Multiple Skip Multiple Pattern Matching Algorithm (MSMPMA)

Ziad A.A. Alqadi[1], Musbah Aqel[2], & Ibrahiem M. M. El Emary[3]
1 Faculty of Engineering, Al Balqa Applied University, Amman, Jordan
E-mail:ntalia@yahoo.com
2 Faculty of Engineering, Applied Science University, Amman, Jordan
E-mail: musbahaqel@yahoo.com
3 Faculty of Engineering, Al Ahliyya Amman University, Amman, Jordan
E-mail: doctor_ebrahim@yahoo.com

***Abstract*-- A new algorithm to search for multiple patterns simultaneously is proposed. The multiple pattern algorithms can be used in many applications that require such type of search and matching. For example, a multi-pattern matching can be used in lieu of indexing or sorting data in some applications that involve small to medium size datasets. One of its advantages is that no additional search structure is needed and no preprocessing phase is required. The proposed algorithm is simple and can suit for multiple patterns matching in a file with unlimited size. The time complexity of the algorithm is O (n*m), but because of the skips it moves to around O (n).**

The number of comparisons rapidly decreased after the first match, and for multiple matching, it will be little greater than n (file size). The algorithm was implemented and compared with some popular multi-pattern matching algorithms and it has shown more enhancement in performance and faster than others.

***Index Terms*-- DNA, MSMPMA, String matching algorithms, and CPC**

I. Introduction

The multi-pattern matching technique can be used in many applications. It is used in data filtering or what is called data mining, to find selected patterns, for example, from a stream of news feed, also, it is used in security applications to detect certain suspicious keywords and it can be used in searching for patterns that can have several forms such as dates. However, it is used in glimpse [10] to support Boolean queries by searching for all terms at the same time and then intersecting the results; and it is used in DNA searching by translating an approximate search to a search for a large number of exact patterns [2].

String matching algorithms also used in Intrusion Detection Systems (IDSs) that have become widely recognized as powerful tools for identifying, deterring and deflecting malicious attacks over the network. Essential to almost every intrusion detection system is the ability to search through packets and identify content that matches known attacks. Space and time efficient string matching algorithms are therefore important for identifying these packets at line rate. [13]. besides there are, many other applications, that can be find in [9, 10, 11, 12].

Aho and Corasick [1] presented a linear-time algorithm for this problem, based on automata approach. This algorithm serves as the basis for the UNIX tool fgrep. A linear-time algorithm is optimal in the worst case, but as the regular string-searching algorithm by Boyer and Moore [4] demonstrated, it is possible to actually skip a large portion of the text while searching, leading to faster than linear algorithms in the average case. Commentz-Walter [5] presented an algorithm for the multi-pattern matching problem that combines the Boyer-Moore technique with the Aho-Corasick algorithm. The Commentz-Walter algorithm is substantially faster than the Aho-Corasick algorithm in practice. Hume [8] designed a tool called gre based on this algorithm, and version 2.0 of fgrep by the GNU project [6] is using it. Baeza-Yates [3] also gave an algorithm that combines the Boyer-Moore-Horspool algorithm [12] (which is a slight variation of the classical Boyer-Moore algorithm) with the Aho-Corasick algorithm.

II. STRING MATCHING ALGORITHMS

String searching algorithms are an important class of string algorithms that try to find a place where one or several strings (i.e. patterns) are found within a larger string or text. Let Σ be an alphabet (finite set). Formally, both the pattern and searched text are concatenation of elements of Σ. The Σ may be usual human alphabet (A-Z). Other applications may use binary alphabet (Σ = {0, 1}) or DNA alphabet (Σ = {A, C, G, T}) in bioinformatics. Table 1 summarizes the most popular algorithms used for single and multiple pattern matching. Where m is the pattern length and n is the file size. The first thing worth noting is that the relevant body of literature for this problem is the multi-pattern string matching problem, which is somewhat different from the single pattern string matching solutions that many people are familiar with such as Boyer-Moore [14]. For single-pattern string matching, there is a large body of work in which a single string is to be searched for in the text. This is processing used in word processing applications, e.g., in search and-replace operations.

