A Fast Pattern Matching Algorithm with Two Sliding Windows (TSW)

Amjad Hudaib, Rola Al-Khalid, Dima Suleiman, Mariam Itriq and Aseel Al-Anani
Department of Computer Information Systems, University of Jordan, Amman 11942 Jordan

Abstract: In this research, we propose a fast pattern matching algorithm: The Two Sliding Windows (TSW) algorithm. The algorithm makes use of two sliding windows, each window has a size that is equal to the pattern length. Both windows slide in parallel over the text until the first occurrence of the pattern is found or until both windows reach the middle of the text. The experimental results show that TSW algorithm is superior to other algorithms especially when the pattern occurs at the end of the text.

Key words: Pattern matching, string matching, berry-ravindran algorithm, boyer moore

INTRODUCTION

Pattern matching is a pivotal theme in computer research because of its relevance to various applications such as web search engines, computational biology, virus scan software, network security and text processing[1-4].

Pattern matching focuses on finding the occurrences of a particular pattern P of length ‘m’ in a text ‘T’ of length ‘n’. Both the pattern and the text are built over a finite alphabet set called \( \Sigma \) of size \( \sigma \).

Generally, pattern matching algorithms make use of a single window whose size is equal to the pattern length[5]. The searching process starts by aligning the pattern to the left end of the text and then the corresponding characters from the pattern and the text are compared. Character comparisons continue until a whole match is found or a mismatch occurs, in either case the window is shifted to the right in a certain distance[6-12]. The shift value, the direction of the sliding window and the order in which comparisons are made varies in different pattern matching algorithms.

Some pattern matching algorithms concentrate on the pattern itself[5]. Other algorithms compare the corresponding characters of the pattern and the text from left to right[6]. Others perform character comparisons from right to left[8,11]. The performance of the algorithms can be enhanced when comparisons are done in a specific order[9,13]. In some algorithms the order of comparisons is irrelevant such as Brute Force and Horspool algorithms[7].

In this study, we propose a new pattern matching algorithm: The Two Sliding Windows algorithm (TSW). The algorithm concentrates on both the pattern and the text. It makes use of two windows of size that is equal to the size of the pattern. The first window is aligned with the left end of the text while, the second window is aligned with the right end of the text. Both windows slide at the same time (in parallel) over the text in the searching phase to locate the pattern. The windows slide towards each other until the first occurrence of the pattern from either side in the text is found or they reach the middle of the text. If required, all the occurrences of the pattern in the text can be found.

Related works: Several pattern matching algorithms have been developed with a view to enhance the searching processes by minimizing the number of comparisons performed[14-16]. To reduce the number of comparisons, the matching process is usually divided into two phases. The pre-processing phase and the searching phase. The pre-processing phase determines the distance (shift value) that the pattern window will move. The searching phase uses this shift value while searching for the pattern in the text with as minimum character comparisons as possible.

In Brute Force algorithm (BF), no pre-processing phase is performed. It compares the pattern with the text from left to right. After each attempt, it shifts the pattern by exactly one position to the right. The time complexity of the searching phase is \( O(nm) \) in the worst case and the expected number of text character comparisons is \( 2n \).

New ways to reduce the number of comparisons performed by moving the pattern more than one position are proposed by many algorithms such as Boyer-Moore (BM)[11,17] and Knuth-Morris-Pratt algorithms (KMP)[6,18].
The KMP algorithm compares the pattern with the text from left to right. If a mismatch occurs it uses the failure function f(j) that indicates the proper shift of the pattern \[^{6,18}\]. The failure function f(j) is defined as the length of the longest prefix of P that is the suffix of P[1..j]. Thus, KMP reduces the number of times it compares each character in P with a character in the text T. KMP performs (2n) text character comparisons and the complexity of the pre-processing phase is O (m). KMP achieves a running time of O (n+m), which is optimal in the worst case\[^{6}\].

