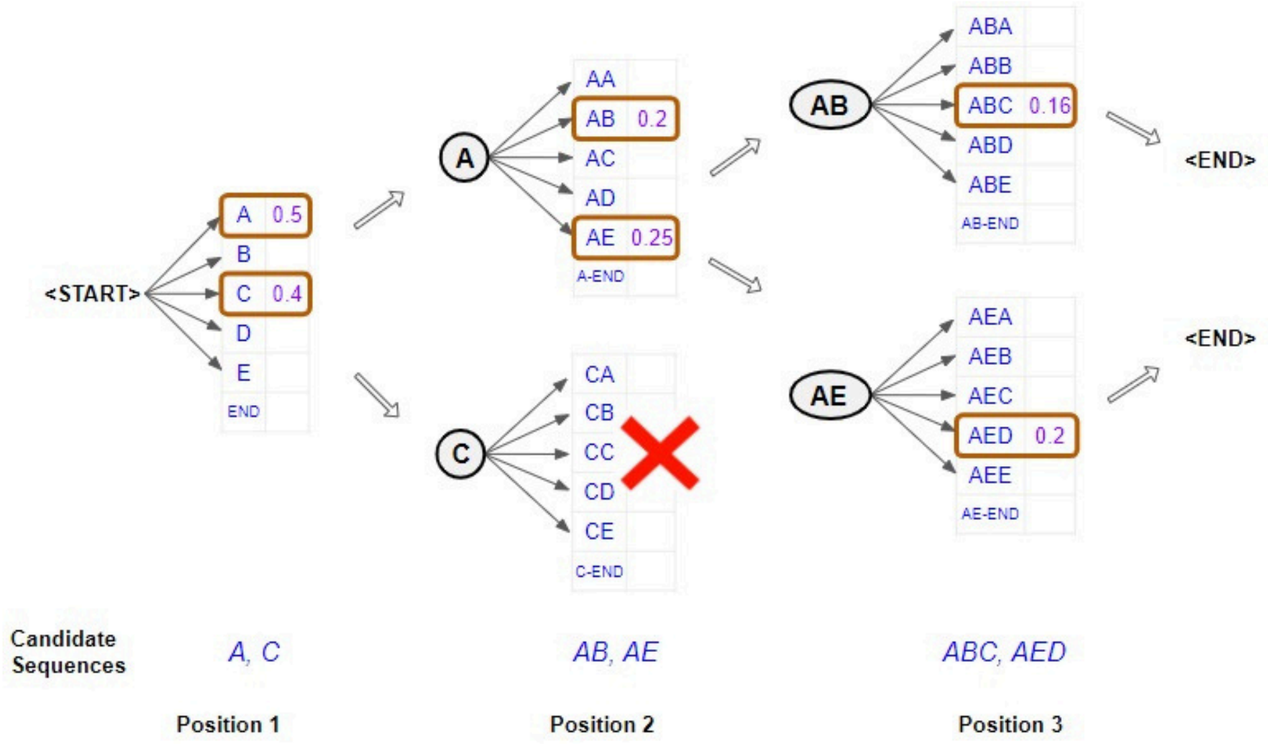
Comment

```python
def hill_climbing(n, max_steps=1000):
    board = random_board(n)
    for _ in range(max_steps):
        current_conf = conflicts(board)
        if current_conf == 0:
            return board
        # tạo các hàng xóm
        neighbors = []
        for row in range(n):
            for col in range(n):
                if col != board[row]:
                    new_board = board[:]
                    new_board[row] = col
                    neighbors.append((new_board, conflicts(new_board)))
        # chọn hàng xóm tốt nhất
        board, new_conf = min(neighbors, key=lambda x: x[1])
        if new_conf >= current_conf:  # không cải thiện
            return board
    return board
```

Algorithms

Beam Search

- Concept: Beam search is an optimized version of breadth-first search. The idea is to use an evaluation function and then select the top k best elements to explore.



Comment

```python
def beam_search(n, k=2):
    states = [[]]  # trạng thái rỗng
    for row in range(n):
        new_states = []
        for state in states:
            for col in range(0, n):
                new_state = state + [col]
                new_states.append(new_state)

        # Tính điểm cho tất cả trạng thái
        new_states.sort(key=lambda s: heuristic(s))
        # Giữ lại k trạng thái tốt nhất
        states = new_states[:k]

    # Lọc ra nghiệm hợp lệ (score=0)
    solutions = [s for s in states if heuristic(s) == 0]
    return solutions
```

Beam Search

Advantages: Provides the best balance between speed and reliability. It is faster than DFS when N is large and has a much higher success rate compared to Hill Climbing.

Disadvantages: Does not guarantee finding a solution. There is a risk of missing the correct path if it temporarily falls outside the top k best options at any step. Its performance heavily depends on choosing the appropriate parameter k.

Practical significance: This is the most practical choice for many optimization problems. It allows users to adjust the balance between time and solution quality. For the watchtower problem, a manager can use Beam Search to quickly generate a high-quality layout with high reliability without waiting for an exhaustive and costly search process.

Algorithms Analyze

Test Case	Thuật toán	Thời gian chạy (ms)
Easy (N=4)	DFS / Backtracking	0.02
	Hill Climbing	0.09
	Beam Search (k=3)	0.04
Medium (N=6)	DFS / Backtracking	0.20
	Hill Climbing	0.30
	Beam Search (k=9)	0.3
Hard (N=8)	DFS / Backtracking	3.80
	Hill Climbing	0.56
	Beam Search (k=40)	3

Conclusion

In conclusion, the three algorithms — DFS, Hill Climbing, and Beam Search — represent distinct approaches to solving optimization problems, such as the watchtower placement scenario.

- DFS stands for completeness and reliability — although slow, it guarantees finding a valid solution if one exists.
- Hill Climbing embodies practicality and speed — ideal when a quick, “good enough” solution is needed in real time.
- Beam Search strikes a balance between the two extremes — significantly faster than DFS while maintaining a higher success rate than Hill Climbing.

Depending on the specific goal — accuracy, speed, or overall efficiency — one can select the most suitable algorithm. In most real-world applications, Beam Search often serves as the best compromise, offering an excellent balance between solution quality and computational cost.

Thank you for listening