Lab Assignment 2: To design a graph search agent and understand the use of a hash table, queue in state space search.

Reedam Choudhary
(Roll no.- 20251602020)
M.Tech(DS)
1st year

Tishu Verma
(Roll no.- 20251602024)
M.Tech(DS)
1st year

Mansi Surti
(Roll no.- 20251602014)
M.Tech(DS)
1st year

***Abstract*—This paper examines two key artificial intelligence concepts: graph search algorithms for state-space problems and the A* search algorithm for text alignment in plagiarism detection. The first part involves designing a graph search agent and applying it to the classical 8-puzzle problem, demonstrating the effectiveness of queues and hash tables in avoiding repeated states. The second part applies heuristic search to plagiarism detection, where the A* algorithm aligns sentences between two documents to minimize edit distance. Results show that heuristic-based search provides efficient and optimal solutions in both discrete puzzle domains and natural language processing tasks.**

I. Introduction

Search algorithms form the backbone of artificial intelligence (AI) as they enable systematic exploration of state spaces to identify solutions. Graph search agents operate by expanding nodes while maintaining an explored set to prevent revisiting the same states. Data structures such as queues and hash tables are crucial for implementing strategies like Breadth-First Search (BFS) and Uniform Cost Search effectively.

The 8-puzzle is a benchmark problem used to evaluate search algorithms, requiring rearrangement of tiles on a 3×3 grid into a goal configuration. This task highlights computational challenges such as state explosion and memory constraints.

In addition to puzzle solving, search techniques are also valuable in natural language processing. This study applies the A* algorithm to plagiarism detection by framing sentence alignment as a heuristic-guided search problem to minimize edit distance between documents.

II. Problem Formulation

A. Graph Search Agent for Puzzle-8

The task involves creating a graph search agent that efficiently explores the state space of the 8-puzzle problem. Functions simulate valid moves, check goal states, and calculate path costs. Iterative Deepening Search (IDS) and backtracking are used to balance memory requirements and completeness.

B. Plagiarism Detection using A Search*

Plagiarism detection can be modeled as a search problem where states represent partial alignments of sentences between two documents. The A* algorithm is employed with the following definitions:

- **Initial State:** Start of both documents.
- **Goal State:** Complete alignment of all sentences.
- **Transition Function:** Align, skip, or mismatch sentences.
- **Cost Function** $g(n)$**:** Edit distance between aligned sentences.
- **Heuristic** $h(n)$**:** Estimated remaining edit distance.

This formulation ensures that the search process is admissible and optimal.

‘‘‘latex

III. Methodology

A. Graph Search

The graph search algorithm maintains a frontier queue of unexplored nodes and an explored set. At each iteration, a node is expanded; if it is the goal, the solution is returned. Otherwise, unexplored successors are generated and added to the frontier.

B. Environment Functions for the 8-Puzzle

get_possible_moves(state): Returns legal moves.
apply_move(state, move): Generates successor states.
goal_test(state): Checks if the goal configuration is reached.
path_cost(): Assigns uniform cost (1) per move.

C. Iterative Deepening Search (IDS)

IDS combines depth-first search with breadth-first search. The depth limit begins at zero and increases until the goal is found, ensuring completeness and optimality with uniform cost.

D. A* for Plagiarism Detection

Each state represents an alignment between parts of two documents. A* expands states in order of increasing evaluation function, $f(n) = g(n) + h(n)$. The cost is edit distance, and the heuristic estimates the remaining distance. After alignment, sentences with edit distances below a threshold are marked as plagiarized.

IV. Pseudocode for A* Search

The following pseudocode illustrates the alignment process using A* search:

[h] A* Search for Sentence Alignment [1] Initialize $open\_set$ as a priority queue ordered by $f(n) = g(n) + h(n)$ $g[start] = 0$, $f[start] = h(start)$ $open\_set$ not empty $current \leftarrow$ node with lowest $f(n)$ $current =$ goal reconstruct_path(current) neighbor in neighbors(current) $tentative\_g \leftarrow g[current] + cost(current, neighbor)$ $tentative\_g < g[neighbor]$ $parent[neighbor] \leftarrow current$ $g[neighbor] \leftarrow tentative\_g$ $f[neighbor] \leftarrow g[neighbor] + h(neighbor)$ Add $neighbor$ to $open\_set$ failure

V. Python Implementation

For clarity and reproducibility, we provide a simplified Python implementation of the A* algorithm for plagiarism detection. The edit distance between sentences is used as the cost function.

```
import heapq

def edit_distance(s1, s2):
    m, n = len(s1), len(s2)
    dp = [[0]*(n+1) for _ in range(m+1)]
    for i in range(m+1):
        for j in range(n+1):
            if i == 0:
                dp[i][j] = j
            elif j == 0:
                dp[i][j] = i
            elif s1[i-1] == s2[j-1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = 1 + min(dp[i-1][j],
                                   dp[i][j-1],
                                   dp[i-1][j-1])
    return dp[m][n]

def a_star(start, goal, neighbors, h):
    open_set = [(h(start), 0, start, [])]
    visited = set()
    while open_set:
        f, g, current, path =
            heapq.heappop(open_set)
        if current == goal:
            return path
        if current in visited:
            continue
        visited.add(current)
        for neighbor in neighbors(current):
            cost = g + edit_distance(current,
                neighbor)
            heapq.heappush(open_set,
                (cost + h(neighbor),
                 cost,
                 neighbor,
                 path+[neighbor]))
    return None
```

‘‘‘

VI. Results and Discussion

A. Puzzle-8

The graph search agent successfully solved the 8-puzzle across different goal depths. BFS consistently provided the shortest paths, while IDS minimized memory requirements. The use of hash tables reduced redundant state exploration. However, as depth increased, both time and space complexity grew exponentially, confirming theoretical predictions.

B. Plagiarism Detection

Experiments were conducted with varying document similarities:

Identical documents: Edit distance = 0, plagiarism = 100

Minor word changes: Edit distance = 0–5

Different documents: Edit distance 70

Partial overlaps: Edit distance = 20–40

The A* algorithm efficiently minimized alignment cost, making it robust against minor modifications such as word substitutions and effective in plagiarism detection.

VII. Conclusion

This study demonstrates the versatility of search algorithms in AI. The graph search agent effectively solved the 8-puzzle using BFS and IDS, while A* successfully detected plagiarism by aligning sentences based on edit distance. These experiments confirm the importance of heuristic search in both discrete state problems and real-world natural language tasks.

References

[1] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed. Pearson, 2016.

[2] R. E. Korf, “Depth-first iterative-deepening: An optimal admissible tree search,” Artificial Intelligence, vol. 27, no. 1, pp. 97–109, 1985.

[3] G. Navarro, “A guided tour to approximate string matching,” ACM Computing Surveys, vol. 33, no. 1, pp. 31–88, 2001.