The BM algorithm improves the performance by pre-processing the pattern using two shift functions: the bad-character shift and the good-suffix shift. During the searching phase, the pattern is aligned with the text and it is scanned from right to left. If a mismatch occurs, the BM algorithm shifts the pattern with the maximum value taken between the two shift functions. The worst case time complexity when searching all occurrences of the pattern is O (mn) and O (nm\(^{-1}\)) for best performance\[^{11,17}\].

A simplification of BM algorithm is the Horspool algorithm\[^{7}\]. It does not use the good suffix function, instead it uses the bad-character shift with the rightmost character. Its pre-processing time complexity is O(m\(\sigma\)) and the searching time complexity is O(mn)\[^{7}\].

The Berry-Ravindran algorithm (BR) calculates the shift value based on the bad character shift for two consecutive text characters in the text immediately to the right of the window. This will reduce the number of comparisons in the searching phase. The pre-processing and searching time complexities of BR algorithm are O(m\(\sigma\)) and O(mn) respectively\[^{7}\]. In this research, the proposed algorithm makes use of the pre-processing phase of BR algorithm.

The Two Sliding Windows (TSW) algorithm: The Two Sliding Windows algorithm (TSW) scans the text from both sides simultaneously. It uses two sliding windows, the size of each window is m which is the same size as the pattern. The two windows search the text in parallel. The text is divided into two parts: the left and the right parts, each part is of size \(\frac{n}{2}\). The left part is scanned from left to right using the left window and the right part is scanned from right to left using the right window. Both windows slide in parallel which makes the TSW algorithm suitable for parallel processors structures. TSW algorithm stops when one of the two sliding windows finds the pattern or the pattern is not found within the text string at all. The TSW algorithm finds either the first occurrence of the pattern in the text through the left window or the last occurrence of the pattern through the right window. If necessary, the algorithm can be modified easily to find all the occurrences of the pattern. Also if the pattern is exactly in the middle of the text, TSW can find it easily.

The TSW algorithm utilizes the idea of BR bad character shift function\[^{8}\] to get better shift values during the searching phase. BR algorithm provides a maximum shift value in most cases without losing any characters. The main differences between TSW algorithm and BR algorithm are:

- TSW uses two sliding windows rather than using one sliding window to scan all text characters as in BR algorithm
- The TSW uses two arrays, each array is a one dimensional array of size (m-1). The arrays are used to store the calculated shift values for the two sliding windows. The shift values are calculated only for the pattern characters. While the original BR algorithm uses a two-dimensional array to store the shift values for all the alphabets\[^{8}\]. Using one dimensional array reduces the search processing time and at the same time reduces the memory requirements needed to store the shift values

Pre-processing phase: The pre-processing phase is used to generate two arrays \(\text{nextl}\) and \(\text{nextr}\), each array is a one-dimensional array. The values of the \(\text{nextl}\) array are calculated according to Berry-Ravindran bad character algorithm (BR). \(\text{nextr}\) contains the shift values needed to search the text from the left side. To calculate the shift values, the algorithm considers two consecutive text characters a and b which are aligned immediately after the sliding window. Initially, the indexes of the two consecutive characters in the text string from the left are \((m+1)\) and \((m+2)\) for a and b respectively as in Eq. 1.

For \(\text{bad char}\) shift \([a,b]\) = \(\min\) \begin{align} \text{1} & \quad \text{if p}[m-l]=a \\ m-i & \quad \text{if p}[i+1]=ab \\ m+1 & \quad \text{if p}[0]=b \\ m+2 & \quad \text{Otherwise} \end{align} (1)

On the other hand, the values of the \(\text{nextr}\) array are calculated according to our proposed shift function. \(\text{nextr}\) contains the shift values needed to search the text from the right side, initially the indexes of the two consecutive characters in the text string from the right
Fig. 1: The pre-processing algorithm

are \((n-m-2)\) and \((n-m-1)\) for \(a\) and \(b\) respectively, which are used to calculate the shift values as in Eq. 2.

The two arrays will be invariable during the searching process. Figure 1 shows the steps of the pre-processing algorithm.

Searching phase: In this phase, the text string is scanned 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 until the pattern is found or the windows reach the middle of the text. Figure 2 explains the steps of the TSW algorithm.

