International Journal for Electronic Crime Investigation

ISSN: 2522-3429 (Print)
ISSN: 2616-6003 (Online)

DOI: https://doi.org/10.54692/ijeci.2025.0901/251

Research Article

Vol. 9 issue 1 Jan-Jun 2025

A Tri-Character guided exact String-matching Algorithm for Efficient str detection In Forensic DNA Analysis

[1] Syed Faizan Ali Shah, [2]Amna Asif Lodhi, [3]Khawar Maqsood

[13]Mohi ud din Islamic University Nerian Sharif AJ&K, Trarkhal, Pakistan,

[2]Riphah International University Sahiwal, Pakistan,

Corresponding Author: faizan.ali@miu.edu.pk

Received: June 17,2025; **Accepted:** June 29,2025; **Published:** June 30,2025

ABSTRACT

The importance of string-matching algorithms in the world of modern DNA forensic technology cannot be over-stated. Short Tandem Repeats (STRs) play an important role in forensic DNA analysis due to their high variability among individuals. Fast and accurate detection of STRs in large-scale genomic data is most important for criminal investigations, identity verification, and population studies. This study introduces a novel exact string-matching algorithm, the Tri-Scan for Left, Right, and Middle Character (TSLRMC) approach, tailored for efficient pattern detection in forensic DNA sequences. This research addresses limitations found in some famous and widely used exact string-matching algorithms. Proposed algorithm improves the running time for scanning pattern string in a long text string. The novelty of the proposed algorithm is to optimize the scanning by scanning the pattern string left, right and middle characters in the long DNA sequence string and then scanning the remaining characters of the pattern string in that partial text window where the pattern string's left, right and middle characters are found. The proposed algorithm shows significant improvement compared with the most popular exact string-matching algorithms, based on running time as well as number of characters compared. Time complexity of this proposed novel TSLRMC algorithm is O (n-m) in worst case, O (mn) in average case and O (1) in best case.

Keywords: TSLMC (Text scan for left right and middle character), T(Text), P(Pattern), STR (Short Tandem Repeat)

1. INTRODUCTION

DNA forensics depends a lot on being able to correctly identify Short Tandem Repeats (STRs), which are specific repeating patterns of 2 to 6 base pairs in genomic DNA. Because these STRs are highly polymorphic, they are useful for identifying people, especially in criminal investigations and legal cases. As genomic data grows and the need for quick analysis grows, it is becoming more and more important to have efficient algorithms for finding patterns in DNA sequences. Traditional sequence alignment tools like BLAST and BWA, although accurate, often suffer from computational inefficiency when processing huge genomic datasets. In this context, exact string matching algorithms play a vital role by enabling rapid and precise matching of nucleotide sequences. Their application in locating and comparing STR regions within vast DNA sequences is crucial for minimizing time and computational resources in forensic workflows. Therefore, developing optimized string matching algorithms tailored for DNA forensic applications — like the proposed Tri-Character Based Matching Algorithm — is of critical importance. Collected data often consists of letters and numbers presented in a format accessible to human readers. When we talk about the string of characters we mostly think as lines of English letters that humans can understand and read. Computers store alphanumeric characters as numerical values, not as human-readable text. In most programming languages, such as C and Java, character data types are internally treated as numeric values. Identifying a specific pattern within a lengthy text string, potentially comprising billions of characters, is a complex task for the human brain. Therefore, specialized computer algorithms are developed to perform such operations with high speed and precision.

The importance of string matching algorithms to the world of modern technology cannot be overstated. Like molecular biology, text processing, web search, image processing, and network intrusion detection [1]. Some other important applications are the categorization of diseases, survival rate prediction for a patient who has specific diseases, verification of fingerprints, detection of a face, iris discrimination, chromosomes shape discrimination, optical character recognition[2]. String matching algorithm is the most studied subject in the wider category of text processing[3]. Normally, pattern or string matching algorithms are categorized into two broader types, approximate and exact string matching algorithm [4].

