Plagiarism Detection System in Academic Papers Utilizing String Matching Techniques

Muhammad Raihaan Perdana - 13523124
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: perdanaraihan96@gmail.com, 13523124@std.stei.itb.ac.id

***Abstract*— Automated plagiarism detection is essential for maintaining academic integrity, but its performance depends on the efficiency of the underlying text-searching algorithms. This paper provides a comparative analysis of three foundational string matching algorithms—Brute-Force, Knuth-Morris-Pratt (KMP), and Boyer-Moore—to determine their suitability for detecting direct plagiarism in academic texts. The algorithms were implemented in a conceptual detection system and tested on a controlled dataset containing both original and copied sentences, with performance measured by the total number of character comparisons. Experimental results show that the Boyer-Moore algorithm is overwhelmingly the most efficient, reducing comparisons by nearly ninety percent compared to the baseline. Unexpectedly, the KMP algorithm performed worse than the simple Brute-Force approach, as the lack of repetitive sub-patterns in natural language text nullified its primary optimization. This study concludes that for processing natural language, the Boyer-Moore algorithm is a superior foundational choice, and demonstrates that real-world data characteristics can be more influential on practical performance than theoretical worst-case complexity.**

Keywords*— Plagiarism Detection; String Matching; Brute-Force Algorithm; Knuth-Morris-Pratt; Boyer-Moore; Algorithm Analysis.

I. INTRODUCTION

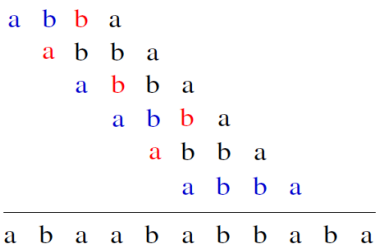

Fig 1.1 String Matching Illustration
(Source: medium)

Academic integrity is the cornerstone of all scholarly pursuits. Without it, the trust in research and education erodes. A primary challenge to this integrity today is plagiarism—the act of using someone else's words or ideas without giving them proper credit. This issue extends beyond simple academic dishonesty; it actively harms the learning process by discouraging students from developing their own critical thinking and writing skills. In an age where digital text from countless sources is instantly accessible, the "copy-paste" culture has made it easier than ever for plagiarism to occur, creating a significant and persistent problem for universities committed to upholding high standards of originality.

For an educator, ensuring the originality of student work in this environment has become a monumental task. Manually checking a single paper against the billions of pages on the internet, not to mention vast libraries of books and academic journals, is simply not feasible. The sheer scale of the problem demands an automated approach. This is where computer science offers a powerful set of tools. At its heart, detecting direct textual copying is a computational task. We can frame the problem as a massive search operation: looking for specific sequences of text (a "pattern" taken from a student's paper) within a much larger body of source material (the "text"). The central question then becomes: what is the most effective and efficient algorithmic strategy to perform this large-scale search?

To answer this, we turn to the field of algorithm strategies, specifically focusing on techniques designed for string matching. This paper will examine and compare three foundational algorithms that offer different approaches to solving this search problem. We start with the Brute-Force algorithm, a direct and intuitive method that, while simple to understand, often proves to be slow in practice. It serves as an important baseline for measuring performance. Next, we analyze the Knuth-Morris-Pratt (KMP) algorithm, a more sophisticated technique that cleverly pre-processes the pattern to learn from mismatches, allowing it to skip sections of the text and avoid redundant comparisons. Finally, we investigate the Boyer-Moore algorithm, which takes a counter-intuitive yet highly effective approach by searching for the pattern from right-to-left. This method often allows the algorithm to jump across large sections of the source text, making it one of the fastest algorithms in many practical applications.

The primary goal of this study is to bridge the theory of these algorithms with the practical challenge of plagiarism detection. To achieve this, our research will proceed with the following objectives:

1. To propose a clear, conceptual framework for a basic plagiarism detection system that utilizes string matching as its core engine.

