ISSN 2278-3091
Volume 9, No.4, July – August 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse316942020.pdf
https://doi.org/10.30534/ijatcse/2020/316942020

Intelligent Predictive String Search Algorithm Using Two Sliding Windows in Parallel Environment

Wafa Dababat [1], Mariam Itriq [2]
[1] Department Of SE, Al-Balqa Applied University, Salt, Jordan
w.dababat@bau.edu.jo
[2] Department Of IT, The University of Jordan, Amman, Jordan
m.itriq@ju.edu.jo

ABSTRACT

String matching is a fundamental problem in computer science and has been extensively studied. Searching for all occurrences of a pattern in a text is a fundamental problem in many applications, like natural language processing, information retrieval, pattern recognition and computational biology. Many string matching algorithms are existing and work efficiently with different applications in different life scopes; one of these algorithms is the Intelligent Predictive String Search Algorithm, this algorithm searches through a given text to find the first occurrence of a pattern without a pre-processing phase that included in many string marching algorithms to calculate the pattern shift values which lead less computations and uses simple rules during a match or mismatch of a pattern character using one sliding window.

In this paper we updated the Intelligent Predictive String Search Algorithm three times resulting with three versions; in the first one we reversed the search direction to be from right using one sliding window while in second version we use two sliding windows to scans the text from both sides sequentially and finally we parallelize this version using real parallel environment. Besides, it is easy to parallelize the new developed algorithm gain significant enhancement in decreasing time and memory requirements.

Key words: Pattern matching, Intelligent Predictive String Search Algorithm, Two Sliding Windows algorithm, parallel environment.

1. INTRODUCTION

Pattern matching algorithms are used in many applications to search for a certain pattern p of length m in a text t of length n, many algorithms exist that maintain this purpose, but they differ from each other in some aspects such as:
Number of sliding windows used in searching process, some algorithms use one window with size equal to pattern size [1-3]. Others using two or more sliding windows each with length equal to pattern length [4-9]. In this case, the comparisons between the text and two sliding windows happened at the same time at the both sides while other algorithms used four sliding windows [10].
Shift values, the shifting value vary from one algorithm to another, such variations depend on the number of consecutive characters in the text immediately after the pattern window [1-10].
In this paper, we made an enhancement on the Intelligent Predictive String Search Algorithm[11][12], while keeping the shift values used in the original Intelligent Predictive String Search Algorithm as it is but we use two sliding window instead of one, the window size is equal pattern size (m), but In this case the two sliding window moves according to the same shift value rules.
Comparisons are made between the new algorithm and the original Intelligent Predictive String Search Algorithm [11][12]. The experimental results section showed that the new algorithm is faster than the others in case of a number of comparisons. The rest of this paper is organized as follows: The next section introduces some literature review about the topic; it is followed by a section that covers the Intelligent Predictive String Search Algorithm. We then present the three adaptations on Intelligent Predictive String Search Algorithm followed by their analysis. The conclusion and future work are drawn in the last section.

2. RELATED WORKS

Several pattern matching algorithms have been developed and improved in the past decades to meet the different needs of different applications [13-18]. As we mentioned previously some of these algorithms used single sliding window to search the text for a certain pattern, while others used two or more sliding windows. On the other hand, some of these algorithms require two phases, pre-processing phase to calculate shift values that used by the sliding window and searching phase [1-10]. The shift values that the window will shift also varies from one algorithm to another, for example; the shifting values in the BoyerMoore algorithm (BM) [19] in case of a

mismatch (or a complete match of the whole pattern) depends on two pre-computed functions to shift the window to the right. These two shift functions are called the good-suffix shift and the bad-character shift. The pre-processing and searching time complexities of (BM) are O(m+$|\sum|$) and Ω(n/m), O(n).

Berry-Ravindran algorithm (BR) [20] depend on the bad character shift function to determine the shift value in case of a mismatch and the searching phase make use of one sliding window from left to the right. The pre-processing and searching time complexities of BR algorithm are O(σ2) and O(nm) respectively. EBR [6] algorithm made modifications on Berry-Ra-vindran bad character shift by using three consecutive characters.

Two Sliding Window algorithm TSW [5] enhanced (BR) by using two sliding windows instead of one, each of them equal to the length of the pattern n. One window aligned with the text from the left the other aligned from the right and the both windows shifted according to bad character shift. In TSW, the best time complexity is O(m) and the worst case time complexity is O(((n/2-m+1))(m)). The pre-process time complexity is O(2(m-1)).

In order to minimize the number of comparisons, Enhanced Two Sliding Window algorithm (ETSW) [7] made some modification on TSW. The preprocessing phase remains the same, and the modifications happened on the comparison process by using two pointers one from the left of the pattern and the other form the right of the same pattern. The same process applied to the two windows, the best time complexity is O(m/2) and the worst case time complexity is O(((n/2-m/2+1))(m/2)). The pre-process time complexity is O(2(m-1)).