Exact string matching algorithm problem generally formalized as follows.

$\sum$ = Finite set of alphabet

T= String of text from $\sum$ where |T|=n

P= Pattern string derived from $\sum$ where |P|=m

There are many applications of exact string matching like molecular biology, text processing, web search, image processing, and network intrusion detection.

The primary objective of the string searching algorithm is to find existences of P (pattern string) in T (text string).length of pattern is m generally known as pattern *P*, where n length of text represented as *T*[5]. Below is fig show overview of exact string matching algorithm's hierarchy

The simplest algorithm used for pattern matching or we can say string searching is a brute force algorithm which is also called the Naive algorithm [6].this algorithm is the simplest one and the working idea is it compare string and pattern with shifting sliding window on character and compare pattern again. The brute force algorithm has *O (nm)* time complexity in all cases (best case, worst case, average case).

Knuth Morris Pratt algorithm developed by D.E. Knuth, with J.H. Morris and V.R. Pratt. This algorithm is also scanning pattern in a string from left to right, but its importance and improvement is shifting of text sliding window.it shifts the text sliding window based on the previous track of comparison if a mismatch occurred. This algorithm works in two phases one is preprocessing and final and the second is searching [7]. The time complexity for the preprocessing phase is *O (m)* and the searching phase is *O (nm)*

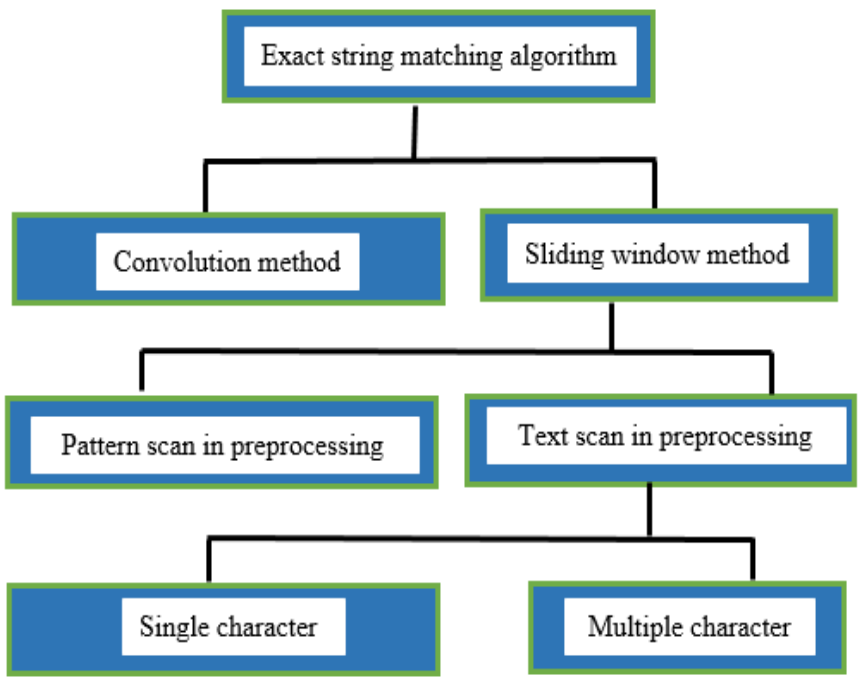

Figure 1: General description of exact string matching algorithm

Boyer Moore algorithm is best and taken as a base for all researchers[8]. Single pattern algorithms are the best tackle with Boyer Moore. This algorithm works a little differently from other algorithms discussed above. This algorithm starts scanning from right to left for pattern scanning. If a mismatch occurs BM builds two heuristics one is a bad character and the second is called good suffix heuristic. Preprocessing phase time and space complexity is O (m+|∑|) worst and best cases for searching is O (nm) and O (n/m) respectively

The Quick Search algorithm is a simplified variant of the Boyer-Moore algorithm that focuses solely on the bad character heuristic to achieve efficient pattern matching. Quicksort algorithm also has two phases preprocessing and scanning. Time complexity for the reprocessing phase is O (m+ σ). It's an important factor is the quadratic worst case for the searching phase.