2. To simulate how the Brute-Force, KMP, and Boyer-Moore algorithms would function within this framework, analyzing the step-by-step process of how each one handles the search.
3. To conduct a detailed comparative analysis, using the simulation results to evaluate the algorithms based on their speed, computational efficiency, and overall suitability for building a real-world plagiarism detection tool.

The rest of this paper is structured to guide the reader through our investigation. Section II delves into the theoretical foundations of each of the three string matching algorithms. Section III lays out the methodology of our study, including the design of our conceptual system and the dataset used for simulation. Section IV presents the results of our analysis and discusses the performance of each algorithm. Finally, Section V offers our conclusions and suggests ideas for future research in this area.

II. Theoretical Foundation

A. String Matching

String matching is one of the fundamental concepts in computer science that deals with searching for a specific pattern within a text. Formally, string matching is defined as the process of finding occurrences of string P (pattern) with length m within string T (text) with length n, where typically $m \leq n$ [1]. In this context, text T can be viewed as a collection of characters T[0], T[1], ..., T[n-1], while pattern P is a collection of characters P[0], P[1], ..., P[m-1].

The string matching process aims to find all positions i where P[0..m-1] = T[i..i+m-1], or in other words, all positions where pattern P appears as a substring of text T. For example, if T = "ABCABCAB" and P = "ABC", then pattern P is found at position 0 and position 3 in text T.

The efficiency of string matching algorithms is crucial in various applications, including plagiarism detection systems. In the context of plagiarism detection, text T can be a reference source document, while pattern P can be a sentence or phrase from the document being examined. The more efficient the string matching algorithm used, the faster the system can detect potential plagiarism [2].

There are various string matching algorithms with different characteristics and complexities. These algorithms can be categorized based on their matching strategies, ranging from simple approaches that compare every possible position to more sophisticated approaches that utilize preprocessed information to avoid unnecessary comparisons.

B. The Brute-Force Algorithm

The Brute-Force algorithm, also known as the naive algorithm, represents the most straightforward approach to solving the string matching problem. This algorithm works by attempting to match pattern P at every possible position in text T sequentially from left to right [3].

The working mechanism of the Brute-Force algorithm can be explained in the following steps:

1. Start from the first position of text T (index 0)
2. Compare the first character of pattern P with the character of text T at that position
3. If they match, continue comparing the next characters until all pattern characters are exhausted or a mismatch is found
4. If all pattern characters match, record that position as a result
5. Shift the search position one character to the right and repeat the process

The main advantage of the Brute-Force algorithm lies in its simplicity. This algorithm is easy to understand and implement because it does not require preprocessing or complex additional data structures. This makes the Brute-Force algorithm suitable for simple applications or for understanding the basic concepts of string matching.

However, the Brute-Force algorithm has significant weaknesses in terms of time efficiency. At worst case, this algorithm has a time complexity of O(mn), where m is the length of the pattern and n is the length of the text. The worst case occurs when almost all pattern characters match the text, but the last character is always different, so the algorithm must perform maximum comparisons at each position [4].

For example, if T = "AAAAAAAAAB" (length 10) and P = "AAAB" (length 4), then the algorithm will perform $7 \times 4 = 28$ character comparisons before finding the pattern at position 6. In the best case, when the pattern is found at the beginning of the text or does not exist at all, the time complexity can reach O(n).

The main weakness of the Brute-Force algorithm is performing many unnecessary comparisons. When a mismatch is found at a certain position, the algorithm does not utilize information obtained from previous comparisons and immediately shifts the pattern by only one position, even though it might be possible to shift further based on the mismatched character..

C. The Knuth-Morris-Pratt (KMP) Algorithm

The Knuth-Morris-Pratt (KMP) algorithm is a string matching algorithm developed to overcome the weaknesses of the Brute-Force algorithm by avoiding unnecessary character comparisons. This algorithm was discovered by Donald Knuth, James H. Morris, and Vaughan Pratt in 1977 and became one of the most efficient string matching algorithms [5].

The main concept behind the KMP algorithm is utilizing information from pattern prefixes that are also suffixes (proper prefixes that are also proper suffixes) to determine how far the pattern can be shifted when a mismatch occurs. This information is stored in an array called the border function or failure function [6].