Table 1: String matching algorithm summary

Algorithm	Preprocessing time	Complicity matching time
Naïve string search algorithm	0 (no preprocessing)	O((n-m+1) m)
Trie-matching	0 (no preprocessing)	*O (m + #pat · n)*
Rabin-Karp string search algorithm	θ(m)	O((n-m+1) m)
Finite automata	O(m |Σ|)	θ(n)
Knuth-Morris-Pratt algorithm	θ(m)	θ(n)
Boyer-Moore string search algorithm	O(m)	average O(n/m), worst O(n m)

On the other hand, the multi-pattern string matching problem searches a body of text (in our case an application file such as DNA sequence or a text file regardless it's size) for a *set* of strings (patterns). One can trivially extend a single pattern string matching algorithm to be a multiple pattern string matching algorithm by applying the single pattern algorithm to the search text for each search pattern. Obviously this does not scale well to larger sets of strings to be matched. Instead, multi-pattern string matching algorithms generally preprocess the set of input strings, and then search all of them together over the body of text. Previous work in precise multi-pattern string matching includes Aho-orasick [15], Commentz-Walter[16], Wu-Manber [17], and others.

There has also been even more recent work in imprecise string matching algorithms using hashing and signature based techniques [18], [19]. Although these methods may meet the criteria of having deterministic execution time text, there is a problem of positive matches that must be revivified using a precise string matching algorithm. Thus, the performance of the underlying precise matching algorithm is still important, albeit at a reduced level.

The simplest and least efficient way to see where one string occurs inside another is to check each place that it may contain, one by one, to see if it's there[17]. So, first it should be seen if there's a copy of the pattern in the first few characters of the file; if not, we look to see if there's a copy of the pattern starting at the second character of the file; if not, we look starting at the third character, and so forth. In the normal case, we only have to look at one or two characters for each wrong position to see that it is a wrong position, so in the average case, this takes O($n + m$) steps, where n is the length (size)of the file and m is the length of the pattern; but in the worst case, searching for a string like "aaaab" in a string like "aaaaaaaaab", it takes O(nm) steps.

Tries offer text searches with costs which are independent of the size of the file being searched, and so are important for large files requiring spelling checkers, case insensitivity, and limited approximate regular secondary storage. The cost of the trie-matching is independent of file size [17, 18].

Imprecise string matching also introduces the possibility that certain innocent data streams may introduce a rate of sequential false positives that overwhelm the exact matching algorithm unless it is capable of processing at line rate. We do not address the open question of whether imprecise methods are completely appropriate for use in situations where worst-case performance is an important metric, but assert that in any case the underlying precise multi-pattern string matcher performance is still important.

As mentioned above, multiple patterns matching algorithms are now used in variant application, so we will focus on two mostly used algorithms, which have good performance in string matching to find the number of occurrences of a certain pattern within a certain file, the Brute-Force and the trie-matching algorithms.

A multiple skip multiple pattern matching algorithm is proposed based on Boyer - Moore ideas. The algorithm is implemented and compared with Brute-Force, and Trie algorithms.

III. The Proposed Algorithm

The MSMPMA algorithm scans the input file to find all occurrences of a pattern within this file, based on skip techniques, and can be described as follows:

- Fix the file index in a cretin position.
- Use this position as a starting point of matching
- Compare the file contents from the defined point with pattern contents.
- Find the skip value depending on the match number(ranges from 1 to m-1)
- Perform the above sequence while the file position dose not reaches n-m.