Robin karp algorithm is another well-known string matching algorithm. Its key factor is using of hash function to search pattern string (m) in long text string (n) if hash value of both string are same then it compare again and if hash value is not matched then pattern string will shift to right which will affect the performance of Robin karp algorithm[9]. Computing of hash value for pattern and long text string is one of the main drawbacks of this algorithm.

2. RELATED WORK

Now a days string matching algorithms are one of the most studied and important field in computer science[10]. Recent advances in DNA sequencing have necessitated the development of highly efficient string matching algorithms tailored for forensic applications. Traditional sequence alignment tools like BLAST and BWA are effective but computationally intensive when applied to large-scale forensic datasets,[11] [12].The Brute force algorithm is also called Naive algorithm [6] and is the simplest algorithm for exact string matching. All positions in a text from 0 to *nm* are parsed. Cycle through all the chains whether they find a pattern or not. The time complexity of the brute force algorithm is O (mn) in the worst case and is linear in practice [13]. KMP exact string matching algorithm is derived somehow from brute force algorithm. Important factor is that it keeps track of previous matches found in string against pattern until mismatch occur because shifts of pattern on sliding text window depends on these previous matches[13].Time complexity for preprocessing phase is *O (m)* and *O (nm)* for searching phase.BM algorithm improve two factors, time taken for execution and number of shifts taken in whole process of string matching very efficiently. This algorithm ranked as most efficient one in this field of exact pattern matching[14], [15] Main factor of this algorithm is that it start from right of the pattern string to left and generate bad character heuristic. Rabin-Karp algorithm uses hash function to find pattern[9] .Its best and efficient applications are plagiarism detection, segment concerning DNA chain. The complexity of in worst case is *O ((n-m + 1) m)* to *O (nm + 1)*[16].

Berry Ravindran algorithm performs

shifts by considering the bad-character shift for two successive characters to the right of the partial text window. The analysis phase of Berry Ravindran's algorithm takes O (nm) time. Therefore, O (nm) is the worst execution time of the BR algorithm. $O\ (n / m + 2)$ is the best execution time of the BR algorithm. The exact string matching problem is one of the most studied problems in computer science [11].

Raita developed an exact pattern matching algorithm named as Raita [17]. Raita algorithm first compares the last pattern's character, then the first and finally the middle character with the selected text window. If three characters are matched, then start comparing the other characters of the pattern. Raita algorithm performs the shifts like the BM Horspool algorithm. Raita has the same preprocessing phase as Boyer-Moore Horspool Bad character function and takes $O(m+|\Sigma|)$ time before searching. $O\ (|\Sigma|)$ extra space is required to compute preprocessing phase as Horpool BM algorithm. The principle factor of BM algorithms to traverse pattern character in long text string from left to right.[14] Pattern matching algorithms means to pores raw data[15]. Today technological development creates huge bulk of raw data which needs to be processed and fetch information from them. Traditional algorithms like Naive and KMP are simple but can be inefficient or require extra memory. Boyer-Moore and Horspool offer better performance using heuristic shifts but perform poorly with small alphabets like DNA. Rabin-Karp is good for multiple patterns but suffers from hash collisions. The proposed TSLRMC algorithm reduces unnecessary comparisons by scanning key characters, making it faster and more efficient for exact DNA pattern matching.

