Artificial Intelligence Lab Report: 2
Plagiarism Detection Using A* Search Algorithm

Anuj Saha, Gadige Nikhil, Divyanshu Ghosh
Indian Institute of Information Technology Vadodara
{202351010, 202351037, 202351036}@iiitvadodara.ac.in

***Abstract*—In this report, we present our implementation of a plagiarism detection system that uses A* search for text alignment. We implemented text preprocessing techniques, used Levenshtein distance for measuring similarity, and developed an A* search algorithm with an admissible heuristic. We validated our system using four test cases. These included identical documents which gave us a cost of zero, slightly modified documents with a cost of eight, completely different documents resulting in a cost of forty-four, and partially overlapping documents with a cost of thirty-six. Our results demonstrate that we can effectively detect plagiarism through optimal alignment and edit distance computation.**

***Index Terms*—Plagiarism Detection, A* Search, Levenshtein Distance, Text Alignment, Edit Distance, Graph Search, Natural Language Processing**

I. Introduction

Plagiarism detection is the process of identifying shared text sequences between documents. In this lab, we implemented a system that uses A* search [1] for text alignment and Levenshtein distance [2] for measuring similarity between texts.

We chose A* search because it combines the optimality guarantees of Dijkstra's algorithm with heuristic guidance through the function $f(n) = g(n) + h(n)$. Here, $g(n)$ represents the actual cost we have incurred so far, and $h(n)$ is our heuristic estimate of the remaining cost. We formulated text alignment as a state space search problem where each state (i, j) represents our current positions in the two documents we are comparing. We defined transitions that include matching sentences or skipping sentences in either document, and we measured costs using edit distances.

We used Levenshtein distance to quantify the minimum number of character edits needed to transform one string into another. These edits include insertions, deletions, and substitutions. This metric provided us with a robust way to measure similarity for plagiarism detection.

II. Problem Statement

Our objective in this lab was to implement a plagiarism detection system that meets several core requirements and handles multiple test scenarios.

A. Core Requirements

We needed to implement the following components:

- **Text Preprocessing**: We had to tokenize documents into sentences and normalize the text. This included converting everything to lowercase and handling punctuation properly.
- **Edit Distance Computation**: We implemented Levenshtein distance to measure how similar sentence pairs are to each other.
- **A* Search Implementation**: We developed a state space search with several key elements. We represented states as (i, j) indices in the two documents. We started from the initial state $(0, 0)$ at the beginning of both documents and aimed to reach the goal state (m, n) at the end of both documents. We defined a transition function that could match sentences, skip sentences in the first document, or skip sentences in the second document. We used a cost function based on accumulated edit distance and designed an admissible heuristic function to estimate the remaining cost.
- **Plagiarism Detection**: We needed to identify sentence pairs with low edit distance as suspicious matches that might indicate plagiarism.

B. Test Cases

We designed four test cases to validate our system.

Test Case 1 - Identical Documents: We tested our system with two identical documents. We expected these to align perfectly with zero total cost.

Test Case 2 - Slightly Modified Document: We tested documents that had minor modifications such as synonyms and word order changes. We expected these to align with low edit distance.

Test Case 3 - Completely Different Documents: We tested completely unrelated documents. We expected these to produce high edit distances that would indicate no plagiarism.

Test Case 4 - Partial Overlap: We tested documents that had some overlapping content. We expected our system to identify the overlapping portions with low edit distance while recognizing the different portions.

III. Methodology

A. Text Preprocessing

We built a preprocessing pipeline that transforms raw text into normalized sentence representations. Here is how we implemented it:

Algorithm 1 Text Preprocessing

Require: raw text string $text$
Ensure: list of normalized sentences
$sentences \leftarrow$ split by delimiters $\{., !, ?\}$
for each s in $sentences$ **do**
  $s \leftarrow$ lowercase(s)
  $s \leftarrow$ strip_whitespace(s)
  if $s \neq$ empty **then**
    add s to result
  end if
end for
return normalized sentences

We used a regular expression pattern for sentence splitting that handles common abbreviations and avoids splitting on periods that appear within words.

B. Levenshtein Distance

We implemented the Levenshtein distance algorithm using dynamic programming to compute the minimum edit distance efficiently. Here is our implementation:

Algorithm 2 Levenshtein Distance

Require: strings s_1, s_2
Ensure: edit distance d
if $|s_1| < |s_2|$ **then**
  return levenshtein_distance(s_2, s_1)
end if
if $|s_2| = 0$ **then**
  return $|s_1|$
end if
Initialize $prev\_row[0..|s_2|] \leftarrow [0, 1, 2, ..., |s_2|]$
for $i = 0$ to $|s_1| - 1$ **do**
  $curr\_row[0] \leftarrow i + 1$
  for $j = 0$ to $|s_2| - 1$ **do**
    $insert \leftarrow prev\_row[j + 1] + 1$
    $delete \leftarrow curr\_row[j] + 1$
    $subst \leftarrow prev\_row[j] + (s_1[i] \neq s_2[j])$
    $curr\_row[j + 1] \leftarrow \min(insert, delete, subst)$
  end for
  $prev\_row \leftarrow curr\_row$
end for
return $prev\_row[|s_2|]$

We achieved a time complexity of $O(mn)$ where $m = |s_1|$ and $n = |s_2|$. We optimized space complexity to $O(\min(m, n))$ by only keeping track of the current and previous rows.