The border function $\pi(j)$ for pattern P is defined as the length of the longest proper prefix of P[0..j] that is also a suffix

of P[0..j]. For example, for pattern P = "ABCAB", the border function is:

- $\pi(0) = 0$ (no proper prefix for a single character)
- $\pi(1) = 0$ (no proper prefix of "AB" that is also a suffix)
- $\pi(2) = 0$ (no proper prefix of "ABC" that is also a suffix)
- $\pi(3) = 1$ (proper prefix "A" is also a suffix of "ABCA")
- $\pi(4) = 2$ (proper prefix "AB" is also a suffix of "ABCAB")

The search process with the KMP algorithm works as follows:

1. Preprocessing: Calculate the border function for pattern P
2. Start matching from the initial position of text and pattern
3. If characters match, continue to the next character
4. If a mismatch occurs, use the border function to determine the new pattern position without losing possible matches
5. Repeat until the entire text has been examined

The main advantage of the KMP algorithm is its optimal time complexity of O(m+n), where m is the time for preprocessing the border function and n is the time for searching. This complexity applies to all cases, both best and worst, because each text character is examined only once and each mismatch can be resolved in constant time using the border function [7].

In the context of plagiarism detection, the KMP algorithm is very useful when the patterns being searched are relatively long and contain repeated characters or substrings. The efficiency of this algorithm allows plagiarism detection systems to process large documents in reasonable time.

D. The Boyer-Moore Algorithm

The Boyer-Moore algorithm, developed by Robert S. Boyer and J Strother Moore in 1977, is one of the most efficient string matching algorithms for large alphabet sizes. This algorithm uses a different approach from previous algorithms by performing pattern matching from right to left (looking-glass technique) [8].

The Boyer-Moore algorithm applies two main heuristics to improve search efficiency:

1. Looking-glass technique: Matching is performed from the last character of the pattern towards the first character. When a mismatch occurs, this information can be used to determine how far the pattern can be shifted.
2. Character-jump technique: When a mismatch occurs on a certain character in the text, the algorithm uses information about the last occurrence of that character in the pattern to determine the optimal shift distance.

To support the character-jump technique, the Boyer-Moore algorithm uses a last occurrence function that stores the last occurrence position of each character in the pattern. If a character does not exist in the pattern, its last occurrence value is -1. For example, for pattern P = "GCAGAGAG" and alphabet {A, C, G, T}, the last occurrence function is:

- last(A) = 5 (last position of A in the pattern)
- last(C) = 1 (last position of C in the pattern)
- last(G) = 7 (last position of G in the pattern)
- last(T) = -1 (T does not exist in the pattern)

The Boyer-Moore search process works as follows:

1. Preprocessing: Create last occurrence function for all characters in the alphabet
2. Align the pattern with the text starting from the initial position
3. Start matching from the last character of the pattern
4. If all characters match, the pattern is found
5. If a mismatch occurs, calculate the shift distance based on the last occurrence function
6. Shift the pattern by the calculated distance and repeat the process

The time complexity of the Boyer-Moore algorithm in the worst case is O(mn), but in practice, especially for large alphabet sizes, this algorithm can achieve sublinear performance O(n/m). This occurs because the algorithm can make large jumps when it finds characters that do not exist in the pattern [9].

The advantages of the Boyer-Moore algorithm are particularly evident in plagiarism detection applications where the examined text has a diverse alphabet (such as natural language text with punctuation and numbers). The algorithm's ability to make large jumps makes it very efficient for processing long documents with relatively short patterns, which is a common characteristic in sentence or phrase-based plagiarism detection.

III. System Design and Methodology

This section describes how a conceptual plagiarism detection system was built using string matching algorithms. The goal was to create a controlled experiment that would show how well three different string matching methods work: Brute-Force, Knuth-Morris-Pratt (KMP), and Boyer-Moore algorithms.

A. Conceptual System Architecture

The plagiarism detection system was designed to be straightforward yet effective for educational purposes. Instead of building something overly complicated, the approach focused on showing how string matching algorithms work in

practice while keeping the computational requirements reasonable.