3. PROPOSED SOLUTION

3.1. Aims And Objectives

The goal of this research work is to study existing exact string-matching algorithms and subsequently to design an efficient exact string-matching algorithm to improve data searching. Many exact string-matching algorithms focus on the number of character counts and running time to find pattern string in long text string. Keeping these factors in mind the specific objective of the study is to propose an algorithm that reduces time and the number of character counts. Some limitations are found in existing string-matching algorithm like most basic Not So Naïve, Information gained about text for one value of shift is entirely ignored in considering other value. And in KMP algorithm Time wasted because of prefix function. Mostly used and well-known algorithms, Boyer-Moore and BM Horspool are less effective when applied to binary strings and short pattern lengths due to limited shift opportunities. Rabin Karp takes more time to compare pattern string in long text string.

The proposed algorithm will be used to obtain comprehensive simulation results for comparison of its computational performance with existing exact string matching algorithms. Proposed algorithm give improved running time but somehow number of shifts taken for sliding text

window are not so good of long pattern string. This limitation could be overcome in future work.

3.2. Data sets

In this study, sequence of DNA is used to test and evaluate the results of our proposed algorithm (TSLRM) as well as to evaluate performance compared with other exact string matching algorithms. The research presented in this dissertation is based on the experimental approach for understanding exact string matching algorithms and its working pattern. The static file containing billions of DNA sequence string used for testing of algorithm and on the same file other previously developed algorithms also test on the same file as data set. This file is prepared by generating paragraphs from the NCBI along with synthetically generated nucleotide sequences. Pattern string from this text statically collected with varying length of 6, 12,18,24,30 and 45.

4. PROPOSED TEXT SCAN FOR LEFT RIGHT AND MIDDLE CHARACTER ALGORITHM (TSLRMC)

Text Scan for pattern string First, last and middle Characters exact pattern matching algorithm compares a given pattern from both sides simultaneously. It did not require the whole pattern to be searched if a mismatch occurs. In case if pattern string left character or left and rightmost character of pattern string does not match then the whole pattern could not need to be scanned. If both characters are found, then look for the middle character at the same time. If it matches in text string, then compare the remaining pattern string character within the selected partial text window. If a mismatch occurs, then shift the pattern to one index forward.

TSPLFMC has two cases when scanning the pattern last, first and middle characters in the text string in preprocessing phase or scanning phase. Below figure explains general work. This algorithm finds pattern left and right character then selects that string as a partial text window which is also known as a sliding text window. This algorithm also starts from left to right.

Like other algorithms the proposed algorithm also start working from left to right. First step is to scan left character of pattern P [0, 1 . . . M-1] in text string T [0, 1. . . . N-1]. If the character is found in the string, then an important step that improves the efficiency of the proposed algorithm is search pattern M-1 character in text string at (m-1) from the character where the pattern left character found if right character not found then left found character shift one step next in a text string. if the string has this character and this character must be the same distance from found left character of pattern in a string as the length of the pattern then move sliding text window where left character found and right charter at pattern length. The below figure shows this process.

These cases are occurred in scanning of pattern string in text string. If first P[0] is not found in the T[i+1…n-1.] if P[0] found at T[i+1…n-1] but P[m-1] not found at T[i’’], and