C. Heuristic Function

We designed a heuristic function to estimate the remaining alignment cost. Our heuristic is:

$$h(i, j) = |remaining_1 - remaining_2| \quad (1)$$

Here, $remaining_1 = m - i$ and $remaining_2 = n - j$ represent the number of unprocessed sentences remaining in each document.

Why Our Heuristic is Admissible: We proved that this heuristic is admissible by considering three cases. First, if $remaining_1 = remaining_2$, then all remaining sentences could potentially match perfectly with a cost of zero. Second, if $remaining_1 > remaining_2$, then at minimum we must skip $|remaining_1 - remaining_2|$ sentences. Third, our heuristic never overestimates because it assumes perfect matches for all aligned sentences. This means our heuristic satisfies the admissibility requirement.

D. A* Search for Text Alignment

We implemented the A* search algorithm for finding the optimal text alignment as follows:

How We Designed the Cost Function: We designed two types of costs. For matching sentences, we used the Levenshtein distance between them. For skipping a sentence, we used the length of the skipped sentence as a penalty. This penalizes unaligned content and encourages the algorithm to find good matches.

E. Plagiarism Detection

After we obtained the optimal alignment from our A* search, we identified suspicious pairs using this algorithm:

We chose a threshold of $\theta = 10$ based on empirical testing. This value gave us a good balance between being sensitive enough to catch plagiarism and being specific enough to avoid false positives.

IV. Results

A. Implementation Details

We implemented our system in Python. We used the `heapq` module for priority queue operations, the `re` module for regular expression-based sentence splitting, and we wrote custom implementations of the Levenshtein distance algorithm and the A* search algorithm.

B. Test Case Results

We summarize our test results in the following table:

TABLE I
Plagiarism Detection Test Results

Test Case	Total Cost	Suspicious Pairs
Identical Documents	0	1
Slightly Modified	8	1
Different Documents	44	0
Partial Overlap	36	≥ 1

1) Test Case 1: Identical Documents: We tested our system with the following identical documents:

- Doc1: "The quick brown fox jumps over the lazy dog."
- Doc2: "The quick brown fox jumps over the lazy dog."

We obtained a total cost of zero, which indicates a perfect match. Our A* algorithm aligned the single sentence with zero edit distance. This correctly identified the identical content.

Algorithm 3 A* Text Alignment

Require: sentence lists doc_1, doc_2
Ensure: optimal alignment path, total cost

Initialize priority queue pq with $(f = 0, g = 0, i = 0, j = 0, path = [])$
Initialize visited set $V \leftarrow \emptyset$
while $pq \neq \emptyset$ **do**
  $(f, g, i, j, path) \leftarrow$ pop_min(pq)
  if $(i, j) \in V$ **then**
    continue
  end if
  $V \leftarrow V \cup \{(i, j)\}$
  if $i = |doc_1|$ AND $j = |doc_2|$ **then**
    return $path$, g
  end if
  // Transition 1: Match sentences
  if $i < |doc_1|$ AND $j < |doc_2|$ **then**
    $cost \leftarrow$ levenshtein($doc_1[i]$, $doc_2[j]$)
    $g' \leftarrow g + cost$
    $h' \leftarrow$ heuristic($i + 1$, $j + 1$, doc_1, doc_2)
    push(pq, $(g' + h', g', i + 1, j + 1, path + [(i, j, cost, 'MATCH')])$)
  end if
  // Transition 2: Skip sentence in doc1
  if $i < |doc_1|$ **then**
    $cost \leftarrow |doc_1[i]|$
    $g' \leftarrow g + cost$
    $h' \leftarrow$ heuristic($i + 1$, j, doc_1, doc_2)
    push(pq, $(g' + h', g', i + 1, j, path + [(i, None, cost, 'SKIP1')])$)
  end if
  // Transition 3: Skip sentence in doc2
  if $j < |doc_2|$ **then**
    $cost \leftarrow |doc_2[j]|$
    $g' \leftarrow g + cost$
    $h' \leftarrow$ heuristic(i, $j + 1$, doc_1, doc_2)
    push(pq, $(g' + h', g', i, j + 1, path + [(None, j, cost, 'SKIP2')])$)
  end if
end while
return [], ∞

Algorithm 4 Detect Plagiarism

texts $text_1$, $text_2$, threshold θ
Ensure: plagiarism report

$s_1 \leftarrow$ preprocess($text_1$)
$s_2 \leftarrow$ preprocess($text_2$)
$alignment$, $total\_cost \leftarrow$ a_star_alignment(s_1, s_2)
$suspicious \leftarrow []$
for each $(i, j, cost, type)$ in $alignment$ **do**
  if $type =$ 'MATCH' AND $cost \leq \theta$ **then**
    add $\{i, j, s_1[i], s_2[j], cost\}$ to $suspicious$
  end if
end for
return $\{total\_cost, alignment, suspicious\}$

2) Test Case 2: Slightly Modified Document: We tested our system with slightly modified documents:

- Doc1: "The quick brown fox jumps over the lazy dog."
- Doc2: "The fast brown fox leaps over the lazy dog."