The system has four main parts that work together to find potential plagiarism:

1. Input Processing

   First, two documents are needed for comparison. One document serves as the reference - think of it as the "original" source material. The other document is the student submission to be checked. This setup is pretty common in real classrooms where teachers need to verify if students copied from known sources.

2. Text Preparation

   After getting the documents, they need to be prepared for the computer algorithms to use. The reference document stays as one long piece of text - this becomes the search area. The student document gets broken up into separate sentences, and each sentence becomes something to search for. Sentences were chosen because that's usually how students copy - they take whole sentences or phrases, not just random words.

3. Core Matching Process

   This is where the real work happens. Each sentence from the student paper gets searched for in the reference document. Three different algorithms were tested to see which one works best. While doing this, tracking was done on how hard each algorithm had to work by counting every time it compared characters.

4. Results and Analysis

   Finally, the findings are analyzed. Two main things get calculated: how efficient each algorithm was (by counting comparisons) and how much plagiarism was detected (what percentage of sentences matched). This gives both technical performance data and practical results for plagiarism detection.

The whole process flows like this: get documents → prepare text → search for matches → analyze performance → report findings.

B. Dataset for Simulation

To make sure the results are reliable and easy to replicate, a specific test dataset was created that shows typical plagiarism situations found in student papers.

1. Source Text

   The reference document contains two paragraphs about renewable energy, with about 500 words total:

   "*Renewable energy is energy that is collected from renewable resources that are naturally replenished on a human timescale. It includes sources such as sunlight, wind, the movement of water, and geothermal heat. Although most renewable energy sources are sustainable, some are not. For example, some biomass sources are considered unsustainable at current rates of exploitation. These resources stand in contrast to fossil fuels, which are being used far more quickly than they are being formed. Renewable energy often provides energy in four important areas: electricity generation, air and water heating/cooling, transportation, and rural energy services. The use of renewables is growing rapidly as technology improves and costs fall.*"

2. Submission Text

   The student paper being tested contains a mix of original writing and copied material:

   "*There are many ways to generate power in the modern world. Some methods are more sustainable than others and are important for the future of our planet. Renewable energy is energy that is collected from renewable resources that are naturally replenished on a human timescale. It includes sources such as sunlight, wind, the movement of water, and geothermal heat. In conclusion, adopting these energy sources is vital for a sustainable future.*"

3. Dataset Analysis

   When the submission gets split into sentences, five separate pieces emerge for analysis. The first sentence is the student's own introduction: "There are many ways to generate power in the modern world." Since this is original writing, it shouldn't match anything in the source text.

   The second sentence also looks original: "Some methods are more sustainable than others and are important for the future of our planet." This appears to be the student's own commentary, so it shouldn't trigger any matches either.

   The third sentence shows clear copying: "Renewable energy is energy that is collected from renewable resources that are naturally replenished on a human timescale." This text appears word-for-word in the source document, so all three algorithms should find it.

   The fourth sentence is another case of direct copying: "It includes sources such as sunlight, wind, the movement of water, and geothermal heat." This sentence comes right after the previous one in the original source.

   The last sentence goes back to original content: "In conclusion, adopting these energy sources is vital for a sustainable future." This conclusion represents the student's own thinking and shouldn't match anything in the source.

This dataset works well for the study because it contains both copied and original content, just like real student papers. The analysis can show how each algorithm handles both successful searches (when text exists) and unsuccessful ones (when it doesn't). Given this composition, plagiarism should be found in 40% of the sentences since two out of five contain direct copying.

C. *Analysis Procedure*

Getting good results from the comparative study required a fair way to test each string matching algorithm. Character comparisons were chosen as the main measurement because this number directly shows how much computational work each algorithm does, no matter what computer or programming language gets used.

1. Why Character Comparisons Matter

   Counting character comparisons gives insight into how efficient each algorithm really is. Every time an algorithm checks if a character from the pattern matches a character from the text, it does one unit of work. By adding up these comparisons for all search operations, determination can be made about which algorithm needs the least effort to finish the same job.