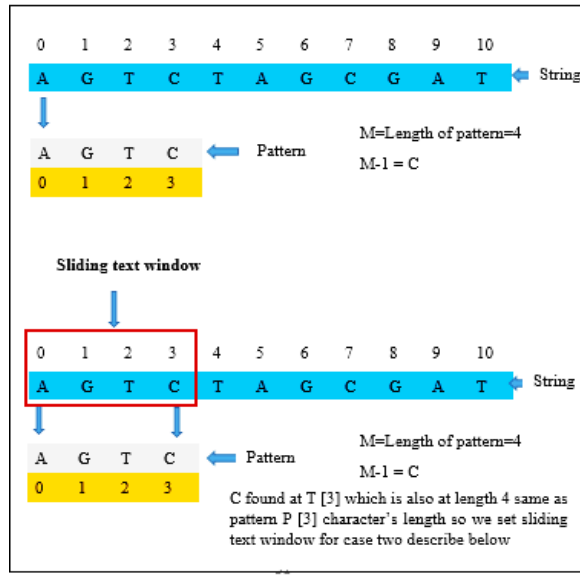

Figure 2: general working of sampling based algorithm

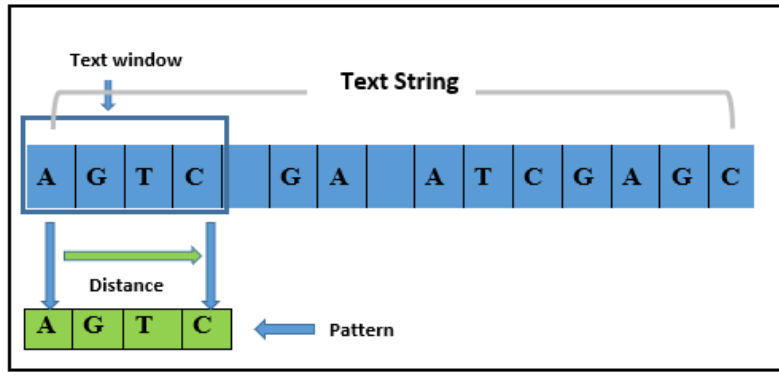

Figure 3 selection of sliding text window

P[0] found at T[i'] and P[m-1] = T[i''] but P[m/2] ≠ T[i'''']. In this case, TSPLRMC continues scanning for the appropriate text window. If fail to find an appropriate window then takes a maximum shift and not only the scanning in the T[i+1…n-1] but also searching of P in T is also completed. Scanning phase search T[7…24] and did not find P[6] at any location as a result searching of pattern P in the Text T is completed and TSPLRMC is

exited.

The 2nd possibility in Figure 3.2; describes that when a mismatch occurs between P[4] and T[4] then the scanning phase called. Scanning phase found P[6] at T[13] but P[0] not found at T[7]. TSPLFMC continues scanning the text string until a combination of P[m-1], P[0]and P[m/2] found at T[i'], T[i''] and T[i'''] or scanning text string for pattern last, first and middle characters completed.

Third possibility when describe in Figure 5.2; that *P[6] = T[13]* and *P[0] = T[7]* but *P[3] ≠ T[10]* in this cases it continues scanning until a text window is found who's last, first and middle character are equal to pattern first, last and middle characters.

The new text window in the text string is only selected when *P[m-1] = T[i']*, *P[0]=T[i'']* *and* *P[m/2] = T[i''']*. Preprocessing (Scanning) phase looks for *P[0]* at *T[i'']* only when *P[m-1] = T[i']* and when *P[0] = T[i']* and *P[m-1] = T[i'']* then look for *P[m/2]* at *T[i''']* otherwise continue scanning text for *P[m-1]*.

4.1. TSLRMC Algorithm Pseudo

Below complete pseudo code of proposed algorithm including finding of partial text window and searching of pattern string in selected PTW is described.

```
SearchingPat (T, P)
N  length (T)
M  length (P)
For i  ← 0 to length (N-M+1)
    If T[i]==P[0] and T[i+M-1]==P[M-1] and T[i+M/2]==P[M/2]
        k=1
        j=1
        t=1
        While j < t +M-1:
            If T[j] ≠ p[k]
                Exit from loop
            J+=1
            k+=1
            m+=1
    If m = M
```

The static file containing billions of English words used for testing of algorithm and on the same file other previously developed algorithms also test on the same file as data set. Text analysis of the characters on the left, right and middle of a real string algorithm compares the string to a given pattern simultaneously on each page. It was not necessary to examine the general trend of the imbalance. In case if pattern string left character or left and rightmost character of pattern string does not match then the whole pattern has no need to be scanned. If both characters are found, then look for the middle character at the same time. If it is matched in text string, then compare the remaining pattern string character within the selected partial text window. If a mismatch occurs, then shift the pattern to one index forward.

