Paper: Comparison between Brute-force, Sunday, KMP, FSM, Rabin-Karp, and Gusfield Z

Hassan Samine

1 Introduction

Pattern matching algorithms are a crucial tool in computer science that allow us to efficiently search for patterns within a larger text or string. There are numerous pattern matching algorithms available, each with their unique strengths and weaknesses. For this paper, we will be comparing the running time of six popular pattern matching algorithms: Brute-force, Sunday, KMP, FSM, Rabin-Karp, and Gusfield Z. Through this research, we hope to improve our understanding of these fundamental algorithms and their practical applications.

1.1. Brute-force algorithm:

The brute-force pattern matching algorithm, also known as the naive algorithm, is a simple string searching algorithm that checks for a pattern by comparing it to all possible substrings of a given text. The algorithm works by sliding the pattern over the text, checking if the pattern matches the substring at each position until it either finds a match or exhausts all possibilities.

The code that we implemented prompts the user to input two strings, "text" and "pattern", and then searches for the pattern within the text by comparing each substring of the text with the pattern. If a match is found, the program outputs the starting index of the match.

The search function takes in two parameters, the text, and the pattern to be searched. The function uses two nested for-loops to compare each character of the pattern with the characters of each substring of the text.

1.2. Knuth Morris Patterson (KMP) Algorithm:

The Knuth-Morris-Pratt (KMP) algorithm is a more efficient string-matching algorithm than the brute-force algorithm, also known as the naive algorithm. Like the brute-force algorithm, the KMP algorithm searches for a pattern within a given text by comparing the pattern to all possible substrings of the text. However, the KMP algorithm utilizes an additional pre-processing step to avoid comparing the pattern with all possible substrings of the text.

The KMP algorithm works by pre-processing the pattern and generating a partial match table, which is also known as the longest proper prefix-suffix table (LPS table). The LPS table is then used to guide the search process and avoid unnecessary comparisons of the pattern with the text.

In this implementation, the KMP algorithm is used to search for a pattern within a given text. The code prompts the user to input the text and the pattern to be searched, and then calls the KMP function with these strings as input. The KMP function searches for the pattern within the text using the LPS table to avoid unnecessary comparisons, and outputs the starting index of any matches found.

1.3. Sunday Algorithm:

The Sunday matching algorithm is a string-matching algorithm that is similar to the brute-force algorithm, which compares a pattern to all possible substrings of the text. However, it utilizes an additional pre-processing step to generate a shift table that maps each character of the pattern to the number of characters by which the pattern can be shifted in case of a mismatch.

The shift table is used to guide the search process and avoid unnecessary comparisons of the pattern with the text. The algorithm compares the pattern with the text by sliding the pattern over the text, starting at the beginning of the text. If there is a mismatch between a character of the pattern and the corresponding character in the text, the algorithm uses the shift table to determine the number of characters by which the pattern can be shifted to the right.

In this implementation, the code prompts the user to input the text and the pattern to be searched, and then calls the Sunday matching function with these strings as input. The Sunday matching function searches for the pattern within the text using the shift table to avoid unnecessary comparisons, and outputs the starting index of any matches found. If no match is found, the function returns -1.

1.4. Finite State Machine Algorithm:

FSM (Finite State Machine) Algorithm: It is a simple string-matching algorithm that works by creating a state machine to represent the pattern to be matched. It then scans the text from left to right, moving from state to state based on the characters encountered in the text. If the machine reaches the final state, it means the pattern has been found. The FSM algorithm has a time complexity of O(m*|Σ|) for preprocessing the pattern and O(n) for matching the pattern in the text, where m is the length of the pattern, |Σ| is the size of the alphabet, and n is the length of the text.

1.5. Rabin-Karp Algorithm:

It is a hash-based string-matching algorithm that works by computing the hash values of the pattern and all substrings of the text of the same length as the pattern. It then compares the hash values to determine if there is a match. This algorithm has a time complexity of O(n+m) in the average and best case, and O(nm) in the worst case, where n is the length of the text and m is the length of the pattern.