ERS-A [4] uses two sliding windows in the searching process the same as used in TSW [5]. In addition to using RS-A[13] algorithm to calculate the shift values of the right pattern, some enhancement to calculate the shifting values for the left pattern was done to maximizes the efficiency of the searching process with O([n/(2*(m+4))]) time complexity in average case.

The Intelligent Predictive String Search Algorithm [11][12], that we're going to develop in several stages have the following properties:

- It does not require pre-processing phase.
- It finds the first occurrence of a pattern in a text that consists of words separated by a blank space.
- It makes use of one sliding window to search the text from left
- It uses two rules to make a shift namely alphabet-blank mismatch and alphabet-alphabet mismatch.

We are going to explain the algorithm in detail in the next section.

3. INTELLIGENT PREDICTIVE STRING SEARCH ALGORITHM

As we mentioned earlier this algorithm get rid of the pre-processing phase with its complex computations involved and makes an assumption that the text consists of words separated by a blank space and the search is made for complete words and not their substrings.

At each comparison this algorithm makes three main steps after aligning the leftmost character of the pattern P to the leftmost character of text T, the steps are simulated in Figure1:

(Compare -> Predict -> Act)

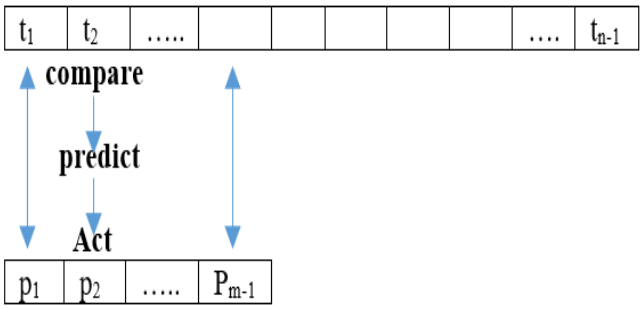

Figure 1: Schematic diagram for the algorithm steps.

Step1-Compare:

At the beginning, the leftmost end of the pattern window with size (m) is aligned with the same end of the text as shown in Figure2. At each alignment of the pattern, the algorithm works on the portion of the text with size equal to pattern size (m) this is known as the text window. The comparison between the first character of the text from the left and the first character of pattern window from left is made, this comparison leads to either match or mismatch.

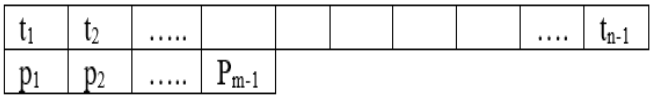

Figure2: Compare step

Step 2-Predict:

In case of match is found the rightmost character of the pattern is compared with the rightmost character of the current window. If this leads to a match, the remaining characters are compared from right to left. In case of mismatch at any other position the pattern is shifted by (m) characters (window size).

In case of mismatch of the leftmost or the rightmost character of the pattern, the two rules Alphabet-Blank mismatch and Alphabet-Alphabet mismatch are taken placed depending on the type of mismatch.

If t_1 is a blank space, then pattern shifted by one position to right, however the next position might be a possible beginning of the pattern.

If t_1 is different from pf_1, then next character might be another character of the same word or it might be a blank. In case of the first possibility the pattern can be shifted by one position to the right while if the second possibility arises the pattern can be shifted by two positions.
If t_1 match p_1, then check if t_m is a blank or not. If t_m is a blank this means that the text word currently checked is shorter than pattern, in this case the pattern is shifted by m positions towards the right
if t_m not a blank but differ from the corresponding pattern character and the next text character is blank, then pattern shifted by m positions towards the right

Step 3-Act:
In this step either a full match happened or the predicted shift value is taken place
These steps are shown in Figure3 [11][12].

```
1: Align T and P
2: Repeat steps 3 to 5 until a match is found or
   until the end of T is reached.
3: Compare P[0] and T[i]
4: if mismatch
        a: Alphabet-blank
                Re align P[0] to T[i+1]
        b: Alphabet-alphabet
                Re align P[0] to T[i+2]
5: if match compare P[m] to T[i+m]
        a: if match
            Compare remaining characters from
            right  to left
        b: if mismatch
            1: Alphabet-blank
                Re align P[0] to T[i+(m-1)+1]
            2: Alphabet-alphabet
                a: if T[i+(m+1)] is blank
                    Re align P[0] to T[i+(m-1)+2]
                b: else
                    Re align P[0] to T[i+2]
```

Figure 3: Act step-(Algorithm_predictive_search(T,P))

4. THE PROPOSED ADAPTATIONS ON THE INTELLIGENT PREDICTIVE STRING SEARCH ALGORITHM

In this paper we proposed three versions of Intelligent Predictive String Search Algorithm. The first two versions will enhance the performance of the algorithm under certain situations while the third version is shown to give better time and speedup of the original algorithm.
The first variation which is named Right Intelligent Predictive String Search Algorithm which is suitable for searching for the last occurrence of the pattern in the text. The second variation called LR Intelligent Predictive String Search Algorithm, this algorithm makes use of two sliding windows to scan the text from its both sides right and left. Finally, the third variation refereed as Parallel Intelligent Predictive String Search Algorithm, in this algorithm we adapted the original Intelligent Predictive String Search Algorithm to work under a parallel environment which leads to enhancement in the performance in all situations. And its valuable in cases where we are interested in all occurrences of the pattern. We have to mentioned here that all versions used the Alphabet-Blank and Alphabet-Alphabet rules in case of mismatch exactly as used in the original algorithm. Next, we will describe each variation separately in details.

