Int. j. inf. tecnol.
https://doi.org/10.1007/s41870-019-00371-1

ORIGINAL RESEARCH

BFM: a forward backward string matching algorithm with improved shifting for information retrieval

MD. Obaidullah Al-Faruk[1] · K. M. Akib Hussain[1] · MD. Adnan Shahriar[1] · Shakila Mahjabin Tonni[1]

Received: 2 May 2018 / Accepted: 9 October 2019

Abstract Mining data from text is often becomes a crucial part of data mining tasks. With the growing tendency of using cloud and sharing more and more files over the internet, the necessity of applying a string matching algorithm in text mining has increased rapidly in present time. These algorithms need to make less character comparisons and pattern shifts while searching. In this paper, we're proposing a new algorithm named *Back and Forth Matching* (*BFM*) algorithm that works faster by matching a pattern from both the forward and backward direction. It shows a tremendous improvement, in comparison to other algorithms, while matching strings in large text files.

Keywords Data mining · Text search · Text mining · String matching · Exact pattern matching · Knuth–Morris–Pratt · Boyer–Moore

1 Introduction

As more and more data and files are being used and need to be processed on real time over the internet, at present, processing huge text files to retrieve information from it has become an imperative data mining task [1]. To find a desired text (pattern) in a text document is a straightforward task, but may become very much time consuming one, with the increase in file size. Because of this reason, a fast pattern matching algorithm is an indispensable part of page ranking in search engines and digital libraries, checking syntax and spelling mistakes, detecting network breach, and in many other applications [2]. It is also essential for bioinformatics, DNA sequences matching, and behavior analysis as well [3, 4].

String matching algorithms tries to find all the occurrences of pattern P of length m in the text T that has n characters. To search through the text, the pattern is used as a window of length m that moves over the text T starting from it's leftmost character. The target of the algorithm is to match the characters of the window with the text characters in multiple attempts [5]. This succession of attempts and window shifts continues, until the right end of the window reaches the length of the text. This type of window mechanism is known as the sliding window mechanism [6]. In most cases, the string matching algorithms have two phases- the preprocessing phase and the searching phase [1]. In the preprocessing phase, generally the pattern is preprocessed to form a table that determines at what position the pattern needs to be shifted in finding a mismatch. Then in the searching phase comparisons are made between pattern and text characters from right to left, or left to right, or in specific ways to find out all the occurrences of exact pattern match.

The main challenges of these algorithms are to minimize the character comparisons and to maximize the length of shifts [1, 5]. In this paper, we've presented a new string-matching algorithm BFM to find all the occurrences of exact patterns or a string in a text. The presented algorithm shows much less character comparisons and more shift

✉ MD. Obaidullah Al-Faruk
obaidullah.faruk05@gmail.com

K. M. Akib Hussain
akibhussain79@gmail.com

MD. Adnan Shahriar
radnan046@gmail.com

Shakila Mahjabin Tonni
shakila@ewubd.edu

[1] Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh

Published online: 31 October 2019

lengths compared to other algorithms. More importantly, the algorithm does both forward and backward checking and managed by a preprocessing table that decreases the number of efforts required to match the window with the text during the matching phase.

We have organized rest of the paper as following: in Sect. 2, we have given a review of the on the available literatures. In Sect. 3, we have discussed the working procedure of existing string matching algorithms. Section 4 introduces our proposed algorithm BFM and the result analysis of the algorithm is discussed in Sect. 5. And finally, in conclusion at Sect. 6, we included the future prospects of our algorithm.