1.6. Gusfield Z Algorithm:

It is a linear time string matching algorithm that works by computing the Z-values of a concatenated string formed by concatenating the pattern, a separator symbol, and the text. The Z-values represent the length of the longest substring that starts from a given position in the concatenated string and matches a prefix of the pattern. The algorithm then looks for positions in the Z-array where the Z-value is equal to the length of the pattern, which indicates a match. The Gusfield Z algorithm has a time complexity of O(n+m) for matching the pattern in the text, where n is the length of the text and m is the length of the pattern. The preprocessing time is O(m) which is the time to compute the Z-values of the concatenated string.

2 Methodology

The algorithms were implemented in C++. We have used essays of varying lengths ranging from 100 to 700 words as the initial text and tested the algorithms using three types of patterns: a single word, a sentence, and a large pattern. To ensure accurate and reliable results, each algorithm was tested 1000 times, and the reported time is the average of these tests. This rigorous testing methodology was employed to reduce the impact of potential outliers or random variations in

the test results, thus ensuring that the measured performance accurately reflects the actual performance of the algorithms.

3 Results

In Figure 1, we can observe the comparison of running times for all six pattern matching algorithms when tested with a text ranging from 100 to 700 words and a single-word pattern. Upon initial inspection, it is evident that the Brute-Force algorithm performs the fastest out of the six algorithms, with a time ranging between approximately 10 Ms and 80 Ms. In contrast, the Gusfield-Z algorithm can be deemed the slowest algorithm to perform the pattern matching task, with a time range of approximately 50 Ms to 180 Ms.

However, a closer analysis of the graph reveals that all algorithms recorded a time between 10 Ms and 60 Ms for the first 200 words. Among these algorithms, Brute-Force is the fastest, taking only 22.5 Ms, while Gusfield-Z is the slowest, taking 48.896 Ms. At the 500-word mark, we observe that the FSM algorithm is the fastest, recording a time of 50.60 Ms, whereas the slowest is again the Gusfield-Z algorithm with a time of 173.46 Ms. By the end mark of 700 words, there is a change, with the Sunday algorithm becoming the slowest algorithm, recording a time of 143.23 Ms, while Brute-Force is the fastest with a recorded time of 81 Ms.

In figure 2, we repeated the same comparison but this time the length of the pattern is a sentence instead of a word, we notice that for the majority of the algorithms the RT is between 0 and 200 MS except FSM algorithm which performed in a strange behavior ranging between 200 and 500 MS with its best performance at the 500 word text

In figure 3 and 4, we conducted the same experience as before just changing the pattern length from a sentence to a paragraph, we made 2 different graphs due to the unexpected behavior of the FSM algorithm and how slow it is compared the other 5 algorithms with its best time around 1800 MS and worst around 6600 MS compared to the fastest algorithm which is Brute force algorithm with a best time of 10 MS and worst 80 MS

