Plagiarism Detection using A* Alignment and Levenshtein Distance

Arpit Jain

September 21, 2025

A. Objective

The objective of this project is to develop a **sentence-level plagiarism detection system** that can align two documents of arbitrary length and compute a **quantitative plagiarism score**.

Unlike simple string-matching systems, our approach uses:

1. **A* search** to optimally align sentences between two documents.
2. **Levenshtein distance** as the similarity metric to evaluate sentence-level matches.
3. **Skip penalties** to discourage unnecessary omission of sentences.

The system outputs:

- Alignment table showing matches, skips, and similarity scores.
- Quick statistics summarizing exact, strong, and moderate similarities.
- An overall plagiarism coverage score.

B. Problem Statement

Given two documents, A and B, in `.txt`, `.pdf`, or `.docx` format, the task is to **align sentences optimally** and compute similarity-based plagiarism measures.

Challenges

- Sentences may not appear in the same order.
- Sentences may be paraphrased.
- One document may contain extra or missing sentences.
- File formats differ and must be normalized into plain text.

State Space

Let Document $A = [a_1, a_2, \ldots, a_n]$ contain n sentences, and Document $B = [b_1, b_2, \ldots, b_m]$ contain m sentences.

- The alignment problem can be seen as navigating a **grid of states**, where each state corresponds to a pair (i, j) meaning we are currently at the i-th sentence of A and the j-th sentence of B.
- State space size $= n \times m$.

Goal State

Reach (n, m), i.e., all sentences from both documents are considered.

Optimization Criterion

We minimize the total alignment cost:

$$C(S) = \sum \text{LevenshteinCost}(a_i, b_j) + \text{SkipPenalty} \times (\#skips)$$

The alignment with the lowest cost corresponds to the **most probable plagiarism mapping**.

C. Methodology

1. File Reading & Normalization

- Supports `.txt`, `.pdf`, and `.docx`.
- Extracted text is cleaned by:
  - Lowercasing.
  - Removing punctuation and control characters.
  - Replacing multiple spaces with one.

2. Sentence Tokenization

Documents are split into sentences using regex rules:

- Split on `.`, `!`, `?` followed by whitespace.
- Keep meaningful sentences, discard empty ones.

Example: Input: "AI is evolving. It impacts many fields." Output: ["AI is evolving", "It impacts ma

3. Sentence Similarity: Levenshtein Distance

The **Levenshtein edit distance** measures dissimilarity:

$$d(a, b) = \min(\#insertions, \#deletions, \#substitutions \text{ to transform } a \to b)$$

Similarity percentage:

$$\text{Sim}(a, b) = 100 - \frac{d(a, b)}{\max(|a|, 1)} \times 100$$

Classification:

- Exact match $\geq 80\%$
- Strong similarity $\geq 60\%$
- Moderate similarity $\geq 40\%$
- Low similarity $< 40\%$

4. A* Alignment Search

Alignment is treated as a **pathfinding problem**:

- **Nodes:** (i, j) position in documents.
- **Edges:** Possible transitions:
  - MATCH: align a_i with b_j.
  - SKIP_A: skip sentence in A.
  - SKIP_B: skip sentence in B.
- **Cost:** edit distance (for match) or skip penalty (200).
- **Frontier:** priority queue ordered by cumulative cost g.
- **Path reconstruction:** backtrack from (n, m) to $(0, 0)$.

5. Plagiarism Scoring

Plagiarism score:

$$\text{Score}(\%) = \frac{\text{Exact} + \text{Strong} + \text{Moderate Matches}}{|A|} \times 100$$

Interpretation:

- High plagiarism $\geq 70\%$
- Medium plagiarism $\geq 40\%$
- Low plagiarism $< 40\%$

D. Pseudocode

```
FUNCTION Plagiarism_Check(fileA, fileB):
    textA = Read_File(fileA)
    textB = Read_File(fileB)

    sentencesA = Normalize(Tokenize(textA))
    sentencesB = Normalize(Tokenize(textB))

    start = (0, 0)
    goal = (len(sentencesA), len(sentencesB))
    frontier = PriorityQueue()
    gscore = {start: 0}
    came_from = {}

    frontier.push((0, start))

    WHILE frontier is not empty:
        (f, (i,j)) = frontier.pop_lowest()
```

```
IF (i,j) == goal:
    path = Reconstruct_Path(came_from, goal)
    RETURN path

# Transition: MATCH
IF i < len(sentencesA) AND j < len(sentencesB):
    cost = Levenshtein(sentencesA[i], sentencesB[j])
    new_state = (i+1, j+1)
    Update(new_state, gscore, cost)

# Transition: SKIP_A
IF i < len(sentencesA):
    cost = SkipPenalty
    new_state = (i+1, j)
    Update(new_state, gscore, cost)

# Transition: SKIP_B
IF j < len(sentencesB):
    cost = SkipPenalty
    new_state = (i, j+1)
    Update(new_state, gscore, cost)
```

E. Results

Example Run

Input: Two documents of 5 sentences each.

Alignment Summary:

- Total alignment cost = 192.1
- Matches = 5
- Exact matches = 0
- Strong similarities = 2
- Moderate similarities = 3

Quick Stats:

- Exact match ratio = 0%
- Strong similarity ratio = 40%
- Moderate similarity ratio = 60%
- Plagiarism Score = **100%** → **High**

Observations

- Paraphrased sentences were still detected as strong/moderate similarity.
- Extra sentences caused SKIP_A or SKIP_B transitions.
- Skip penalty tuning is critical: too low → excessive skipping, too high → forced mismatches.

F. Conclusion

The plagiarism detection system demonstrates how **search-based alignment** can outperform naive word matching.

Strengths

- Works across multiple formats (`txt`, `pdf`, `docx`).
- Handles paraphrasing.
- Provides interpretable outputs: alignment table, cost, and similarity labels.
- Uses classical algorithms (A* + edit distance), ensuring transparency.

Limitations

- Large documents lead to higher computational time.
- Semantic plagiarism (reworded meaning) is harder to detect.
- All sentences treated equally, regardless of importance.

Future Work

- Replace Levenshtein with semantic embeddings (e.g., BERT, SBERT).
- Add visual plagiarism heatmaps.
- Weight sentences differently (e.g., topic sentences more important).