4.1Right Intelligent Predictive String Search Algorithm

As we indicated earlier, this algorithm is a mirror version of the original algorithm, it will give better results than the original algorithm in case where the application is interested in the last occurrence of the pattern assuming an equal distribution of the pattern occurrences in the text.
Right predictive will start by placing the pattern's right by the rightest position in the text, comparing the right most character of the pattern pm-1 with the right most character of the text tn-1, this may result with mismatch where the pattern is moved one position to the left if the text's character is a blank or two positions to the left otherwise. In case of matching, an attempt to find a total match starts by comparing the pattern and the text from the right of the pattern. In case of a mismatch the pattern slides by its length (m) to the left if the mismatch character in the text is a blank, or by (m+1) if the character to the right of the mismatch character in the text is a blank. The way how the Right Intelligent Predictive String Search Algorithm works with working example is shown in Figure4 and Figure5 respectively.

```
1: Align P to the rightmost position in T, n= length of T,
m= length of P
2: Repeat steps 3 to 5 until a match is found or until the
start of T is reached.
3: Compare P[m-1] and T[(n)-1]
4: if mismatch
      a: Alphabet-blank
         Re align P[m-1] to T[(n-1)-1]
      b: Alphabet-alphabet
         Re align P[m-1] to T[[(n-1)-2]
5: if match compare P[0] to T[[(n-1)-m+1]
      a: if match
         Compare remaining characters from left to right
      b: if mismatch
         1: Alphabet-blank
            Re align P[0] to T[[(n-1)-m]
         2: Alphabet-alphabet
            a: if T[[(n-1)-m] is blank
               Re align P[0] to T[[(n-1)-m-1]
            b: else
               Re align P[0] to T[[(n-1)-2]
```

Figure 4: Right Intelligent Predictive Search Algorithm

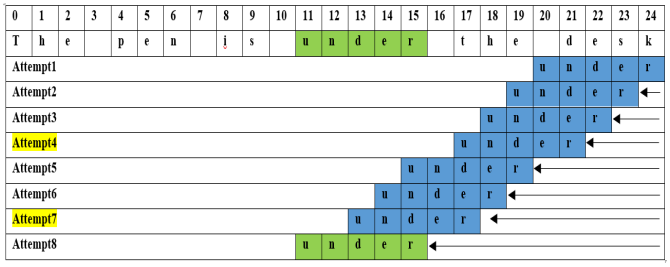

Figure 5: Working example on Right Intelligent Predictive Search Algorithm

4.1.1 Working Example

In this section, we will present an example to clarify the idea of Right Intelligent Predictive String Search Algorithm as shown in Figure5.

Given:

Pattern(P)=”under”, m=5,

Text(T)= ”The pen is under the desk”, n=25

Attempts (1-3): see Figure5 (attepmt1-3), In attempt1 we align the sliding window with the text from the right. In this case, a mismatch occurs between text character (k) and pattern character (r) and the text character$_{23}$(s) is not a blank; therefore, according to Alphabet-Alphabet mismatch rule the sliding window shifted to left one position, the same rules happened in attempts 2-3 when we align pattern character(r) with text character (s) and pattern character (r) with text character (e) respectively.

Attempt4: see Figure5 (attepmt4), In attempt4 a mismatch occurs between text character (d) and pattern character (r) and the text character$_{20}$ is a blank; therefore, according to Alphabet-Alphabet mismatch rule the sliding window shifted to left two positions.

Attempts (5-6): see Figure5 (attepmt5-6), In attempt5 a mismatch occurs between text character (e) and pattern character (r) and the text character$_{18}$(h) is not a blank; therefore, according to Alphabet-Alphabet mismatch rule the sliding window shifted to left one position, the same rules happened in attempt6 when we align pattern character(r) with text character (h).

Attempt7: see Figure5 (attepmt7), In attempt7 a mismatch occurs between text character (t) and pattern character (r) and the text character$_{16}$ is a blank; therefore, according to Alphabet-Alphabet mismatch rule the sliding window shifted to left two positions.

Attempt8: see Figure5 (attepmt8), In attempt8 a match occurs between text character$_{15}$ (r) and pattern character (r) therefore, we compare the text character$_{11}$(u) with pattern charchter$_0$ (u) since matching occurred the remaining characters are compared from right to left and a complete match of the whole pattern is found.