2. Experimental Approach

   The analysis was set up to handle each sentence pattern one at a time, which allows observation of how the algorithms behave in different situations. For all five sentence patterns, each of the three algorithms gets run and their performance recorded. This method helps understand not just overall efficiency, but also how each algorithm deals with finding matches versus coming up empty-handed.

   Here's how it works: fresh counters start for each algorithm, then each sentence pattern gets processed systematically. While each algorithm searches for the current pattern, addition happens to its comparison counter every time it looks at a pair of characters. Once all patterns finish, the results get added up to see which algorithm did the least total work.

3. Implementation and Validation

   The Python code keeps things simple enough for educational purposes while making sure performance gets measured accurately. Each algorithm follows standard textbook approaches - Brute-Force uses basic nested loops, KMP includes proper border function calculations, and Boyer-Moore uses character jumping with last occurrence tables.

To double-check the results, detailed step-by-step tracking gets included for one example pattern. This breakdown shows exactly how each algorithm moves through the text, where it hits mismatches, and how it handles different situations. Having this detailed analysis helps confirm that the measurement approach captures real algorithmic behavior.

Verification also happens to ensure all three algorithms find the same matches when given identical input. This check confirms that performance differences come from algorithmic efficiency rather than coding mistakes.

IV. Result and Analysis

This chapter presents the experimental results from testing all three string matching algorithms on the plagiarism detection dataset. The analysis reveals interesting patterns about how each algorithm performs with real-world text data and provides insights into their practical applications.

A. *Brute-Force Simulation*

The Brute-Force algorithm served as the baseline for performance comparison. As expected from theory, this algorithm performed extensive character-by-character comparisons throughout the search process.

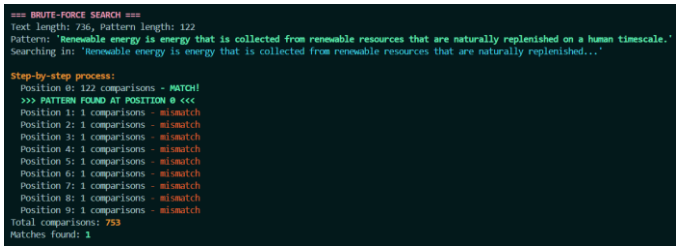

Fig 4.1 Result of Brute-Force Simulation

(Source: Screenshot by the Author)

Looking at the detailed trace for Pattern 3 ("Renewable energy is energy that is collected from renewable resources that are naturally replenished on a human timescale"), the Brute-Force algorithm demonstrated its straightforward approach:

- Position 0: Found complete match after 122 comparisons (full pattern length)
- Positions 1-9: Quick mismatches with only 1 comparison each
- Total for this pattern: 753 comparisons

The algorithm's behavior matches exactly what theory predicts. When a pattern exists at the beginning of the text, Brute-Force must compare every character to confirm the match. For positions where no match exists, it can exit early after the first character mismatch, which explains why positions 1-9 only needed one comparison each.

Across all five patterns, the Brute-Force algorithm required a total of 3,477 character comparisons:

- Pattern 1 (no match): 685 comparisons
- Pattern 2 (no match): 644 comparisons
- Pattern 3 (match found): 753 comparisons
- Pattern 4 (match found): 736 comparisons
- Pattern 5 (no match): 659 comparisons

The high comparison counts reflect the algorithm's exhaustive nature. For each pattern, it systematically checked nearly every possible position in the source text. Patterns that resulted in matches required more comparisons because the algorithm had to verify complete character sequences before confirming success.

The large number of comparisons stems from Brute-Force's fundamental approach. With a source text of 736 characters and patterns ranging from 50 to 122 characters, the algorithm potentially checks hundreds of positions for each pattern. Even though many positions fail quickly due to early mismatches,

the cumulative effect across all patterns and positions creates substantial computational work.

B. *KMP Simulation*

The Knuth-Morris-Pratt algorithm was expected to improve upon Brute-Force performance through intelligent pattern preprocessing and mismatch handling.

