**Report on Search Algorithms for the Coin-Moving Puzzle**

**1. Heuristic Evaluation Function Used for the A\* Algorithm**

The heuristic function used in the A\* search algorithm evaluates the number of circles (coins) that have not yet been moved from the center stack `stack\_E` onto any of the other stacks. The aim is to estimate how far the current state is from the goal state. Here’s a breakdown of how the heuristic works:

**Heuristic Calculation**

- It takes into account the coins that remain in the center stack (`stack\_E`).

- The function examines if the coins in `stack\_E` are consecutively ordered by decrementing values (i.e., checking if `stack\_E[i] - stack\_E[i+1] == 1`).

- It counts how many coins are in sequence, which gives a measure of how "organized" or "close to solution" the current state is.

- The more consecutive coins there are in `stack\_E`, the lower the heuristic value, suggesting that fewer moves are required to reach the goal.

This heuristic is admissible because it never overestimates the true cost of reaching the goal, making it suitable for A\* search. The cost of each move is fixed at 1, so the total cost for A\* is the sum of the heuristic value and the path cost (`gcost`), ensuring the algorithm finds the optimal solution.

**2. Comparison of the Performance of Search Methods**

In this section, we compare the performance of four search algorithms: Depth-First Search (DFS), Breadth-First Search (BFS), Best-First Search (Greedy), and A. Performance is measured by the number of nodes visited, the path length, and the efficiency of solving problems with varying numbers of coins (4, 6, 8, and 10).

(a**) Depth-First Search (DFS)**

- Path Explored: DFS explores the depth of the tree first, following one path until it reaches the goal or a dead end.

- Nodes Visited: DFS tends to visit more nodes than BFS and A\* because it does not use a heuristic and can go deep into suboptimal paths.

- Efficiency: It is less efficient for larger problem sizes since it doesn't use any cost or heuristic to guide its search, potentially visiting irrelevant nodes.

- Path Length: DFS may not always find the shortest path, especially for complex problem

(b) **Breadth-First Search (BFS)**

- Path Explored: BFS explores all nodes at the current level before moving deeper. It guarantees the shortest path but can be memory-intensive as it needs to keep track of many nodes.

- Nodes Visited: BFS explores fewer nodes than DFS for finding the optimal solution but still visits more nodes than heuristic-based algorithms like A\*.

- Efficiency: While BFS guarantees finding the shortest path, it is slower compared to A\* and Greedy Search, particularly for larger problem instances.

- Path Length: Always finds the shortest path due to its level-order exploration.

(c) **Best-First Search (Greedy Search)**

- Path Explored: Best-First Search uses the heuristic function alone to guide its search, aiming to minimize the estimated distance to the goal.

- Nodes Visited: It visits fewer nodes compared to BFS and DFS because it focuses on the most promising path based on the heuristic. However, it can be misled by local optima as it doesn't consider the cost of moves (`gcost`).

- Efficiency: Best-First Search is fast and efficient for smaller problems but may not find the optimal solution in larger problem instances since it doesn't take into account the cost of each move.

- Path Length: The path found may not always be optimal because it is driven only by the heuristic and ignores the actual move cost.

(d**) A\* Search**

- Path Explored: A\* uses both the heuristic and the cost (`gcost`) to explore the path with the lowest total estimated cost.

- Nodes Visited: A\* visits the fewest nodes of the four algorithms because it balances between path cost and the heuristic value, making it the most efficient for larger problem instances.

- Efficiency: A\* is the most efficient and guarantees finding the optimal solution. It is particularly effective for larger instances of the problem, such as those with 8 or 10 coins, due to its informed search strategy.

- Path Length: Always finds the optimal solution path, ensuring that it balances between short-term gains (based on the heuristic) and long-term costs based on gcost.

**Performance Summary Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Nodes Visited | Path Length | Efficiency (Large Problems |
| DFS | High | Varies | Low |
| BFS | Moderate | Optimal | Moderate |
| Greedy Search | Low | Varies | Moderate |
| A\* Search | Low | Optimal | High |

**Conclusion**

The performance comparison shows that A\* search, with its balance of heuristic guidance and path cost evaluation, is the most effective method for solving the coin-moving puzzle, especially as the problem size increases. While DFS and BFS have their merits, particularly in smaller problem instances, they become inefficient as the problem grows. Best-First Search is fast but prone to suboptimal paths, whereas A\* consistently finds the optimal solution with the fewest nodes visited.

This report highlights the importance of using an informed search strategy for solving complex search problems efficiently.