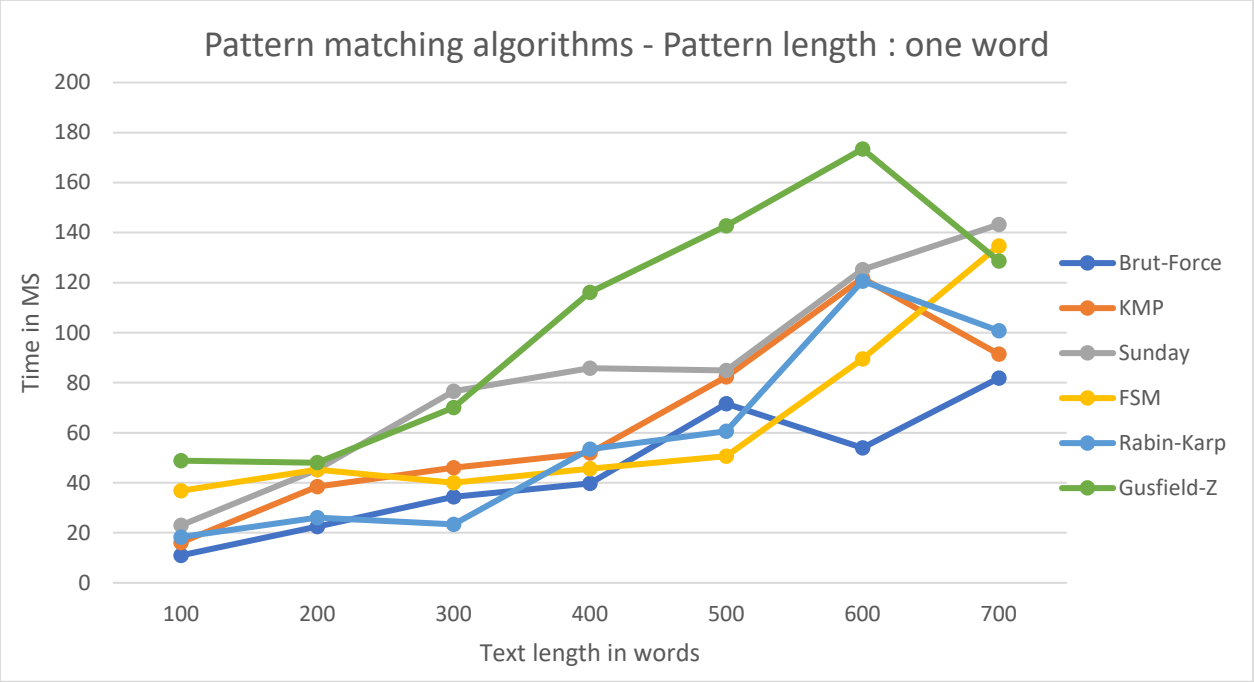

Figure 1: average time each algorithm took to find a pattern of one word.