Step 1: Compare the characters of the two sliding windows with the corresponding text characters from both sides. If there is a mismatch during comparison from both sides, the algorithm goes to step 2, otherwise the comparison process continues until a complete match is found. The algorithm stops and displays the corresponding position of the pattern on the text string. If we search for all the pattern occurrences in the text string, the algorithm continues to step 2.

Step 2: In this step, we use the shift values from the next arrays depending on the two text characters placed immediately after the pattern window. The two characters are placed to the right side of the left window and to the left side of the right window. The corresponding windows are shifted to the correct positions based on the shift values, the left window is shifted to the right window and the right window is shifted to the left. Both steps are repeated until the first occurrence of the pattern is found from either sides or until both windows are positioned beyond \(n/2\).

Working example: In this study we will present an example to clarify the TSW algorithm. Part of nucleotide sequence of a gene (only 47 nucleotides) from Chromosome 1 (CHR-I) has been used to test the algorithm\(^{[10]}\), this sequence is taken from the gene index 32854-32901\(^{[10]}\). The plant genome (Arabidopsis thaliana) consists of 27,242 gene sequences distributed over five chromosomes (CHR-I to CHR-V).
Fig. 3: The nextl and nextr arrays

**Given:**

*Pattern (P) = "GAATCAAT", \( m = 8 \)*

*Text (T) = "ATCTAACATCATAACCCTAATTGGCA GAGAGAATCAATCGAATCA", \( n = 47 \)*

**Pre-processing phase:** Initially, \( \text{shiftl} = \text{shiftr} = m + 2 = 10 \).

The shift values are stored in two arrays nextl and nextr as shown in Fig. 3a and 3b respectively.

To build the two next arrays (nextl and nextr), we take each two consecutive characters of the pattern and give it an index starting from 0. For example for the pattern structure GAATCAAT, the consecutive characters GA, AA, AT, TC, CA, AA and AT are given the indexes 0, 1, 2, 3, 4, 5 and 6 respectively.

The shift values for the nextl array are calculated according to Eq. 1 while the shift values for the nextr array are calculated according to Eq. 2.

**Searching phase:** The searching process for the pattern \( p \) is illustrated through the working example as shown in Fig. 4.

**First attempt:** In the first attempt (Fig. 4a), we align the first sliding window with the text from the left. In this case, a mismatch occurs between text character (A) and pattern character (T), therefore we take the two consecutive characters from the text at index 8 and 9 which are (A and A) respectively. To determine the amount of shift (shiftr), we have to do the following two steps:

- We find the index of TC in the pattern which is 3
- Since we search from the left side we use nextl array and shiftr = nextl[3] = 5

Therefore the window is shifted to the right 5 steps.

**Second attempt:** In the second attempt (Fig. 4b), we align the second sliding window with the text from the right. In this case, a mismatch occurs between text character (A) and pattern character (T), therefore we take the two consecutive characters from the text at index 37 and 38 which are (A and A) respectively. To determine the amount of shift (shiftr), we have to do the following two steps:

- We find the index of AA in the pattern, AA has two indexes 1 and 5
- Since we search from the right side we use nextr array for the two indexes
  
  \[ \text{nextr}[1] = 3, \text{nextr}[5] = 7 \]
  
  Then we choose the minimum value to determine shiftr. Shiftr = nextr[1] = 3.

Therefore the window is shifted to the left 3 steps.

**Third attempt:** In the third attempt (Fig. 4c), a mismatch occurs from the left between text character (A) and pattern character (G), therefore we take the two consecutive characters from the text at index 13 and 14 which are (A and C) respectively, since AC is not found in the pattern, so the window is shifted to the right 10 steps.

**Fourth attempt:** In the fourth attempt (Fig. 4d), a mismatch occurs from the right between text character (A) and pattern character (T), therefore we take the two consecutive characters from the text at index 34 and 35 which are (A and T) respectively. To determine the amount of shift (shiftr) we have to do the following two steps:

- We find the index of AT in the pattern, AA has two indexes 2 and 6
- Since we search from the right side, we use nextr array for the two indexes

  \[ \text{nextr}[2] = 4, \text{nextr}[6] = 8 \]
  
  We choose the minimum.