2 Related work

The Boyer Moore algorithm is one of the most renowned, efficient and extensively used pattern matching algorithm [1]. It preprocesses the pattern and determines the maximum shifts of the pattern in case of a mismatch based on two heuristics—the good and the bad heuristics. These heuristics works independently over the pattern to produce two different arrays known as the preprocessing table [7]. During the searching phase at each mismatch, Boyer Moore determines best of the two heuristics and thus can slide the pattern by maximum number of characters. [8–10] Describes another way to improve the algorithm by match shifting. Another popular algorithm is the Quick Search algorithm. Though, according to [11] this algorithm is more appropriate when applied on a large character set to find a small pattern. Quick-Skip Search algorithm [12] proposed a combination of Quick Search and the Skip Search algorithm. Again, in [3] another hybrid algorithm was proposed using the idea of Quick-Skip and Boyer–Moore algorithms.

Another highly used algorithm is the Knuth–Morris–Pratt (KMP) algorithm that works like a naive algorithm, but uses the degenerative property of the searched string [13]. That is, instead of checking all the characters after each shift, the algorithm preprocesses P to construct a table that helps skipping comparisons of those pattern characters that had already matched with the window characters. A parallel string matching algorithm based on this work is proposed in [14]. [15] Proposes a new algorithm combining the idea of Boyer–Moore and KMP algorithm.

Beside all these algorithms, a comperatively old string matching technique is Rabin Carp string searching algorithm [16] that can search both single and multiple patterns in a string. A more recent enhancement of this algorithm is proposed in [17], that deploys a modified version of the Rabin Carp algorithm using a GPU processor.

3 Exact string matching algorithms

The target of an exact string matching algorithm is to find if a pattern matches exactly with parts of a given text. The algorithm finds all the occurrences and positions where the pattern appears. The overall mechanism is illustrated in Fig. 1

The KMP algorithm [15, 18–20] forms a window over a text T and scans from left to right in order to see whether the successive characters of the current window match with the pattern P. We consider that the pattern length is m, and the text of length n.

Whenever checking for the first character of the pattern $P[0]$, if there is mismatch then the pattern is shifted by one character. For other characters in P, if there is a mismatch between $T[i]$ and $P[j]$, then it shifts the pattern using the preprocessing table. In this way it continues, in order to find all occurrences of P in T. To illustrate the algorithm, let us consider text T and pattern P as follows:

T : GCATGCAG

P : GCAG

The constructed preprocessing table based on the pattern and the steps of searching the pattern are depicted in Figs. 2 and 3 respectively. Total number of shifts involved in this task is 10.

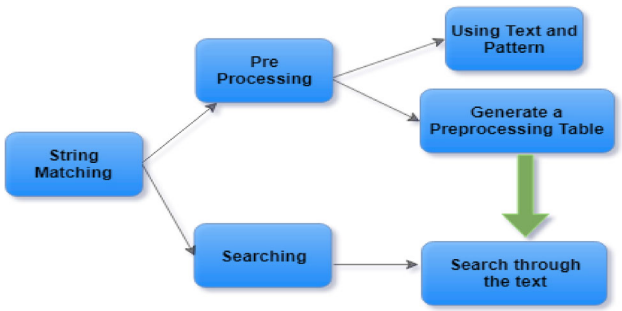


Fig. 1 Exact string matching algorithm mechanism

G	C	A	G
0	0	0	1

Fig. 2 Preprocessing table for KMP

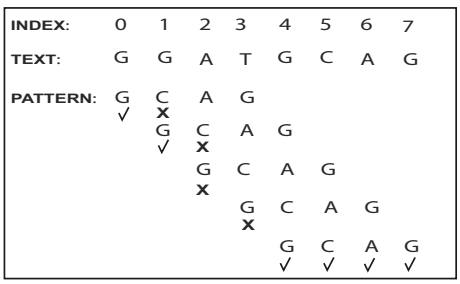


Fig. 3 Searching pattern P in T using KMP

G	C	A	*
3	2	1	4

Fig. 4 Preprocessing table for Boyer Moore Hoorspool

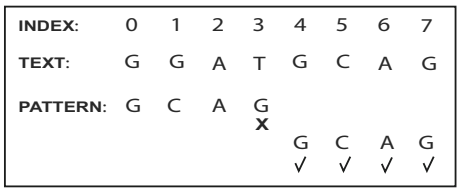


Fig. 5 Searching in Boyer Moor Horspool

The Boyer Moore Horspool (BMH) algorithm is a simpler and an improved form of the Boyer Moore algorithm [21]. It only uses the bad character heuristics of the Boyer Moore algorithm ignoring the good suffix heuristics, as its practical implementation is difficult and more complex [1, 2]. To illustrate the process of string matching using BMH algorithm we again consider searching previously assumed pattern P in text T.