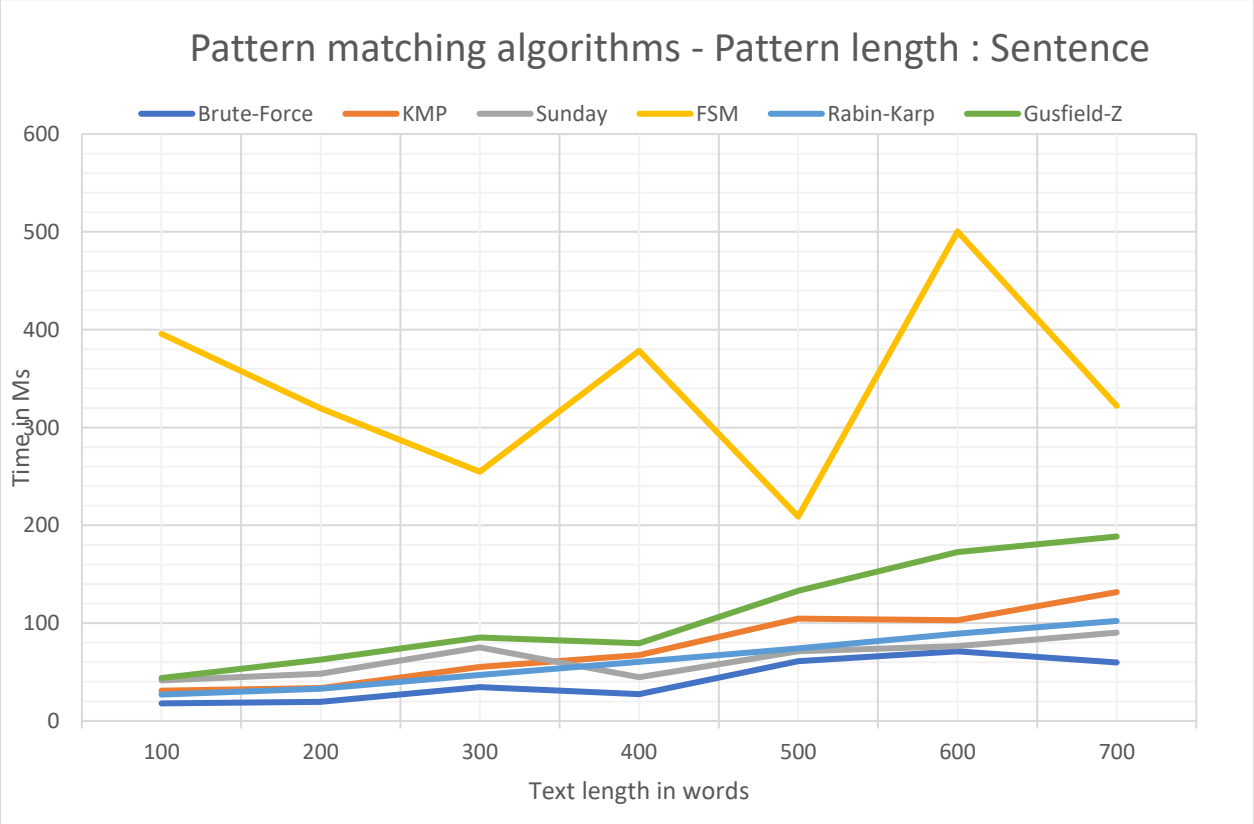

Figure 2: average time each algorithm took to find a pattern of a sentence.

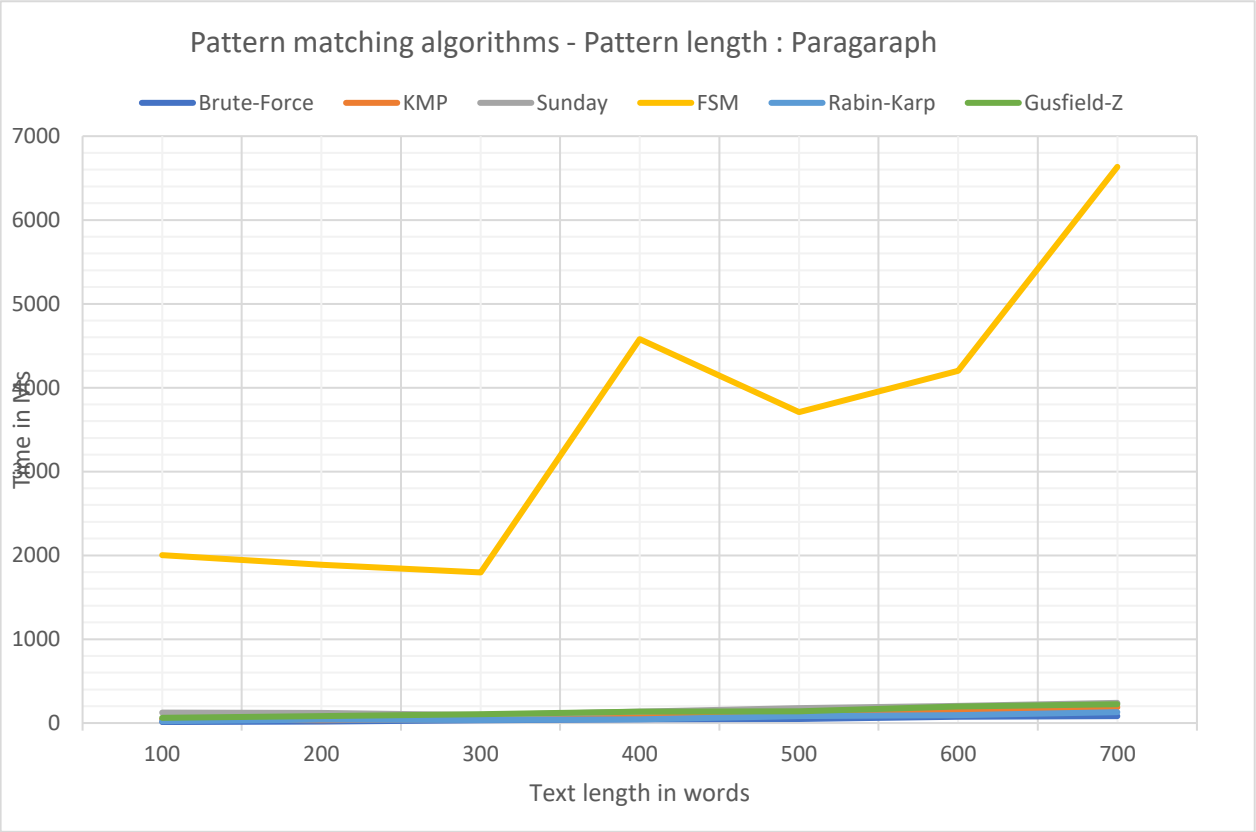

Figure 3: average time each algorithm took to find a pattern of a paragraph.