  Shiftr = nextr[1] = 4.

Therefore the window is shifted to the left 4 steps.

**Fifth attempt:** We align the leftmost character of the pattern \( P[0] \) with \( T[32] \). A comparison between the pattern and the text characters leads to a complete match at index 32. In this case, the occurrence of the pattern is found using the right window.
Analysis:

**Preposition 1:** The space complexity is $O(2^{(m-1)})$ where $m$ is the pattern length.

**Preposition 2:** The pre-process time complexity is $O(2^{(m-1)})$.

**Lemma 1:** The worst case time complexity is $O((n/2-m+1))m$)

**Proof:** The worst case occurs when at each attempt, all the compared characters of both the pattern and the text are matched except the last character and at the same time the shift value is equal to 1. If the pattern is aligned from the left then shift by one occurs when the first character of the two consecutive characters is matched with the last pattern character, while if the pattern is aligned from the right then shift by one occurs when the second character of the two consecutive characters is matched with the first pattern character.

**Lemma 2:** The best case time complexity is $O(m)$.

**Proof:** The best case occurs when the pattern is found at the first index or at the last index $(n-m)$.

**Lemma 3:** The Average case time complexity is $O(n/(2\times(m+2)))$.

**Proof:** The Average case occurs when the two consecutive characters of the text directly following the sliding window is not found in the pattern. In this case, the shift value will be $(m+2)$ and hence the time complexity is $O([n/(2\times(m+2))])$.

#### RESULTS AND DISCUSSION

Several experiments have been conducted using TSW algorithm. In each experiment, we consider Book1 from the Calgary corpus to be the text.

| Pattern length | TSW Attempts | TSW Comparisons | BR Attempts | BR Comparisons |
|----------------|--------------|-----------------|-------------|----------------|
| 4              | 8103         | 3904            | 4213        | 6409           | 7039           |
| 5              | 4535         | 4456            | 4809        | 9577           | 10645          |
| 6              | 2896         | 7596            | 8311        | 11953          | 13345          |
| 7              | 1167         | 10856           | 11087       | 13256          | 14807          |
| 8              | 681          | 9538            | 10538       | 14149          | 15892          |
| 9              | 382          | 9283            | 10272       | 14127          | 15799          |
| 10             | 191          | 5451            | 5967        | 12808          | 14243          |
| 11             | 69           | 6384            | 7168        | 9598           | 10923          |
| 12             | 53           | 7947            | 8673        | 10334          | 11370          |
| 13             | 139          | 19437           | 21319       | 19548          | 21673          |
| 14             | 32           | 19682           | 21739       | 19817          | 22384          |
| 15             | 10           | 20029           | 21596       | 26086          | 28644          |
| 16             | 3            | 21897           | 25404       | 22554          | 28148          |

Table 1: The average number of attempts and comparisons of TSW and BR algorithms.

The searching process is performed from both sides of Book1 and the pattern is located from either side.

The average number of attempts and comparisons respectively.

In Table 1, the length of the pattern is given in column one while the second column is the number of words selected for each pattern length from Book1. For example, as shown in Table 1, 1167 patterns with length 8, were taken. The average number of
comparisons made by TSW algorithm is 11087 and average number of attempts is 10056, while the average number of comparisons made by BR is 14807 and average number of attempts is 13256. TSW made the minimum average number of comparisons and average number of attempts. Although TSW algorithm uses the same shift function of BR algorithm the TSW algorithm searches the text from both sides simultaneously, while BR algorithm only searches the text from the left side, so the average number of comparisons and attempts in BR algorithm are more than that of our algorithm.

Table 2 shows the results of comparing TSW algorithm with other algorithms. TSW algorithm has the minimum average number of comparisons and attempts among all other algorithms. The results are reasonable since TSW algorithm searches the text from both sides while all other algorithms search the text from one side. This can be justified by the following two advantages of TSW algorithm. First, it searches the text from both sides simultaneously. Second, the BR shift function shifts the pattern by a value that ranges from 1 up to m+2 positions from both sides when a mismatch occurs. This has a positive effect on the number of comparisons and attempts in most cases.

