Lab II:-Graph Search Agent: State Space Exploration Using Queue and Hash Table

Soham Naukudkar Roll No: 202351135
Shreyash Borkar Roll No: 202351136
Siddhikesh Gavit Roll No: 202351040

***Abstract*—This report presents the implementation and application of the A* search algorithm in the context of Artificial Intelligence. The first part focuses on understanding A* as a heuristic-based search method used to find optimal solutions in state-space problems. The second part applies this theoretical understanding to a practical problem — designing a plagiarism detection system that aligns text between two documents using A* search and edit distance. The objective is to demonstrate how heuristic search can be adapted for text alignment and similarity detection tasks efficiently. Experimental evaluation is conducted on several test cases to analyze the performance and accuracy of the proposed system.**

***Index Terms*—A* Search, Artificial Intelligence, Edit Distance, Heuristic Search, Levenshtein Distance, Plagiarism Detection, Text Alignment**

I. Week 2 Lab Objective

The learning objective of this lab is to design a graph search agent and understand the use of hash tables and queues in state-space search.

A. Background Reading for Week 2

This work is based on the concepts of state-space search detailed by Russell and Norvig [0] and Khemani [0]. Foundational principles are covered in Chapters 2-3 of *Artificial Intelligence: A Modern Approach* and Chapter 2 of *A First Course in Artificial Intelligence*.

II. Introduction

Artificial Intelligence (AI) aims to design systems capable of intelligent decision-making and problem-solving. Search algorithms play a foundational role in AI by exploring possible solutions in a defined state-space. Among them, A* (A-star) is one of the most widely used informed search algorithms that combines path-cost with heuristic estimates to find optimal solutions efficiently.

In this laboratory work, we first explore the A* search algorithm conceptually and then apply it practically in a plagiarism detection system. The plagiarism detection system aligns sentences from two documents using A* to minimize their edit distance, effectively identifying similar or copied text. The full Python implementation is available on GitHub.

III. Search Algorithms in AI

Search algorithms are fundamental to solving AI problems such as pathfinding, optimization, and planning. These can be broadly classified as:

- **Uninformed Search:** Algorithms such as Breadth-First Search (BFS), Depth-First Search (DFS), and Uniform Cost Search explore the search space without domain knowledge.
- **Informed Search:** Algorithms like Greedy Best-First and A* use heuristic information to estimate proximity to the goal and hence improve efficiency.

A. A* Search Overview

A* search combines the advantages of Uniform Cost Search and Greedy Best-First Search using the cost function:

$$f(n) = g(n) + h(n) \tag{1}$$

where:

- $g(n)$: Cost from the start node to the current node.
- $h(n)$: Heuristic estimate of the cost to reach the goal.

A* guarantees the optimal solution if the heuristic function is admissible.

B. Applications of A* Search

A* is widely used in robotics, route optimization, pathfinding in games, and other domains requiring efficient shortest-path computation. In this lab, the same principle is adapted to text alignment and plagiarism detection.

IV. Problem Statement for Submission

Objective: Implement a plagiarism detection system using the A* search algorithm applied to text alignment. The system should identify similar or identical sequences of text between two documents by aligning their sentences or paragraphs and detecting potential plagiarism.

A. Background

Plagiarism detection involves comparing two documents to identify shared sequences of words or phrases that might indicate copying. Text alignment is a common approach to this, where one text is aligned with another based on similarity measures such as edit distance. The A* algorithm can efficiently find the optimal alignment path minimizing differences between sentences.

B. Prerequisites

- Basic understanding of the A* algorithm.
- Familiarity with string similarity and edit distance.
- Python programming knowledge.

V. Approach and Methodology

A. State Representation

Each state represents a partial alignment between two documents and includes:

$$State = (i, j, g(n))$$

where i and j are indices of the current sentences being compared.

B. Initial and Goal States

- **Initial:** Beginning of both documents (first sentences).
- **Goal:** All sentences from both documents aligned.

C. Transition Function

From each state, transitions include:

- Align current sentences: $(i+1, j+1)$
- Skip a sentence in Document 1: $(i+1, j)$
- Skip a sentence in Document 2: $(i, j+1)$

D. Cost Function $g(n)$

The cumulative cost of sentence alignments measured by the Levenshtein edit distance.

E. Heuristic Function $h(n)$

Estimated minimum cost of aligning remaining sentences (e.g., assuming all unmatched sentences incur minimal edit distance).

VI. Algorithm Design

Algorithm 1: A* Search for Plagiarism Detection

[1] AStarAlign$D1, D2$ $start \leftarrow (0,0)$; $goal \leftarrow (|D1|, |D2|)$ Initialize *open* priority queue ordered by $f(n) = g(n) + h(n)$ $g[start] \leftarrow 0$; $came\_from[start] \leftarrow$ NULL **while** *open not empty* **do**

$current \leftarrow$ node with lowest f **if** $current == goal$ **then**

return ReconstructPath($came\_from, current$) **forall** *valid transitions* **do**

Compute $new\_cost = g[current] +$ EditDistance($current$) **if** *neighbor not visited or* $new\_cost < g[neighbor]$ **then**

$g[neighbor] \leftarrow new\_cost$ $f[neighbor] \leftarrow g[neighbor] +$ Heuristic($neighbor$) Add $neighbor$ to *open*

VII. Implementation

A. Text Preprocessing

- Tokenize documents into sentences.
- Convert to lowercase and remove punctuation.
- Optionally remove stopwords.

B. Edit Distance Function

```
def levenshtein(a, b):
    n, m = len(a), len(b)
    dp = [[0]*(m+1) for _ in range(n+1)]
    for i in range(n+1): dp[i][0] = i
    for j in range(m+1): dp[0][j] = j
    for i in range(1, n+1):
        for j in range(1, m+1):
            cost = 0 if a[i-1] == b[j-1] else
                1
            dp[i][j] = min(dp[i-1][j]+1, dp[i
                ][j-1]+1, dp[i-1][j-1]+cost)
    return dp[n][m]
```

VIII. Sample Plagiarism Report

TABLE I
Sample Alignment and Plagiarism Detection

Sentence 1	Sentence 2	Edit Distance	Plagiarism Likelihood
Text from Doc1	Text from Doc2	0	High
Another sentence	Modified sentence	2	Medium
Different text	No similarity	10	Low

IX. Conclusion

This lab experiment successfully demonstrates both the theoretical and practical aspects of the A* search algorithm. In the theoretical part, A* was studied as an optimal heuristic search technique. In the applied part, it was implemented for plagiarism detection through sentence alignment, leveraging edit distance and heuristic optimization. The system effectively detects textual similarities and can be extended for large-scale document comparison.

X. References

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Hoboken, NJ, USA: Pearson, 2020.

[2] D. Khemani, *A First Course in Artificial Intelligence*. New Delhi, India: McGraw Hill, 2013.

[3] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. draft. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/