The preprocessing table constructed for P is illustrated in Fig. 4. The searching steps are shown in Fig. 5. After completing the search, total number of shift is 1.

4 Proposed algorithm

The Back and Forth Matching (BFM) algorithm is developed targeting larger shifts of pattern window for each mismatch, thus minimizing the total number of shifts while matching strings. Also, this will ensure less character comparison. To serve the purpose, the algorithm compares characters from both left and right side in each attempt while searching. As the first step of the algorithm, a preprocessing table is prepared that stores the positions in the text, where the first and last characters of the pattern matches with the text. Algorithm 1 depicts the preprocessing phase of BFM.

Algorithm 1 *Preprocessing(T, P)*

```
1.  i ← 0
2.  while i ≤ (n − m) do
3.      if T[i]==P[0] then
4.          if T[i + m-1] == P[m − 1] then
5.              posIndex[] ← i
6.          end if
7.      end if
8.      i ← i + 1
9.  end while
10. Searching(T, P, m, n, posIndex[])
```

To search a pattern in search phase, the algorithm checks for the pattern only at the positions stored in the preprocessing table. The whole process is illustrated in algorithm 2.

Algorithm 2 *Searching(T, P, m, n, posIndex[])*

```
1.  length ← len(posIndex[])
2.  for i = 0 to length − 1 do
3.      k ← posIndex[i]
4.      s ← 0
5.      txtlast ←m + k − 1
6.      patlast ←m−1
7.      while k ≤ txtlast do
8.          if T[k+1]==P[s+1] and T[txtlast−1]==P[patlast−1] then
9.              k ← k + 1, s ← s + 1
10.             txtlast ← txtlast − 1, patlast ← patlast − 1,
11.         else
12.             break
13.         end if
14.     end while
15.     if all character matched then
16.         Pattern found
17.     end if
18.     posIndex[] ←next posIndex[]
19.     i ← i + 1
20. end for
```

To demonstrate the mechanism of BFM consider the same text and pattern we used to describe KMP and BMH algorithm. In the preprocessing phase, BFM algorithm finds out all the possible positions in the text, where the first and last characters of the pattern are matched with the text. If a match is found, *posIndex* stores the position. This preprocessing phase is depicted in Fig. 6.

According to *posIndex* values determined in the preprocessing phase, the algorithm searches characters after position $T[1]$ and $T[4]$. So, at first by forward check, BFM checks if the second character from the left of the pattern is matched with the text or not. At the same time by doing backward check, it checks if the second character from the right is also matched or not. It counts it as a total match if all the characters between the first and last character of the pattern matches with the characters between $T[1]$ and $T[4]$.

As the first and last character was checked before, they are not checked during this phase. Following this procedure, BFM finds only one match starting at the position