BF algorithm in Table 2, has the largest number of comparisons and attempts because it shifts the pattern by one position to the right each time a mismatch occurs. Noticeably, KMP algorithm has almost the same number of comparisons and attempts as in BF algorithm. Although, KMP uses a failure function to determine the shift values in case of a mismatch, its searching results are close to BF because multiple occurrence of a substring in a word is not common in a natural language. The performance of BM algorithm is better than KMP and BF, since it uses the good suffix function and the bad-character shift to calculate the shift which depends on the reoccurrence of the substring in a word which reduces the number of comparisons performed.

Table 3-5 show the index, number of comparisons and attempts needed to search for the first appearance of the pattern (with different lengths) in the beginning, middle and at the end of Book1. In Table 3 and 4, TSW algorithm results are reasonably
Table 3: The number of attempts and comparisons performed to search for the first appearance of a selected pattern from the beginning of the text

| Pattern length | TSW | BR | BM | KMP | BF |
|----------------|-----|----|----|-----|----|
|                | Index | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons |
| 4              | 67   | 25  | 29 | 13  | 17 | 18  | 22 | 68  | 72 | 68  | 72 |
| 5              | 33   | 11  | 15 | 6   | 10 | 8   | 12 | 34  | 38 | 34  | 38 |
| 6              | 82   | 23  | 28 | 12  | 17 | 16  | 22 | 83  | 89 | 83  | 89 |
| 7              | 39   | 11  | 17 | 6   | 12 | 7   | 14 | 40  | 46 | 40  | 46 |
| 8              | 99   | 21  | 28 | 11  | 18 | 16  | 24 | 100 | 109| 100 | 109|
| 9              | 260  | 51  | 65 | 31  | 44 | 35  | 43 | 259 | 278| 261 | 280|
| 10             | 590  | 105 | 120| 55  | 69 | 74  | 93 | 589 | 607| 591 | 607|
| 11             | 189  | 35  | 47 | 16  | 26 | 19  | 29 | 190 | 202| 190 | 202|
| 12             | 2401 | 363 | 402| 198 | 226| 254 | 269| 2398| 2537|2402|2541|

Table 4: The number of attempts and comparisons performed to search for the first appearance of a selected pattern from the middle of the text

| Pattern length | TSW | BR | BM | KMP | BF |
|----------------|-----|----|----|-----|----|
|                | Index | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons |
| 4              | 370383 | 129947 | 137725 | 6258 | 72993 | 92820 | 98573 | 370384 | 370352 |
| 5              | 370382 | 113693 | 136161 | 57118 | 63667 | 76853 | 81719 | 370310 | 430611 |
| 6              | 370381 | 98091  | 106374 | 49270 | 54977 | 57885 | 61504 | 368479 | 378230 |
| 7              | 370380 | 87631  | 99564  | 41400 | 47672 | 46915 | 50528 | 365718 | 388684 |
| 8              | 370379 | 79241  | 92310  | 40177 | 44166 | 46615 | 52106 | 361018 | 398145 |
| 9              | 370378 | 71229  | 80309  | 37118 | 41356 | 41758 | 51656 | 356576 | 380896 |
| 10             | 370377 | 66303  | 71378  | 32152 | 370381 | 378230 | 370378 | 395697 |
| 11             | 370376 | 61585  | 71378  | 32152 | 370381 | 378230 | 370378 | 395697 |
| 12             | 370375 | 54745  | 63718  | 29387 | 29440 | 41713 | 41758 | 365826 | 435156 |

