Heuristic Search for Marble Solitaire using UCS, Best-First and A* Algorithms

Your Name
Department of Computer Science, Your University

A. Objective

The objective of this work is to understand the role of heuristic functions in reducing the size of the search space for complex problems. We explore the application of non-classical search algorithms to the game of Marble Solitaire. Specifically, we implement:

- Priority Queue based Uniform Cost Search (UCS),
- Greedy Best-First Search with two heuristic functions,
- A* Search algorithm.

B. Problem Statement

Marble Solitaire (Peg Solitaire) is a single-player game played on a cross-shaped board with 33 valid positions. Initially, 32 positions are filled with marbles and the center is left empty. A move consists of jumping one marble over an adjacent marble into an empty hole, removing the jumped marble. The goal state is to have exactly one marble remaining, located at the central position.

C. Methodology

We use three algorithms for solving this problem:

1. Uniform Cost Search (uninformed baseline).

2. Best-First Search with heuristics.

3. A* Search with admissible heuristics.

D. Algorithm

Algorithm 1 Uniform Cost Search (UCS)

```
function UCS(start_state, goal_state)
    Initialize open_list ← priority queue with (start_state, cost = 0)
    Initialize visited ← ∅
    while open_list not empty do
        Pop node with lowest path cost g
        if node in visited then
            continue
        end if
        Add node to visited
        if node.state = goal_state then
            return solution path
        end if
        for each successor of node do
            if successor not in visited then
                Push successor into open_list with cost g + 1
            end if
        end for
    end while
    return failure
end function
```

Algorithm 2 Best-First Search

```
1: function BestFirst(start_state, goal_state, h)
2:     Initialize open_list ← priority queue with (start_state, h(start_state))
3:     Initialize visited ← ∅
4:     while open_list not empty do
5:         Pop node with lowest h value
6:         if node in visited then
7:             continue
8:         end if
9:         Add node to visited
10:        if node.state = goal_state then
11:            return solution path
12:        end if
13:        for each successor of node do
14:            if successor not in visited then
15:                Push successor into open_list with h(successor)
16:            end if
17:        end for
18:    end while
19:    return failure
20: end function
```

Algorithm 3 A* Search Algorithm

```
function A*(start_state, goal_state, h)
    Initialize open_list ← priority queue with (start_state, f = g + h)
    Initialize visited ← ∅
    while open_list not empty do
        Pop node with lowest f = g + h
        if node in visited then
            continue
        end if
        Add node to visited
        if node.state = goal_state then
            return solution path
        end if
        for each successor of node do
            if successor not in visited then
                Compute g' = g + 1, f' = g' + h(successor)
                Push successor into open_list with f'
            end if
        end for
    end while
    return failure
end function
```

E. Results

Experiments show that:

- UCS guarantees optimal solution but is computationally infeasible.
- Best-First is fast but non-optimal.
- A* balances efficiency and optimality, best when heuristic is admissible.

F. Conclusion

Heuristic-driven algorithms dramatically reduce search space in Marble Solitaire. A* with Manhattan distance heuristic achieves the best trade-off between performance and optimality.