4.1 LR Intelligent Predictive String Search Algorithm (with two sliding windows)

In this version the algorithm tries to get a match of the pattern inside the text by using two sliding windows to scan the text string from two directions; from left to right and from right to left. In mismatch cases, during the searching process from the left, the left window is shifted to the right, while during the searching process from the right, the right window is shifted to the left. Both windows are shifted depending on Alphabet-Alphabet and Alphabet-Blank mismatch rules until the pattern is found or the windows reach the middle of the text. The way how the LR Intelligent Predictive String Search Algorithm works is shown in Figure 6.

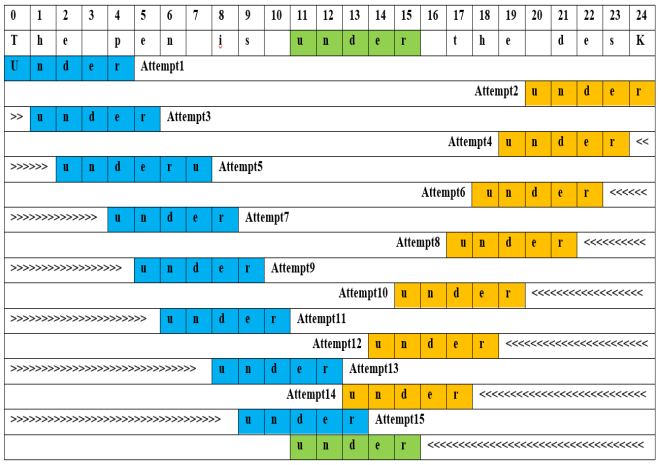

Figure 6: Working example on Intelligent Predictive Search Algorithm with two sliding windows

As shown in Figure6, we present an example to clarify the idea of Intelligent Predictive String Search Algorithm with two sliding windows moving sequentially before using parallel environment. Given:

Pattern(P)=”under”, m=5,

Text(T)= ”The pen is under the desk”, n=25

Attempts (1,3,5,7,9,11,13,15): see Figure6 these attempts follow the original Intelligent Predictive String Search Algorithm rules from left side; while attempts (2,4,6,8,10,12,14,16) follows the adaptive version that we named Right Intelligent Predictive String Search Algorithm as discussed early in section 4.1.

4.2 A parallel version of predictive pattern matching algorithms

Most computing environment nowadays have multi processors or multi cores. If an algorithm is converted to a parallel version properly, computations run time can be significantly decreased without compromising the quality of algorithm's output. In this part, an enhancement of predictive pattern matching algorithm is presented, experimental results of this part are presented in the next section.

Converting an algorithm to one of its parallel equivalent algorithms can be achieved either by decomposing the processing steps of the original algorithm among different simultaneous computation threads, or by having multiple identical threads that run the same steps on different input data. In this work, we follow the second method by decomposing the text and distribute it on different threads running at the same time to maximize speedup of the algorithm. This procedure can be applied on any of the Predictive algorithms: original, right predictive or LR predictive. The original predictive algorithm has been chosen for this work, but the same parallel algorithm can be applied to other versions of the predictive algorithm without difficulty.

The parallel algorithm consists of three main parts: first, Text is decomposed into n parts (n is the number of threads), each Text part with the pattern is sent to a different computation thread. Second, each thread will run predictive pattern matching algorithm on its text part. Finally, results of simultaneous threads are collected and a proper output is presented. The steps of the parallel predictive algorithm are shown in Figure 7.

Algorithm: parallel predictive search(T,P)

Step1: Decompose T to n parts T_1, T_2,....., Tn.

Step2: for i= 1 to n

1- create thread$_i$ (T_i, Pattern)

2- start thread$_i$ Predictive (T_i, Pattern)

Step3: collect results from threads and display output

Figure 7: Steps of the parallel Intelligent Predictive Search Algorithm

For text decomposition details we refer to [21]. The activity diagram in Figure 8 shows the workflow of the algorithm.

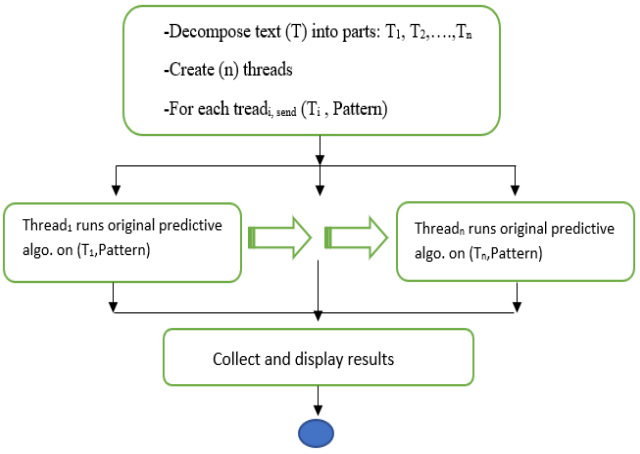

Figure 8: The workflow of a parallel Intelligent Predictive Search Algorithm

5. EXPERIMENTAL RESULTS

The proposed algorithms were tested for work with different experiments depending on random patterns with variable lengths from random parts of the text file. The same experiments were applied to the original predictive algorithm for comparison purposes.