Table 5: The number of attempts and comparisons performed to search for the first appearance of a selected pattern from the end of the text

| Pattern length | TSW | BR | BM | KMP | BF |
|----------------|-----|----|----|-----|----|
|                | Index | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons |
| 4              | 768606 | 66   | 76  | 136118 | 151948 | 195168 | 207233 | 768607 | 816438 |
| 5              | 768468 | 94   | 102 | 112181 | 114792 | 158135 | 160057 | 768469 | 780205 |
| 6              | 768189 | 164  | 183 | 99358  | 102611 | 132797 | 134899 | 768190 | 805725 |
| 7              | 768510 | 68   | 79  | 93558  | 104611 | 116750 | 124074 | 768532 | 741589 |
| 8              | 768486 | 66   | 78  | 83653  | 93537  | 101472 | 107957 | 768485 | 795109 |
| 9              | 768595 | 44   | 55  | 77868  | 87499  | 93751  | 99992  | 768593 | 780584 |
| 10             | 768222 | 104  | 117 | 70970  | 79150  | 84280  | 89630  | 768220 | 780205 |
| 11             | 768240 | 92   | 107 | 65279  | 72965  | 77326  | 82378  | 768238 | 780224 |
| 12             | 768306 | 78   | 96  | 60905  | 67108  | 70803  | 74580  | 768304 | 780292 |

Table 6: The average number of attempts and comparisons performed to search for (100) patterns selected from the beginning of the text

| Pattern length | TSW | BR | BM | KMP | BF |
|----------------|-----|----|----|-----|----|
|                | No. of words | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons |
| 4              | 100   | 143 | 157 | 76  | 85  | 113 | 121 | 422 | 437 | 423 | 439 |
| 5              | 100   | 185 | 206 | 99  | 115 | 138 | 151 | 633 | 658 | 635 | 660 |
| 6              | 100   | 227 | 255 | 121 | 142 | 163 | 179 | 866 | 903 | 869 | 906 |
| 7              | 100   | 347 | 388 | 195 | 226 | 255 | 281 | 1550| 1621| 1555| 1627|
| 8              | 100   | 504 | 568 | 270 | 310 | 350 | 380 | 2366| 2484| 2378| 2496|
| 9              | 100   | 670 | 750 | 363 | 417 | 409 | 409 | 3493| 3652| 3511| 3671|
| 10             | 100   | 1160| 1290| 640 | 727 | 812 | 878 | 6645| 6951| 6689| 6996|
| 11             | 100   | 622 | 705 | 331 | 396 | 426 | 474 | 3698| 3822| 3710| 3834|
| 12             | 100   | 865 | 972 | 478 | 557 | 580 | 632 | 5542| 5823| 5567| 5848|

acceptable compared with BR and BM algorithms since they search the text only from left to right, while TSW algorithm searches the text from both sides, which normally increases the number of comparisons and attempts.

On the other hand, BF and KMP have the largest number of comparisons. TSW algorithm performance is observed in Table 5 where a pattern with different lengths is selected from the end of Book1. TSW algorithm finds it with minimum effort by the right to left window.

Table 6-8 show the average number of comparisons and attempts needed to search for the first, middle and last appearance of 100 words selected from

---

399
Table 7: The average number of attempts and comparisons performed to search for (100) patterns selected from the middle of the text

| Pattern length | TSW | BR | BM | KMP | BF |
|----------------|-----|----|----|-----|----|
|                | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons |
| 4              | 100   | 2732 | 2959 | 3645 | 4070 | 5451 | 5794 | 2528 | 2098 | 4034 | 21043 |
| 5              | 100   | 13956 | 15140 | 11558 | 12793 | 16057 | 16984 | 74308 | 74685 | 74524 | 76701 |
| 6              | 100   | 16682 | 18317 | 12878 | 14337 | 17545 | 18639 | 93846 | 97081 | 94112 | 97347 |
| 7              | 100   | 27267 | 30095 | 19547 | 22006 | 25732 | 27537 | 156518 | 162057 | 157247 | 162786 |
| 8              | 100   | 27830 | 30915 | 20831 | 23336 | 27043 | 28845 | 184588 | 191677 | 185458 | 192551 |
| 9              | 100   | 33929 | 37200 | 23284 | 25852 | 30154 | 32043 | 226004 | 234510 | 227028 | 235537 |
| 10             | 100   | 29676 | 32817 | 20546 | 22989 | 26300 | 28049 | 214942 | 224036 | 216065 | 225160 |
| 11             | 100   | 23195 | 24646 | 20264 | 22005 | 26377 | 27620 | 231558 | 237139 | 232474 | 238061 |
| 12             | 100   | 26806 | 30222 | 21113 | 24235 | 26818 | 29098 | 255284 | 269960 | 256314 | 270993 |