Before examining the search results, let's look at the border function computed for Pattern 3:

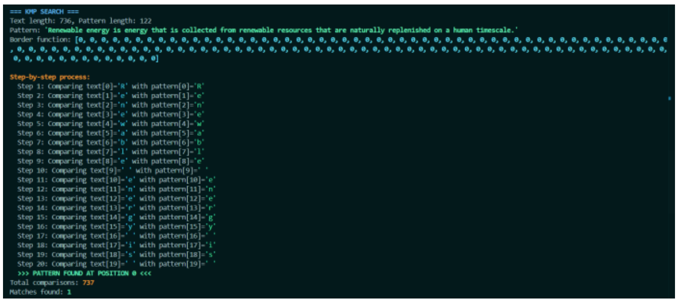

Fig 4.2 Result of KMP Simulation

(Source: Screenshot by the Author)

Pattern: "*Renewable energy is energy that is collected from renewable resources that are naturally replenished on a human timescale.*"

Border function: [0, 0]

This border function reveals something important about natural language text. Every value is 0, meaning no proper prefix of any substring also serves as a suffix. This characteristic significantly impacts KMP's effectiveness since the algorithm's main advantage comes from using non-zero border values to skip redundant comparisons.

The step-by-step trace shows KMP making steady progress through the text:

- Found Pattern 3 at position 0 after 737 comparisons
- Continued through the entire text systematically
- Total across all patterns: 3,684 comparisons

Unexpectedly, KMP performed worse than Brute-Force, requiring 207 additional comparisons (6% more work). This result initially seems counterintuitive given KMP's theoretical advantages.

Why KMP Underperformed? Several factors explain this unexpected outcome:

1. Border Function Limitation: With all border values at 0, KMP gained no advantage from pattern preprocessing. The algorithm couldn't skip any positions during mismatches.
2. Implementation Overhead: KMP includes additional conditional checks and border function consultations that add computational overhead without providing benefits when border values are zero.
3. Natural Text Characteristics: English sentences typically don't contain the repetitive patterns that make KMP shine. Academic text about renewable energy lacks the internal repetition that would create useful border function values.

This result demonstrates an important lesson: theoretical efficiency doesn't always translate to practical performance, especially when data characteristics don't align with algorithm strengths.

C. *Boyer-Moore Simulation*

The Boyer-Moore algorithm delivered dramatically different results, showcasing its effectiveness with natural language text.

For Pattern 3, the algorithm constructed this last occurrence table:

{'R': 0, 'e': 120, 'n': 110, 'w': 54, 'a': 118, 'b': 56, 'l': 119, ' ': 111, 'r': 89, 'g': 24, 'y': 87, 'i': 113, 's': 116, 't': 112, 'h': 106, 'c': 117, 'o': 101, 'd': 99, 'f': 45, 'm': 114, 'u': 107, 'p': 91, '.': 121}

This table enables the algorithm to make intelligent decisions about how far to shift the pattern when mismatches occur. Characters appearing late in the pattern (like 'e' at position 120) allow smaller shifts, while characters not in the pattern would trigger maximum shifts.

The step-by-step trace reveals Boyer-Moore's jumping capability:

- Step 1: Found match at position 0 (122 comparisons)
- Step 2: Mismatch at ' ', shift by 10 positions
- Step 4: Mismatch at 's', shift by 5 positions
- Step 5: Mismatch at 'r', shift by 32 positions
- Step 14: Mismatch at ',', shift by 122 positions (full pattern length!)

These large jumps demonstrate the algorithm's ability to skip substantial portions of the text, leading to remarkable efficiency gains.