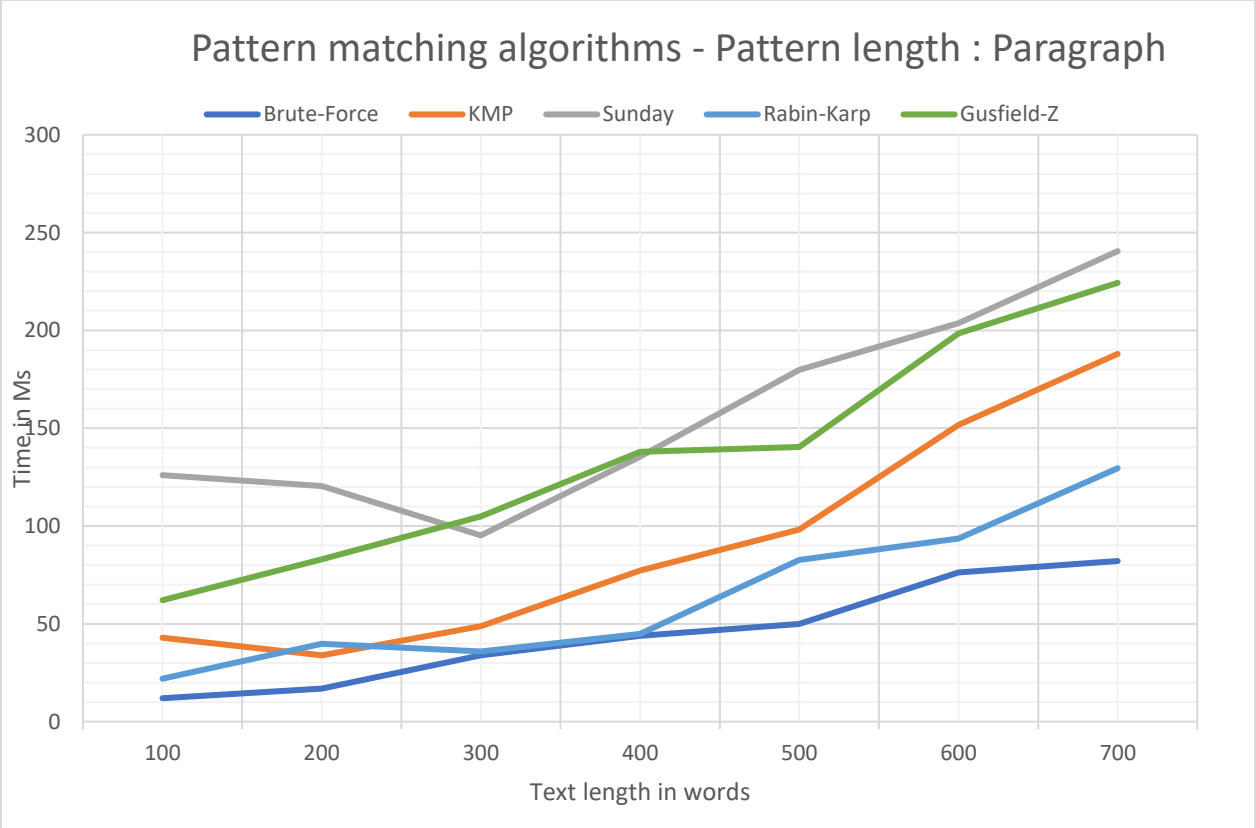

Figure 4:average time each algorithm took to find a pattern of a sentence (without FSM)

4 Conclusion

We can safely conclude that for a text ranging of 100 to 700 word and a pattern ranging from a word to a paragraph, the Brute force algorithm is the fastest by far with a bit of competition from Robin-Karp algorithm in some cases specially in the 300 to 400 word texts, beside that we got a very unexpected behavior from FSM algorithm which can be caused by implementation issues