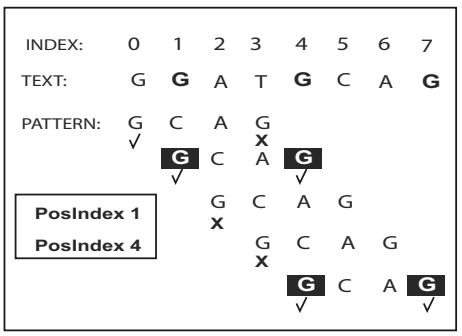


Fig. 6 Preprocessing in BFM

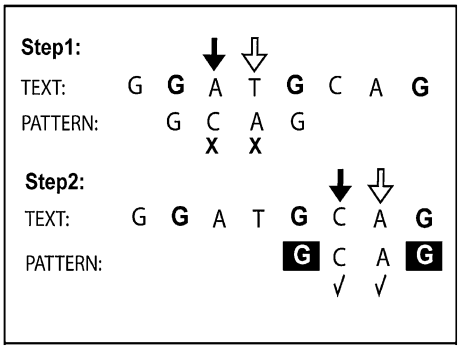


Fig. 7 Searching in BFM

$T[4]$. An illustration of the process is displayed in Fig. 7. Total number of shifts performed is 1.

Thus, for a pattern $P[1..m]$ in a text string $T[1..n]$, both build over a finite set of character or alphabet, our proposed algorithm BFM will result in preprocessing time complexity of O(n) and searching time complexity of O(mn), where $n \geq m$.

5 Result analysis

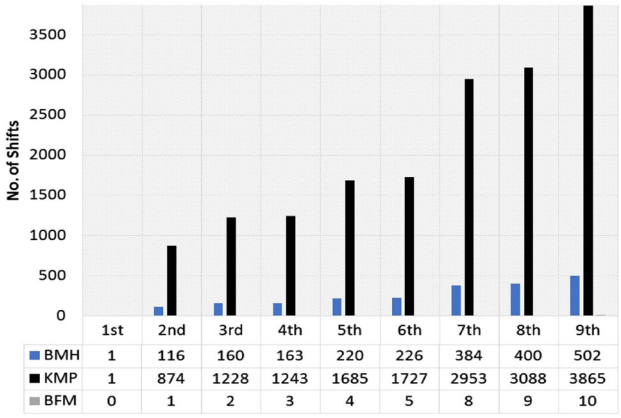
To evaluate the performance of BFM, we've used the large corpus available at the Canterbury Corpus [22]. We've used the "World192.txt" file with 1,905,891 characters and the "bible.txt" file with 3,250,898 characters files for our purpose. Along with our algorithm BFM, we evaluated both BMH and KMP algorithms to compare our algorithm's performance on both files.

The three string matching algorithms are run using a personal computer having Intel(R) core(TM) i5-3230M CPU @ 2.6 GHz, x-64-based processor, with a DDR3 8.00 GB RAM(7.90 GB usable) and 1TB SATA HDD storage. The whole test was conducted in a 64-bit Windows 10 operating system.

As in many previous works, the total number of comparisons and shifts needed to find a pattern is considered as the measures of an algorithm's quality [15], we are using these two criteria to evaluate our algorithm. The first evaluation was done on "World192.txt" file, to find the pattern "Bangladesh". This pattern appears 9 times in file. Total shifts and comparisons required for the task is displayed in Figs. 8 and 9 respectively. The execution time taken to accomplish the search is given in Fig. 10.

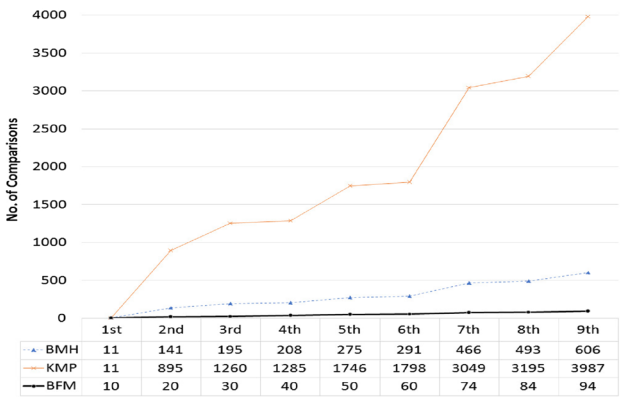
From these result it can be said that the number of shifts, comparison and execution time of our algorithm BFM are very much lower than BMH and KMP.