5. EXPERIMENTAL ANALYSIS

Like other algorithms the proposed algorithm also starts working from left to right. First step is to scan the left character of the pattern P [0, 1 . . . M-1] in long text string T [0, 1. . . . N-1]. If the character found in the string, then an important step that improves the efficiency of the proposed algorithm is

to search for pattern M-1 character in text string at (m-1) from the character where pattern left character found in text string. If the right character does not find, then left found character shift one index next in that text string. If the string has this character and this character must be the same distance from found left character of pattern in a string as the length of the pattern, then move sliding text window where left and right charter of pattern find these characters. Then at the same time find middle character. If these conditions matched the proposed algorithm traversed in that partial text window to find complete sequence.

TSLRMC exact string matching algorithm takes remarkably less time to compare pattern string P in long text string. Below table show statistical calculation of proposed TSLRM algorithm and some important and well known algorithms used as base to the field of exact string matching algorithm.

A text string of length ten million (10,000,000) characters is selected for the experiment of different exact pattern matching algorithms. Same text file tested on different length of pattern like 6, 12,18,24,30 and 45. The experiments calculate the total time or simply running time to find characters all the existences of pattern *P* in text *T*.

Table 1: Running time base comparison of TSLRMC algorithm with other exact string matching algorithms for different length of pattern string

Pattern Size / Algorithm	6	12	18	24	30	45
Notsonaiv	28.544522	21.890326	21.945335	21.854243	22.18419218	22.0491895
Naive	10.282784	8.1266021	8.2954790	8.1266021	7.97943258	7.8651649
BM	12.015146	6.436575	7.7103913	5.739215	6.3917074	6.3076739
Horspool	11.022146	6.436575	7.7102584	5.737415	6.3912856	6.3011258
Robin Karp	24.983665	20.145691633	20.3674415	20.1336185	19.3406360	18.8220953
KMP	9.397804	7.342433	7.35949110	7.2282016	7.0740635	6.63744945
TSPLRMC	6.213896	4.3378226	4.343657	4.9047944	5.420764	3.9443511

This table shows clearly proposed algorithm improves running time effectively. Below chart represent graphical calculation and describes different behaviors for different length of pattern strings.

This chapter presents experiments and results of the research work and TSLRMC exact string matching algorithms with existing pattern matching algorithms as Naivee, Not So Naive, BM, BM Horspool, KMP and BM. Text Scan for

Pattern left, right and middle characters is compared using character compared base and running in seconds. Comparisons with existing algorithms in different sections and the results are shown by using tables and graphs. Clearly above results show studied novel algorithms improved the results than existing algorithms and these are the algorithms taken as base for these research area and BM is considered as most used and important algorithm.

Average running time for TSLRMC is 4.860881 whereas for Notsonaive, Naive, Boyer Moore, BM Horspool, Robin Karp and KMP is 23.07797, 8.446011, 7.433451, 7.266468, 21.43852, 7.506574.