Table 8: The average number of attempts and comparisons performed to search for (100) patterns selected from the end of the text

| Pattern length | TSW | BR | BM | KMP | BF |
|----------------|-----|----|----|-----|----|
|                | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons |
| 4              | 100   | 133  | 148 | 6899 | 7719 | 10197 | 10857 | 38402 | 39254 | 38459 | 39310 |
| 5              | 100   | 271  | 297 | 12930 | 14404 | 18352 | 19484 | 83041 | 85608 | 83229 | 85795 |
| 6              | 100   | 364  | 402 | 21315 | 23957 | 28737 | 30713 | 154185 | 158658 | 154638 | 159112 |
| 7              | 100   | 402  | 447 | 22237 | 24731 | 29552 | 31372 | 180131 | 185979 | 180783 | 186631 |
| 8              | 100   | 536  | 592 | 21495 | 23841 | 28051 | 29727 | 191651 | 197606 | 192232 | 198187 |
| 9              | 100   | 776  | 859 | 24919 | 28257 | 31525 | 33891 | 240397 | 248454 | 241177 | 249235 |
| 10             | 100   | 1579 | 1756 | 31603 | 35360 | 40451 | 43195 | 333436 | 345747 | 334643 | 346956 |
| 11             | 100   | 619  | 669 | 32797 | 36438 | 41812 | 4450 | 367043 | 377922 | 368994 | 379873 |
| 12             | 100   | 1667 | 1872 | 30928 | 35067 | 38553 | 41374 | 367780 | 384834 | 369501 | 386555 |

Table 9: The number of attempts and comparisons performed to search for a set of patterns that do not exist in the text

| Pattern length | TSW | BR | BM | KMP | BF |
|----------------|-----|----|----|-----|----|
|                | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons | Attempts | Comparisons |
| 4              | 132894 | 142538 | 135240 | 151799 | 193892 | 206747 | 768769 | 777901 | 768769 | 777941 |
| 5              | 113904 | 121958 | 115988 | 129713 | 156015 | 166626 | 768768 | 777900 | 768768 | 777940 |
| 6              | 96782  | 98008 | 98075 | 98201 | 135651 | 135728 | 768767 | 777899 | 768767 | 777939 |
| 7              | 85970  | 86086 | 87081 | 87141 | 115235 | 115270 | 768766 | 777898 | 768766 | 777938 |
| 8              | 78722  | 82124 | 81500 | 86587 | 100541 | 103498 | 768765 | 777897 | 768765 | 777937 |
| 9              | 72136  | 76506 | 75625 | 82879 | 91793 | 96158 | 768764 | 777896 | 768764 | 777936 |
| 10             | 65890  | 69354 | 68904 | 74522 | 85619 | 89079 | 768763 | 777895 | 768763 | 777935 |
| 11             | 60094  | 61964 | 62235 | 64683 | 80370 | 82038 | 768762 | 777894 | 768762 | 777934 |
| 12             | 56774  | 60319 | 60911 | 67164 | 74262 | 78281 | 768761 | 777893 | 768761 | 777933 |

The results of taking 100 words are similar to that of taking a single word with different lengths. As shown in Table 8, TSW algorithm best performance is when we search for words selected from the end of Book1. In case of a complete mismatch, as in Table 9, the average number of comparisons and attempts of the TSW algorithm is the minimum; this is because the shift value in most cases reaches m+2.

CONCLUSION

In this research, we presented a fast pattern matching algorithm The Two Sliding Windows algorithm TSW which makes use of two sliding windows. It employs the main idea of BR by maximizing the shift value and using two sliding windows rather than using one sliding window to scan all text characters as in BR algorithm. The TSW uses two arrays; each array is a one dimensional array of size (m-1). The arrays are used to store the calculated shift values for the two sliding windows, while the original BR algorithm uses a two-dimensional array.