Given below is the chart in Table 1, showing the time complexity [2] of BMH and KMP, along with our proposed algorithm.



	1st	2nd	3rd	4th	5th	6th	7th	8th	9th
BMH	1	116	160	163	220	226	384	400	502
KMP	1	874	1228	1243	1685	1727	2953	3088	3865
BFM	0	1	2	3	4	5	8	9	10

Fig. 8 Total number of shifts to search "Bangladesh"



	1st	2nd	3rd	4th	5th	6th	7th	8th	9th
BMH	11	141	195	208	275	291	466	493	606
KMP	11	895	1260	1285	1746	1798	3049	3195	3987
BFM	10	20	30	40	50	60	74	84	94

Fig. 9 Total number of comparisons made to search "Bangladesh"

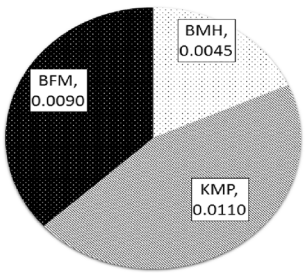


Fig. 10 Execution time (s) required to search "Bangladesh"

To evaluate BFM algorithms perform with respect to the file size, we applied all three algorithms on the "bible.txt" file. On this file we search for the word "God" as this word has preferably many occurrences throughout the file. First of all we considered the total character size of the file and then we decrease the character size of the search text and perform the search again on it. The result of the process is displayed in Table 2. This analysis also shows our proposed algorithm is performing better than the other algorithms.

Table 1 Time complexity analysis for KMP, BMH and BFM

Algorithm	Process time complexity	Search time complexity
Knuth Morris Pratt algorithm (KMP)	O(m)	O(m+n)
Boyer Moore Hoorspool algorithm (BMH)	O(m+n)	O(mn)
Back and Forth algorithm (BFM)	O(n)	O(mn)

Table 2 Performance analysis on different character sized text

Char size	3250898	1577526	677382	416172	242632
No. of shifts					
BFM	8694136	8694136	1759121	552104	195329
BMH	3.1E+09	3.1E+09	2.4E+09	9.8E+07	3.2E+07
KMP	8.9E+09	8.9E+09	2.4E+09	2.8E+08	9.2E+07
No. of comparisons					
BFM	3.4E+07	3.4E+07	6946292	2211962	783158
BMH	3.3E+09	3.3E+09	2.4E+09	1E+08	3.4E+07
KMP	8.9E+09	8.9E+09	2.4E+09	2.8E+08	9.3E+07
Execution time (s)					
BFM	0.0625	0.0313	0.0313	0.0313	0.0156
BMH	1.0157	2.1721	0.8595	0.2031	0.0938
KMP	1.9221	1.9846	1.2189	0.1719	0.1094

6 Conclusion

The Back and Forth Matching (BFM) algorithm is preprocessing a text beforehand by applying forward and backward matching to match both the first and last characters of the pattern. This preprocessing task enhances the searching by enabling the algorithm to search only on the indexed positions, thus decreasing both the number of shifting and character comparisons. One limitation of this algorithm is if the text and pattern both are small, then BFM's performance degrades, as there is a preprocessing phase to complete before searching. Nevertheless, we found the performance of BFM in respect to execution time, number of shifts and comparisons to be exceptionally high in contrast to KMP and BMH. Hence, this algorithm would certainly an efficient choice in case of the tasks that require huge amount of text searches.