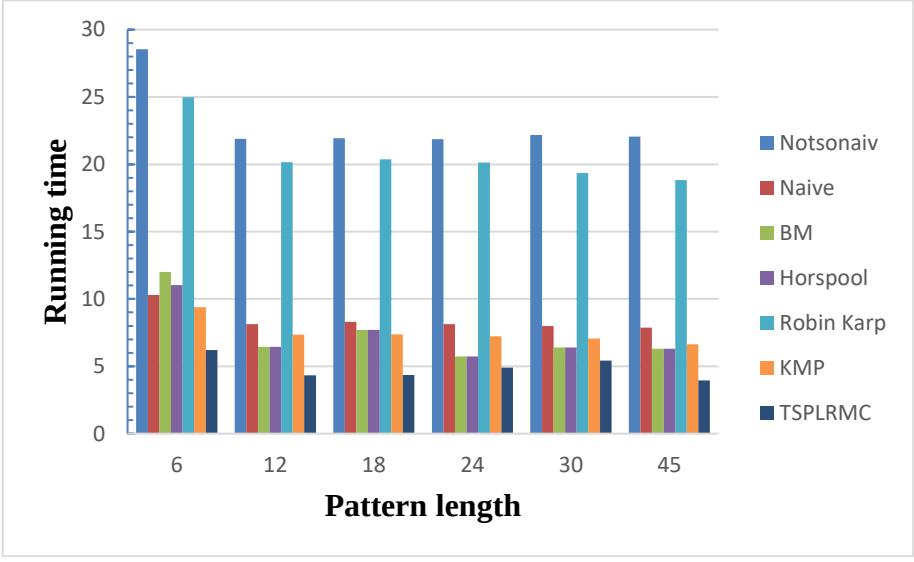

Figure 4: Running time base comparison of TSLRMC algorithm with some of the most well-known exact string matching algorithm for different length of pattern string

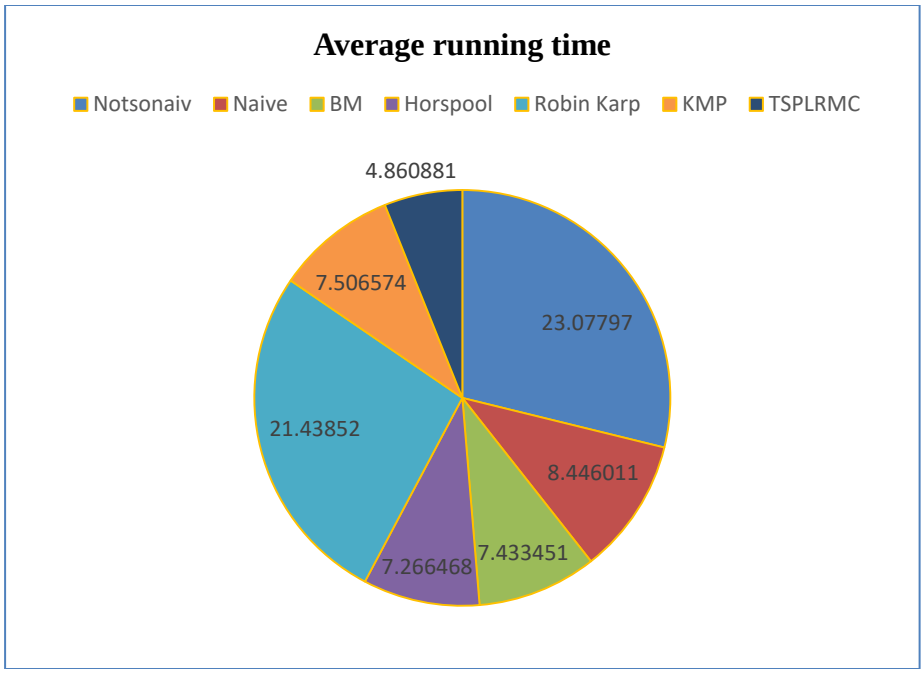

Figure 5: Average running time of proposed TSPLRMC algorithm compared with other exact string matching algorithms

As in Figure 4 shows that TSLRMC exact string matching algorithm takes less time to compare pattern string in long text string than the all other discussed algorithm.

6. CONCLUSION

The proposed TSLRMC algorithm holds strong potential for real-world applications, particularly in forensic STR profiling, where rapid and accurate identification of short tandem repeats in DNA sequences is crucial for criminal investigations, paternity testing, and identity verification. Its efficient pattern-matching approach makes it well-suited for integration into forensic DNA analysis pipelines that handle large-scale genomic databases. Study shows TSLRMC exact string-*matching* algorithm used three pointers simultaneously to generate sliding text window of pattern *P[0…m-1]* with the text *T[i-m…i]*. TSLRMC exact pattern matching algorithm scans pattern last, first and middle characters in the text to select PTW. If left, right and middle characters are found at appropriate position in the text string then pattern is aligned with them, and new text window is selected which is PTW for remaining string of pattern. Otherwise continue scanning text for pattern left, right and middle characters. The time-complexity is *O (n-m)* in the worst case, *O (km)* in the average case and *O (1)* in the best case.