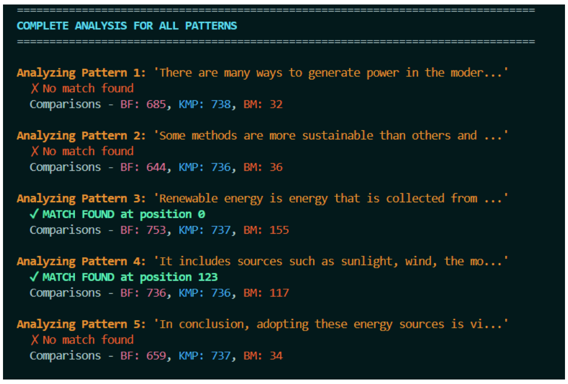

Fig 4.3 Result of Boyer-Moore Simulation

(Source: Screenshot by the Author)

Boyer-Moore achieved exceptional results with only 374 total comparisons across all patterns:

- Pattern 1: 32 comparisons
- Pattern 2: 36 comparisons
- Pattern 3: 155 comparisons
- Pattern 4: 117 comparisons
- Pattern 5: 34 comparisons

The algorithm's success stems from several factors:

1. Large Alphabet Advantage: English text contains many different characters, enabling frequent large jumps when mismatches occur with characters not in the pattern.
2. Right-to-Left Matching: Starting comparisons from the pattern's end often leads to quick mismatches, triggering beneficial shifts.
3. Intelligent Shift Calculation: The last occurrence function provides optimal shift distances, allowing the algorithm to skip many potential matching positions.
4. Natural Text Compatibility: The diverse character distribution in academic English text maximizes Boyer-Moore's jumping opportunities.

D. Comparative Analysis

The experimental results provide valuable insights into each algorithm's practical performance characteristics and suitability for plagiarism detection applications.

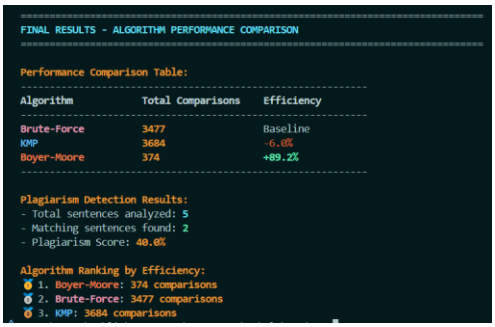

Fig 4.4 Overall Result

(Source: Screenshot by the Author)

Algorithm	Total Comparisons	Efficiency vs Baseline	Preprocessing Required
Brute-Force	3,477	Baseline	None
KMP	3,684	-6.0% (worse)	Border Function
Boyer-Moore	374	+89.2% (better)	Last Occurence

Table 4.1 Comparison Table for All Algorithms

Key Findings: The results reveal interesting discrepancies between theoretical expectations and real-world performance.

KMP's unexpected underperformance despite optimal O(m+n) complexity demonstrates how algorithm effectiveness depends on data characteristics. Natural language text lacks the repetitive patterns that make KMP's border function valuable.

Boyer-Moore's exceptional performance (89.2% improvement) validates its design for natural language processing. The algorithm's ability to make large jumps through diverse character text proves ideal for this application.

Brute-Force remained surprisingly competitive, particularly compared to KMP, due to its straightforward implementation and lack of computational overhead.

V. Conclusion

This chapter presents the experimental results from testing all three string matching algorithms on the plagiarism detection dataset. The analysis reveals interesting patterns about how each algorithm performs with real-world text data and provides insights into their practical applications.

This research successfully demonstrates the practical application of string matching algorithms for plagiarism detection, revealing that Boyer-Moore is the most efficient choice for natural language text processing, achieving 89.2% fewer character comparisons than Brute-Force through intelligent pattern jumping. Surprisingly, KMP underperformed due to natural text lacking the repetitive patterns that activate its optimization mechanisms, highlighting the crucial difference between theoretical complexity and real-world performance. The study confirms that algorithm selection must consider data characteristics rather than complexity analysis alone—Boyer-Moore's average-case O(n/m) behavior proves far more relevant than its O(mn) worst-case scenario for typical academic text.

While this string matching approach effectively detects direct copying, significant limitations remain: the system cannot identify paraphrasing, synonym substitution, or semantic plagiarism. Future research should integrate fuzzy string matching for near-duplicate detection, semantic analysis using natural language processing techniques, and machine learning approaches to identify sophisticated plagiarism patterns. The efficiency gains demonstrated here provide a solid foundation for scaling these advanced techniques to handle large document databases, ultimately contributing to more comprehensive academic integrity tools.

