Solving the 5-Queens Problem Using A* Search Algorithm

Problem Description

The 5-Queens problem is a classical artificial intelligence problem. The goal is to place 5 queens on a 5×5 chessboard such that no two queens attack each other.

This means:
- No two queens are in the same row
- No two queens are in the same column
- No two queens are on the same diagonal

State Representation

Each state is represented as a list of integers:
- The index represents the column
- The value represents the row of the queen in that column

Example:
[0, 2, 4]
This means:
Queen at column 0, row 0
Queen at column 1, row 2
Queen at column 2, row 4

Only one queen is placed per column, which automatically avoids column conflicts.

Initial State

The initial state is an empty list:
[]

No queens are placed yet.

Goal State

A state is considered a goal state when:
- The length of the state is 5
- There are no conflicts between any two queens

Successor Function

From a given state, successor states are generated by:
- Placing a queen in the next column
- Trying all possible rows from 0 to 4
- Adding the queen only if the position is safe (no row or diagonal conflict)

Cost Function

A* Search uses a path cost g(n):
- Each step has a cost of 1
- The total cost equals the number of queens placed so far

Heuristic Function

The heuristic function h(n) estimates how close the current state is to a valid solution.

The heuristic used is:
- Number of conflicts between queens

This heuristic is admissible because it never overestimates the remaining cost.

A* Search expands nodes based on the evaluation function:
f(n) = g(n) + h(n)

Where:
- g(n) is the path cost
- h(n) is the heuristic value

This allows A* to explore promising states first and reduce unnecessary expansions.

Python Code

```
import heapq


# Board size
N = 5

# Function to check if a queen can be safely placed in the given
row
def is_safe(state, row):
    col = len(state)   # Next column to place
    for c, r in enumerate(state):
        # Check row and diagonal conflicts
        if r == row or abs(r - row) == abs(c - col):
            return False
    return True

# Heuristic function: number of conflicts between queens
def heuristic(state):
    conflicts = 0
    for i in range(len(state)):
        for j in range(i + 1, len(state)):
            if state[i] == state[j] or abs(state[i] - state[j])
== abs(i - j):
                conflicts += 1
    return conflicts

# A* Search function
def a_star_search():
    # Priority Queue: (f, g, state)
    pq = []
    heapq.heappush(pq, (0, 0, []))  # Initial state

    while pq:
        f, g, state = heapq.heappop(pq)

        # Goal test: 5 queens placed safely
        if len(state) == N:
            return state

        # Generate successors
        for row in range(N):
            if is_safe(state, row):
                new_state = state + [row]
                new_g = g + 1
                new_h = heuristic(new_state)
                new_f = new_g + new_h
                heapq.heappush(pq, (new_f, new_g, new_state))

    return None

# Run A*
solution = a_star_search()
print("Solution:", solution)
```

Example Output

Solution: [0, 2, 4, 1, 3]

Each number represents the row position of the queen in the corresponding column.

Conclusion

A* Search successfully solves the 5-Queens problem by combining path cost and heuristic information.
Compared to BFS and UCS, A* reduces the number of expanded states by guiding the search toward conflict-free configurations.