Text characters are consisting of billions of static English alphabets. Pattern characters are also static sentences collected from same text string. Experiments were conducted on the file size of 10 million characters with different pattern sizes (6, 12, 18,

24, 30, and 45). Experimental results show that TSLRMC exact string-matching algorithms are quite efficient than the existing algorithms

For future work, TSLRMC could be extended with multithreading capabilities to improve runtime performance on large datasets further. Additionally, we aim to optimize the algorithm for binary string inputs and explore its adaptableness for approximate string matching scenarios, which are common in noisy or error-prone DNA sequencing data.

7. REFERENCES

[1] S. Hakak, A. Kamsin, P. Shivakumara, M. Y. I. Idris, and G. A. Gilkar, "A new split based searching for exact pattern matching for natural texts," PLoS One, vol. 13, no. 7, p. e0200912, Jul. 2018.

[2] S. Elie, "An overview of Pattern Recognition," Apr. 2013.

[3] C. Charras and T. Lecroq, Handbook of Exact String-Matching Algorithms, p. 221.

[4] L. H. Keng, "Approximate String Matching With Dynamic Programming and Suffix Trees," p. 102.

[5] A. Karcioglu and H. Bulut, "q-frame hash comparison based exact string matching algorithms for DNA sequences," Concurrency and Computation: Practice and Experience, vol. n/a, no. n/a, p. e6505.

[6] Mohammad, O. Saleh, and R. A. Abdeen, "Occurrences Algorithm for String Searching Based on Brute-force Algorithm," Journal of Computer Science, vol. 2, no. 1, pp. 82–85, Jan. 2006.

[7] R. Rahim, I. Zulkarnain, and H. Jaya, "A review: search visualization with Knuth Morris Pratt algorithm," IOP Conference Series: Materials Science and Engineering, vol. 237, p. 012026, Sep. 2017.

[8] K. Al-Khamaiseh and S. ALShagarin, "A Survey of String Matching Algorithms," vol. 4, no. 7, p. 13, 2014.

[9] Leonardo and S. Hansun, "Text Documents Plagiarism Detection using Rabin-Karp and Jaro-Winkler Distance Algorithms," Indonesian Journal of Electrical Engineering and Computer Science, vol. 5, no. 2, p. 462, Feb. 2017.

[10] Z. Zhang, "Review on String-Matching Algorithm," SHS Web of Conferences, vol. 144, p. 03018, 2022.

[11] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," Journal of Molecular Biology, vol. 215, no. 3, pp. 403–410, Oct. 1990.

[12] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," Bioinformatics, vol. 25, no. 14, pp. 1754–1760, Jul. 2009.

[13] M. Crochemore and T. Lecroq, "Pattern matching and text compression algorithms," p. 66.

[14] "An improved algorithm for boyer-moore string matching in chinese information processing," in Proc. 2011 Int. Conf. Computer Science and Service System (CSSS), Nanjing, China: IEEE, Jun. 2011, pp. 182–184.

[15] Obeidat and M. AlZubi, "Developing a faster pattern

matching algorithms for intrusion detection system," International Journal of Computer, pp. 278–284, Sep. 2019.

[16] Aziz, S. Shoaib, K. S. Khurshid, T. Ahmad, and M. Awais, "Performance evaluation of DNA pattern matching algorithms," Pakistan Journal of Science, vol. 74, no. 3, pp. 169–175, Sep. 2022.

[17] T. Raita, "Tuning the boyer-moore-horspool string searching algorithm," Software: Practice and Experience, vol. 22, no. 10, pp. 879–884, 1992.