We evaluated TSW performance by using a text string and various set of patterns. Also in the algorithm, during the pre-processing phase we reduced the memory required by using one-dimensional arrays for the pattern characters only. The concept of searching the text from both sides simultaneously gives TSW algorithm a preference over other algorithms in the number of comparisons and attempts especially if the pattern searched for occurs at the end of the text. In future research, we intend to implement the TSW algorithm on real parallel processors to minimize the number of comparisons and attempts. Also we intend to...
implement the idea of the two sliding windows on other algorithms such as KMP and BM.

REFERENCES

1. Wang, Y. and H. Kobayashi, 2006. High performance pattern matching algorithm for network security. IJCSNS, 6: 83- 87. URL:http://paper.ijcsns.org/07_book/200610/200610A3.pdf

2. Navarro, G. and M. Raffinot, 2002. Flexible Pattern Matching in Strings-Practical On-line Search Algorithms for Texts and Biological Sequences. First Edition. Cambridge University Press, New York. ISBN: 0-521-81307-7

3. Crochemore, M. and W. Rytter, 2002. Jewels of Stringology. First Edition. WorldScientific, Singapore. ISBN: 9789810247829

4. Smyth, W.F., 2003. Computing Patterns in Strings. First Edition. Pearson Addison Wesley. United States. ISBN: 978-0-201-39839-7

5. Charras, C. and T. Lecroq, 2004. Handbook of Exact String Matching Algorithms. First Edition. King’s College London Publications. ISBN: 0954300645

6. Knuth, D.E., J.H. Morris and V.R. Pratt, 1977. Fast pattern matching in strings. SIAM J. Comput., 6: 323-350.

7. Horspool, R.N., 1980. Practical fast searching in strings. Software Practice Experience, 10: 501-506.

8. Berry, T. and S. Ravindran, 1999. A fast string matching algorithm and experimental results. In: Proceedings of the Prague Stringology Club Workshop ’99, Liverpool John Moores University, pp: 16-28.

9. Crochemore, M. and D. Perrin, 1991. Two-way string-matching. ACM, 38: 651-675. DOI: http://doi.acm.org/10.1145/116825.116845

10. Thathoo, R. et al., 2006. TVSBS: A fast exact pattern matching algorithm for biological sequences. Current Sci., 91: 47-53. URL: http://profile.iit.ac.in/avirmani_02/TVSBS.pdf

11. Boyer, R.S. and J.S. Moore, 1977. A fast string searching algorithm. Commun. ACM., 20: 762-772. DOI:10.1145/359842.359859

12. Michael, T.G. and Roberto Tamassia, 2002. Algorithm Design, Foundations, Analysis and Internet Examples. First Edition. John Wiley and Sons, Inc, USA. ISBN: 0-471-38365-1

13. He, L., F. Binxing and J. Sui, 2005. The wide window string matching algorithm. Theor. Comput. Sci., 332: 391-404. DOI: 10.1016/j.tcs.2004.12.002

14. Hume, A. and D. Sunday, 1991. Fast string searching. Software Practice Experience, 21: 1221-1248. DOI:10.1002/spe.4380211105

15. Lecroq, T., 1995. Experimental results on string matching algorithms. Software Practice and Experience, 25: 727-765. DOI: 10.1002/spe.4380250703

16. Davies G., and Bowsher S., 1996. Algorithms for pattern matching, Software-Practice and Experience,16:575-601. DOI:10.1002/spe.4380160608

17. Tsai, T.H., 2003. Average case analysis of the boyer-moore algorithm. Source. Random Struct. Algorithms, 28: 481-498. DOI: 10.1002/rsa.v28:4

18. Frantisek F., Christopher G. Jennings and W. F. Smyth, 2007. A simple fast hybrid pattern-matching algorithm, J. Discrete Algorithms 5: 682–695. DOI: 10.1016/j.jda.2006.11.004

19. T. Bell, J. Cleary, and I. Witten, Text Compression. First Edition. Englewood Cliffs, N.J.: Prentice Hall, 1990. ISBN: 0-13-911991-4