References

1. Gurung Dipendra, Chakraborty Udit Kr, Sharma Pratikshya (2016) Intelligent predictive string search algorithm. Proc Comput Sci 79:161–169
2. Singla Nimisha, Garg Deepak (2012) String matching algorithms and their applicability in various applications. Int J Soft Comput Eng 1(6):218–222
3. Mahmood Al-Dabbagh SS, Sahib Naser MA, Barnouti NH (2017) Fast hybrid string matching algorithm based on the quick-skip and tuned Boyer-Moore algorithms. Int J Adv Comput Sci Appl 8(6):117–127
4. Raju S Viswanadha, Babu A Vinaya, Mrudula M (2006) Back-end engine for parallel string matching using boolean matrix. In: International symposium on parallel computing in electrical engineering, 2006. PAR ELEC 2006, pp 281–283. IEEE
5. Charras C, Lecrog T, Pehoushek JD (1998) A very fast string matching algorithm for small alphabets and long patterns. In: Annual symposium on combinatorial pattern matching, Springer, New York, pp 55–64
6. Rasool Akhtar, Khare Nilay, Arora Himanshu, Varshney Amit, Kumar Gaurav (2012) Multithreaded implementation of hybrid string matching algorithm. Int J Comput Sci Eng 4(3):438
7. Bhandari Jamuna (2014) String matching rules used by variants of Boyer-Moore algorithm. J Global Res Comput Sci 5(1):8–11
8. Mahmood SS, Dabbagh A, Barnouti NH (2017) A new efficient hybrid string matching algorithm to solve the exact string matching problem. Br J Math Comput Sci 20(2):1–14
9. Tarhio Jorma, Ukkonen Esko (1993) Approximate Boyer-Moore string matching. SIAM J Comput 22(2):243–260
10. Sahli Mohammed, Shibuya Tetsuo (2012) Max-shift bm and max-shift horspool: practical fast exact string matching algorithms. J Inf Process 20(2):419–425
11. Klaib Ahmad Fadel, Zainol Zurinahni, Ahamed Nurul Hashimah, Ahmad Rosma, Hussin Wahidah (2007) Application of exact string matching algorithms towards smiles representation of chemical structure. Int J Comput Inf Sci Eng 1:235–239
12. Naser Mustafa Abdul Sahib, Aboalmaaly Mohammed Faiz et al (2012) Quick-skip search hybrid algorithm for the exact string matching problem. Int J Comput Theory Eng 4(2):259
13. Al-Khamaiseh Koloud, ALShagarin Shadi (2014) A survey of string matching algorithms. Int J Eng Res Appl 4(7):144–156
14. Rao CS, Raju KB, Raju SV (2013) Parallel string matching with multi-core processors-a comparative study for gene sequences. Global J Comput Sci Technol 13(1):26–41
15. Tsarev RY, Chernigovskiy AS, Tsareva EA, Brezitskaya VV, Nikiforov AY, Smirnov NA (2016) Combined string searching algorithm based on Knuth-Morris-Pratt and Boyer-Moore algorithms. In: IOP conference series: materials science and engineering, vol 122, IOP Publishing, p 012034
16. Karp Richard M, Rabin Michael O (1987) Efficient randomized pattern-matching algorithms. IBM J Res Dev 31(2):249–260
17. Shah P, Oza R (2017) Improved parallel Rabin-Karp algorithm using compute unified device architecture. In: International conference on information and communication technology for intelligent systems, Springer, New York, pp 236–244
18. Rahim Robbi, Zulkarnain Iskandar, Jaya Hendra (2017) A review: search visualization with Knuth Morris Pratt algorithm. In: IOP conference series: materials science and engineering, vol 237, IOP Publishing, p 012026
19. Abu-Zaid IM, El-Rayyes EK (2012) Parallel search using kmp algorithm in Arabic string. Int J Sci Technol 2(7):427–431
20. Janani R, Vijayarani S (2016) An efficient text pattern matching algorithm for retrieving information from desktop. Indian J Sci Technol 9(43):1
21. Raita Timo (1992) Tuning the Boyer-Moore-Horspool string searching algorithm. Softw Pract Exp 22(10):879–884
22. Powell M (2007) The canterbury corpus. http://corpus.canterbury.ac.nz/descriptions. Accessed 15 Feb 2018