For the implementation we used Java NetBeans 8.1 running on windows 10. We made the experiments on two different hardware environments, the first using dual core 2.2 GHz with 6GB RAM, and using 8 core 3.4 GHz with 8GB RAM in the second environment.

5.1 The results of LR predictive algorithm

To make the tests, we considered a text file with 125032 words. Several experiments were conducted. In the first part of this work, patterns were randomly selected from the first part of the text file. Next, we considered random variable length patterns from the last part of text file. The number of attempts and comparisons needed is presented in tables Table 1 and Table 2. In Table 3 we show the averages of all attempts.

Table 1: Results when pattern is selected from the first part of the text file.

	Original Predictive		Predictive with two windows	
Pattern length	Attempts	Comparisons	Attempts	Comparisons
5	2056	2388	4192	4870
6	9969	11764	19460	23306
7	9960	11531	12775	14237
8	6279	7273	12519	14587
9	9561	11903	15088	17792
10	30770	36167	52153	57524
11	13220	15432	25457	30171
12	16805	19332	32773	38090
16	19051	22478	38360	45347
18	38925	45443	77066	90699

Table1 contains the number of comparisons and number of attempts needed for searching a pattern in a text file where the pattern is selected from the first part of the text file. In each experiment we considered different patterns having the same length and applied the search using both the two-sliding predictive algorithm and the original predictive algorithm, the average of attempts and comparisons for each pattern length was recorded.

Table2 contains the results of taking patterns from the last part of the text file. The same procedure was applied as mentioned previously.

Table 2: Results when pattern is selected from the last part of the text file.

	Original Predictive		Predictive with two windows	
Pattern length	**Attempts**	**Comparisons**	**Attempts**	**Comparisons**
5	46576	53716	6377	7426
6	20498	24007	3926	4618
7	183005	213450	20998	24557
8	260614	279320	23568	24684
9	253838	253847	40485	47386
10	334088	396483	24581	28984
11	309029	346904	25517	27620
12	389474	450002	30365	35366
13	292606	343721	45161	45161
18	320133	324810	14471	14471

Table 3 contains the average of all attempts from the first and last part of the text file.

Table 3: Average of all results from the first and last parts of the text file

	Original Predictive		Predictive with two windows	
Pattern length	**Attempts**	**Comparisons**	**Attempts**	**Comparisons**
5	25344	29246	7381	8583
6	20218	23768	21423	25615
7	101463	118256	23274	26516
8	136586	146933	24303	26929
9	136480	138827	35331	41485
10	197814	234409	64444	72016
11	167735	188884	38216	43981
12	211542	244333	47956	55773
13	165354	194339	60941	67928
18	198992	207848	84302	97935

The results of experiments are shown in Figure9, Figure10 and Figure 11 respectively.

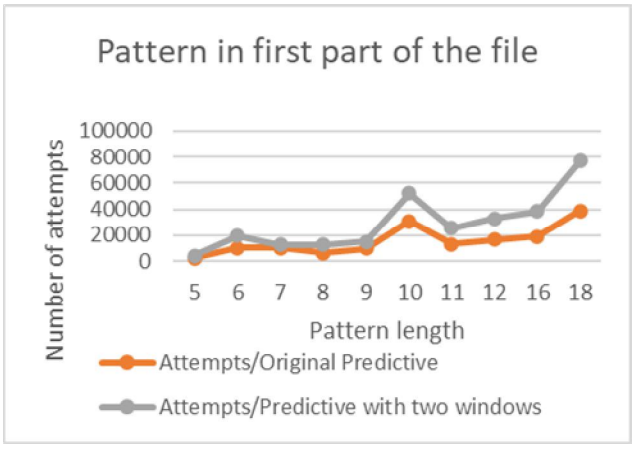

Figure 9: Average number of attempts and comparisons when pattern is in first part of the text file

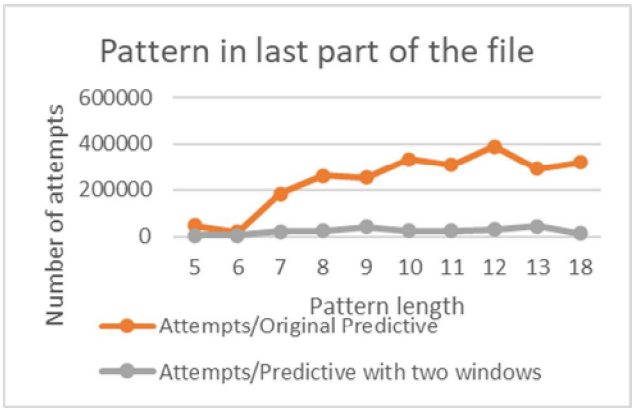

Figure 10: Average number of attempts and comparisons when pattern is in last part of the text file