We obtained a total cost of eight. We analyzed the modifications and found that changing "quick" to "fast" gave us a Levenshtein distance of approximately four, and changing "jumps" to "leaps" also gave us a distance of approximately four. The low total cost correctly identified this as a suspicious similarity despite the word substitutions.

3) Test Case 3: Completely Different Documents: We tested our system with completely different documents:

- Doc1: "The quick brown fox jumps over the lazy dog."
- Doc2: "Lorem ipsum dolor sit amet, consectetur adipiscing elit."

We obtained a total cost of forty-four. This high cost reflects substantial differences in vocabulary, sentence structure, and semantic content. Our system correctly indicated that there is no plagiarism between these documents.

4) Test Case 4: Partial Overlap: We tested our system with documents that have partial overlap:

- Doc1: "The quick brown fox jumps over the lazy dog. It is a sunny day."
- Doc2: "It is a sunny day. The quick brown fox jumps over the lazy dog."

We obtained a total cost of thirty-six. The reordered sentences created alignment challenges for our algorithm. Our A* search had to skip sentences to find the best alignment, which resulted in a moderate cost. We detected at least one suspicious pair, which demonstrates that our system can identify overlapping content even when the sentences are reordered.

C. Performance Analysis

We analyzed the performance characteristics of our implementation.

Time Complexity: Our preprocessing step takes $O(n)$ time where n is the text length. Computing the Levenshtein distance takes $O(L_1 \times L_2)$ time for each sentence pair. Our A* search takes $O(b^d)$ time in the worst case, where b is the branching factor (which is three in our case) and d is the solution depth. Overall, our worst-case time complexity is $O(m \times n \times L^2)$ for documents with m and n sentences of length L.

Space Complexity: We use $O(m \times n)$ space for storing visited states plus additional space for the priority queue.

V. Discussion

A. A* Search Effectiveness

We found that the A* algorithm finds optimal alignments efficiently by using our admissible heuristic. This heuristic ensures optimality while reducing the number of states we need to explore. The priority queue maintains states ordered by $f(n) = g(n) + h(n)$, which guarantees that the first path we find to the goal is optimal.

B. Levenshtein Distance Analysis

We observed that Levenshtein distance provides character-level precision and satisfies important metric properties. These include symmetry ($d(s_1, s_2) = d(s_2, s_1)$) and the triangle inequality. However, we also noticed limitations. The distance cannot detect semantic similarity when synonyms are used, and it does not handle word reordering efficiently.

C. Limitations and Solutions

We identified several limitations in our approach. Our sequential state space (i, j) enforces ordering, which causes high costs when content is reordered. We propose several solutions to this problem. We could implement sentence-level permutation invariance, use semantic embeddings like BERT or Word2Vec, or implement a sliding window comparison approach.

We also found that our threshold of $\theta = 10$ was chosen empirically and might not work well for all document types. We suggest that adaptive thresholding using $\theta_{normalized} = \alpha \times \max(|s_1|, |s_2|)/2$ could improve accuracy across different document lengths.

VI. Conclusion

In this lab, we successfully implemented a plagiarism detection system using A* search and Levenshtein distance. We achieved several key results. We found optimal alignments using our admissible heuristic. We accurately detected plagiarism across our four test cases with costs of zero, eight, forty-four, and thirty-six. We efficiently implemented our system using dynamic programming and priority queues.

Our implementation demonstrates several core AI concepts. We formulated the problem as a state space search. We designed an admissible heuristic function. We engineered an appropriate cost function. We applied dynamic programming techniques. The applications of our work extend beyond plagiarism detection. Our techniques could be used for code similarity detection, document version comparison, and duplicate content detection.

Limitations We Found: We found that our sequential alignment approach struggles with reordered content. Our character-level distance metric ignores semantic relationships. We had to tune our threshold manually, which is not ideal.

Future Work We Propose: We suggest integrating semantic embeddings like BERT or Word2Vec to better capture meaning. We could implement hierarchical alignment to handle document structure better. We should develop adaptive thresholding mechanisms that work across different document types. We could apply parallel processing to improve scalability for large document collections. This exercise successfully bridged classical AI search techniques with practical natural language processing applications.

References

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

[2] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. Draft available online, 2020.