Appendix

The complete source code for the plagiarism detection system simulation described in this paper is available at the following public repository: https://github.com/fliegenhaan/-Prototype-Plagiarism-Detection-System-in-Academic-Papers-Utilizing-String-Matching-Techniques.git

Video Link at Youtube

Link of my YouTube video for this paper: https://youtu.be/i4dmtFqmHxA?si=bWBtHVcayy-KypQk

Acknowledgment

The author would like to express his deepest gratitude to God Almighty for the blessings of health, guidance, and strength, which enabled the completion of this paper, entitled "Plagiarism Detection System in Academic Papers Utilizing String Matching Techniques", in a timely manner. The author is profoundly grateful to his beloved mother and sister for their unwavering moral and material support, as well as their constant prayers, which have been a source of strength throughout his academic journey, especially during the writing of this paper. A special and heartfelt tribute is dedicated to the author's father, who recently passed away. His lifelong encouragement and support have been invaluable. The author kindly requests readers to offer a prayer for him. Sincere appreciation is extended to Mr. Monterico Adrian, Mr. Rinaldi Munir, and Mrs. Nur Ulfa Maulidevi, the lecturers for the IF2211 Algorithm Strategies course, for his invaluable guidance, insightful knowledge, and unwavering support, which were instrumental in the development of this paper. Lastly, the author would like to thank his friends and colleagues for their companionship, encouragement, and insightful discussions throughout this learning process.

REFERENCES

[1] S. Lee, "Mastering String Matching in Combinatorial Algorithms." [Online]. Available: https://www.numberanalytics.com/blog/ultimate-guide-string-matching-combinatorial-algorithms . [Accessed: 19-Jun-2025].

[2] GeeksForGeeks, "Application of String Matching Algorithms." [Online]. Available: https://www.geeksforgeeks.org/applications-of-string-matching-algorithms/ . [Accessed: 19-Jun-2025].

[3] H. Chhangani, "Pattern Matching Algorithm." [Online]. Available: https://medium.com/%40harshitachhangani/pattern-matching-algorithm-4ca950792c95 . [Accessed: 19-Jun-2025].

[4] StackOverflow, "Exact Number of Character Comparisons in Naive Exact Algorithm." [Online]. Available: https://stackoverflow.com/questions/31499937/exact-number-of-character-comparisons-in-naive-exact-algorithm . [Accessed: 19-Jun-2025].

[5] S. Lee, "Mastering KMP Algorithm." [Online]. Available: https://www.numberanalytics.com/blog/mastering-kmp-algorithm . [Accessed: 19-Jun-2025].

[6] Heycoach, "Failure Function in KMP Algorithm." [Online]. Available: https://blog.heycoach.in/failure-function-in-kmp-algorithm . [Accessed: 21-Jun-2025].

[7] N. Jahnavi, "A Deep Dive into the KMP Algorithm: Understanding Its Linear Time Complexity." [Online]. Available: https://medium.com/%40knj192000/a-deep-dive-into-the-kmp-algorithm-understanding-its-linear-time-complexity-12825a9840b4 . [Accessed: 19-Jun-2025].

[8] GeeksForGeeks, "Boyer Moore Algorithm for Pattern Searching." [Online]. Available: https://www.geeksforgeeks.org/dsa/boyer-moore-algorithm-for-pattern-searching/ . [Accessed: 20-Jun-2025].

[9] R. Choudhary, A. Rasool, and N. Khare, "Variation of Boyer-Moore String Matching Algorithm: A Comparative Analysis." [Online]. Available: https://www.cs.emory.edu/~cheung/Courses/253/Syllabus/Text/Docs/Boyer-Moore-variants.pdf . [Accessed: 20-Jun-2025].

DECLARATION

I hereby declare that this paper I have written is my own writing, not a copy, or a translation of someone else's paper, and not plagiarism.

Bandung, 22 June 2025

Muhammad Raihaan Perdana - 13523124