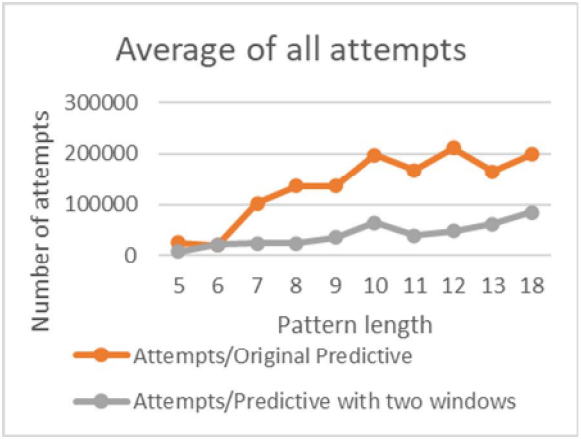

Figure 11: Average number of all attempts and comparisons when pattern is in last of first part of the text file

5.1.1 Analysis of results of the LR predictive algorithm

When considering the results, we notice that the original predictive algorithm will always show better performance in cases where the pattern is in the first part of the text file. This is clear in Figure8.

In cases where the pattern is selected from the last part of the text file, it is clear that the two sliding window predictive algorithm will give a better performance in all cases. Figure9 shows these results. These results make sense, the original predictive algorithm will go all the way from the left to the right part of the text to find the pattern, whereas the LR predictive algorithm will try matching from one side at a time yielding a better performance.

When considering the overall results shown in Figure10, we conclude that in average, the two-window predictive algorithm will give better results in most cases.

We have noticed that the performance of the two-window predictive algorithm will tend to degrade when that pattern is selected from the middle part of the text file, in this case the original predictive algorithm always gave better results.

5.2 Results of the parallel predictive search algorithm

First, we applied the algorithm on the hardware environment that has dual core processor. Using the same data file, we fixed the pattern size, taking different patterns with the same size. We stated by taking one thread, applying the algorithm by searching one pattern using one thread at a time. The time of each search experiment was recorded. When finished, the average of all experiments using one thread was recorded.

Then, we used the same patterns from the previous step, and repeated the search using two threads this time, and results were recorded. We continued the same process by maintaining the same patterns and repeating the search by increasing the number of threads one at a time until we reached 5 threads.

Speedup is defined as the ratio of the worst-case execution time of the fastest known sequential algorithm for a particular problem to the worst-case execution time of the parallel algorithm. The more speedup gained, the better the parallel algorithm is. The time(ns) and speedup are recorded in Table4 and Table5 respectively.

Table4 contains the results from applying the parallel predictive pattern matching algorithm on a dual core processor, by fixing the pattern size and increasing the number of threads on each trial.

Table 4: Processing time(ns) of the parallel predictive algorithm on a dual core processor.

Number of threads	time(ns)
1	95048064
2	51194189
3	52402024
4	52938293
5	57938293

Table5 shows the speedup gained by applying the parallel predictive algorithm on a dual core processor.

Table 5: Speedup of the parallel predictive algorithm on a dual core processor.

Number of threads	speedup
1	1
2	1.856618219
3	1.81382429
4	1.79545011
5	1.64050508

The results of the parallel predictive algorithm on a dual core processor are depicted in Figure12 that shows the time(ns) and Figure13 that shows the speedup.

Next, we repeated the same process on 8 core processor, we again fixed the pattern size, starting with one thread, increasing the number of threads one at a time and recording the average search time and speedup until we reached 10 threads. The average time(ns) results are shown in Table6, while Table7 contains the speedup results of the search experiments.

Table 6: Processing time(ns) of the parallel predictive algorithm on 8 core processor.

Number of threads	time(ns)
1	53154189
2	36223356
3	33341856
4	26521275
5	21591023
6	20973559
7	18112191
8	16859120
9	19120663
10	26525240

Table 6 contains the results of applying a multithreading parallel predictive algorithm on 8 core processor. The average search time for a number of patterns having the same length and varying the number of threads.

Table 7: Speedup of the parallel predictive algorithm on 8 core processor

Number of threads	speedup
1	1
2	1.467401005
3	1.59421806
4	2.004209413
5	2.461865239
6	2.534342836
7	2.934718886
8	3.15284481
9	2.779934409
10	2.003909823

Table7 shows the speedup obtained by applying the parallel predictive algorithm on multithreading environment on 8 core processor.

Time and speedup results of the experiments on 8 core processor are presented in Figure14 that presents the processing time and Figure15 showing the speedup.

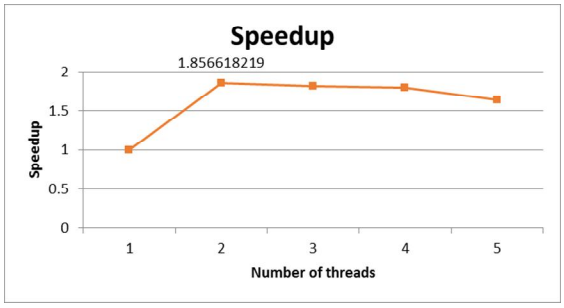

Figure 12: Time(n) of the parallel predictive search algorithm on a dual core processor

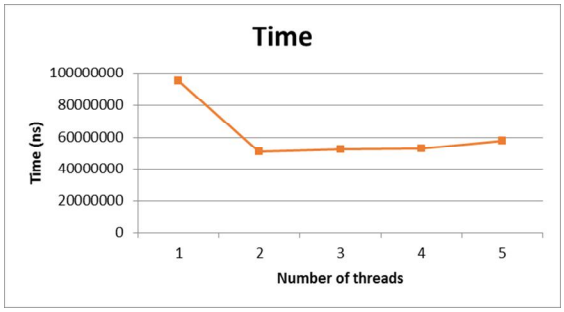

Figure 13: Speedup of the parallel predictive search algorithm on a dual core processor

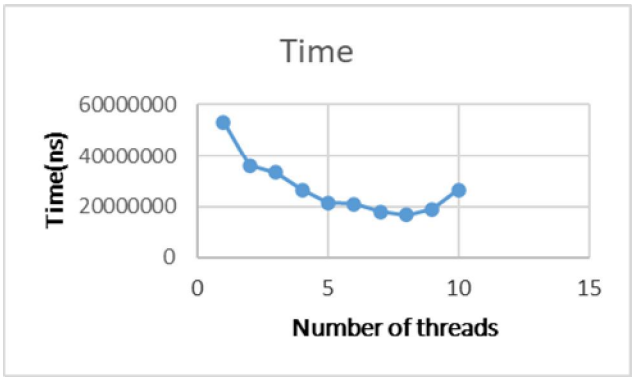

Figure 14: Time(n) of the parallel predictive search algorithm on 8 core processor

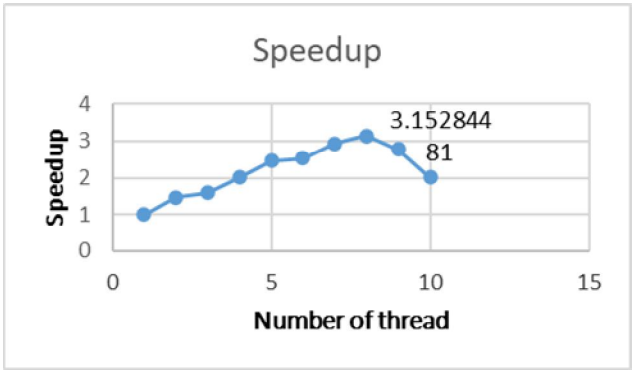

Figure 15: Speedup of the parallel predictive search algorithm on 8 core processor

5.2.1 Analysis of results of the LR parallel predictive algorithm

In the first parallel environment that has two CPUs, the lowest average search time was 51194189 ns and highest speedup gained was 1.856618219. These numbers represent the best time and best speedup respectively. Both best results were obtained when the number of threads was two, which is equal to the number of CPUs in the machine. We can see that no further improvement can be done by increasing the number of threads, since the time tends to increase rather than decreasing after this peak point. This increase of time is due to communication overhead increase when number of threads increase.

When considering search results of the 8 core processor experiments, it is clear that the minimum average search time is 16859120 ns and the highest speedup is 3.15284481. These two results are gained when the number of threads was 8. Which is equal to the number of CPUs. And again, tending to increase the number of parallel threads more than 8 is of no benefit, since it increases the search time.

In both experiments, one thread represents the sequential original predictive search algorithm. It is obvious that the parallel algorithm will give better results than the

sequential original algorithm in all cases. As seen, the worst time in all recorded times is that of the case having one thread.

So, depending on the obtained results, we can say that the parallel version of the predictive algorithm will always give better results than the original version.

Regarding the parallel predictive algorithm, the best search time and speedup will always be obtained when the number of threads is equal to the number of CPUs in the parallel machine.

6. CONCLUSIONS

In this present paper, three proposed versions of the Intelligent Predictive String Search Algorithm have been successfully implemented and tested. The experimental results of proposed LR parallel predictive algorithm method satisfy the lowest average search time and the highest speedup compared with the other approaches.

REFERENCES

1. Hudaib A., Suleiman D., Awajan A. 2016. **A Fast Pattern Matching Algorithm Using Changing Consecutive Characters**, Journal of Software Engineering and Applications, 399-411.
2. Ganardi M., Hucke D., Lohrey M., 2018. **Sliding Window Algorithms for Regular Languages**, International Conference on Language and Automata Theory and Applications LATA 2018, 26-35.
3. AbdulRazzaq A., Abdul Rashid N., Hasan A., Abu-Hashem M., Zaino Z.2016 **New Searching Technique of Hybrid Exact String Matching Algorithm, International Review on Computers and Software** (I.RE.CO.S.), Vol. 11, N. 10 ISSN 1828-6003 https://doi.org/10.15866/irecos.v11i10.10321
4. SULEIMAN D,HUDAIB A, AL-ANANI A,AL-KHALID R & ITRIQ M, 2013. **ERS-A Algorithm for Pattern Matching**. Middle East Journal of Scientific Research, 15(7), 1067-1075.
5. HUDAIB A., AL-KHALID R., SULEIMAN D., ITRIQ M. & AL-ANANI A, 2008. **A Fast Pattern Matching Algorithm with Two Sliding Windows (TSW)**. Journal of Computer Science, 4(5), 393-401.
6. SULEIMAN D, 2014. **Enhanced Berry Ravindran Pattern Matching Algorithm (EBR).** Life Science Journal, 11(7), 395- 402.
7. ITRIQ M., HUDAIB A., AL-ANANI A., AL-KHALID R. & SULEIMAN D, 2012. Enhanced **Two Sliding Windows Algorithm for Pattern Matching (ETSW)**. Journal of American Science, 8(5), 607- 616.
8. Hudaib A., Suleiman D. & Awajan A, (2016, April). **Dynamic Berry Ravindran Algorithm for Pattern Matching (DBR),** 6th International Conference on Applied Computer Science (ACS '16), At Istanbul, Turkey, (pp 15-17).
9. SULEIMAN D., ITRIQ M., AL-ANANI A., Al-KHALID R. & HUDAIB A, 2015. **Enhancing ERS-A Algorithm for Pattern Matching (EERS-A).** Journal of Software Engineering and Applications, 8, 143-153. https://doi.org/10.4236/jsea.2015.83015
10. HUDAIB A., Al-KALID R., AL-ANANI A, ITRIQ M & SULEIMAN D, 2015. **Four Sliding Windows Pattern Matching Algorithm (FSW).** Journal of Software Engineering and Applications, 8, 154-165.
11. Gurung D, Chakraborty U K and Sharma P (2016), **Intelligent Predictive String Search Algorithm, Procedia Computer Science**, Vol. 79, pp.161-69 https://doi.org/10.1016/j.procs.2016.03.116
12. Gurung D, Chakraborty U K and Sharma P (2017), **An analysis of the Intelligent Predictive String Search Algorithm: A Probabilistic Approach**, Information Technology & Computer Science, Vol. 2, pp.66-75,
13. Senapati, K.K., Mal, S. and Sahoo, G. (2012) RS-A **Fast Pattern Matching Algorithm for Bio-Logical Sequences**. In- ternational Journal of Engineering and Innovative Technology (IJEIT), 1, 116-118.
14. Singla N. & Garg D. 2012. **String Matching Algorithms and their Applicability in various Applications**, International Journal of Soft Computing and Engineering (IJSCE) 1(6).
15. CHAO Y. 2012. **An Improved BM Pattern Matching Algorithm in Intrusion Detection System**. Applied Mechanics and Materials, vol. 148 – 149, 1145-1148.
16. SENAPATI K.K., MAL S. & SAHOO G. 2012. **RS-A Fast Pattern Matching Algorithm for Bio-logical Sequences**. International Journal of Engineering and Innovative Technology (IJEIT), 1(3), 116- 118.
17. D.N. Goswami,Dr. Anshu Chaturvedi,Raghuvanshi C.S , **Efficient Algorithm for Frequent Pattern Mining Based On Apriori**, International Journal of Advanced Trends in Computer Science and Engineering, July 2010.
18. Irma T. Plata, Allan C. Taracatac, **Edward B. Panganiban,Development and Testing of Embedded System for Smart Detection and Recognition of Witches' Broom Disease on Cassava Plants using Enhanced Viola-Jones and Template Matching Algorithm**, International Journal of Advanced Trends in Computer Science and Engineering, October 2019.
19. Boyer, R.S. and Moore, J.S. (1977) **A Fast String Searching Algorithm.** Communications of the

Association for Com- puting Machinery, 20, 762-772.
https://doi.org/10.1145/359842.359859

20. BERRY, T. & RAVINDRAN, S., 2001**. A Fast String Matching Algorithm and Experimental Results.** In Proceedings of the Prague Stringology Club Workshop "99 (eds Holub, J.and Simanek, M), Collaborative Report DC-99-05, Czech Technical University, Prague, Czech Republic, 16-26.
21. Wafa Dababat, Mariam Itriq, **Parallel Enhanced Pattern Matching Algorithm with Two Sliding Windows PETSW**, International Journal of Computer Applications (0975 – 8887) Volume 179 – No.18, February 2018.
https://doi.org/10.5120/ijca2018916317