A .Description of the algorithm:

The proposed algorithm MSMPMA assumes that their is input text file (T) that has size (n) and their is a pattern (P) with size (m) so the algorithm proceeds as follows:

```
1. Input text T of size n and pattern (P) of size (m)
2. Output starting index of all substring occurrences of (T) that is equal to (P) and output (-1) if no such substring exists
3. Initialization: skip=1, index i of T=1 , and number of occurrence = 0
4. Check index, if index <= n-m then proceed to step 5, else go to step 12
5. Set index j of P to 1, and save i if (k=i)
6. Check j. If j<=m go to step 7, else go to step 8
7. Compare T(k) and P(j) . If they are equal increment k and j and go to step 6
8. Skip if skip=j
9. Increment number of occurrences
10 add skip to i
11 go to step 4
12 return number of occurrences
```

IV. MSMPMA Implementation

The algorithm was implemented using object-oriented programming with C++, and it was tested using different DNA sequence with different file sizes. However, the proposed algorithm is compared with other three algorithms. They are Brute-Force, Trie, and Naïve string search algorithm. These algorithms are selected due to common features with the proposed algorithm as follows:

- Multiple string matching
- No preprocessing operations(and thus no preprocessing time)
- Maintaining different type of files (in contents and sizes).

The implementation and comparison with other algorithms process is carried out When text file size = 1024 bytes, using different patterns and sizes in implementation process. The results are obtained and shown in the following tables.

```
AGAACGCAGAGACAAGGTTCTCATTG
TGTCTCGCAATAGTGTTACCAACTCGG
GTGCCTATTGGCCTCCAAAAAAGGCT
GTTCAACGCTCCAAGCTCGTGACCTC
GTCACTACGACGGCGAGTAAGAACGC
CGAGAAGGTAAGGGAACTAATGACGC
```

GTGGTGAATCCTATGGGTTAGGATCGT
GTCTACCCCAAATTCTTAATAAAAAAC
CTAGGACCCCCTTCGACCTAGACTATC
GTATTATGGACAAGCTTTAACTGTCGT
ACTGTGGAGGCTTCAAAACGGAGGGA
CCAAAAAATTTGCTTCTAGCGTCAATG
AAAAGAAGTCGGGTGTATGCCCCAATT
CCTTGCTGCCCGGACGGCCAGGCTTA
TGTACAATCCACGCGGTACTACATCTT
GTCTCTTATGTAGGGTTCAGTTCTTCG
CGCAATCATAGCGGTACTTCATAATGG
GACACAACGAATCGCGGCCGGATATC
ACATCTGCTCCTGTGATGGAATTGCTG
AATGCGCAGGTGTGAATACTGCGGCT
CCATTCGTTTTGCCGTGTTGATCGGGA
ATGCACCTCGGGGACTGTTCGATACG
ACCTGGGATTTGGCTATACTCCATTCC
TCGCGAGTTTTCGATTGCTCATTAGGC
TTTGCGGTAAGTAAGTTCTGGCCACCC
ACTTCGAGAAGTGAATGGCTGGCTCC
TGAGCGCGTCCTCCGTACAATGAAGA
CCGGTCTCGCGCTAAATTTCCCCCAG
CTTGTACAATAGTCCAGTTTATTATCAA
AGATGCGACAAATAAATTGATCAGCAT
AATCGAAGATTGCGGAGCATAAGTTTG
GAAAACTGGGAGGTTGCCAGAAAACT
CCGCGCCTACTTTCGTCAGGATGATTA
AGAGTATCGAGGCCCCGCCGTCAATA
CCGATGTTCTTCGAGCGAATAAGTACT
GCTATTTTGCAGACCCTTTGCCAGGCC
TTGTCTAAAGGTATGTTACTTAATATTG
ACAATACATGCGTATGGCCTTTTCCGG
TTAACTCCCTG

Table (2-a): pattern=A (m=1)

Algorithm	Number of occurrences	Number of comparisons	Comparisons per character
MSMPMA	259	1024	1
Brute-Force	259	1024	1
Trie-matching	259	1025	1.001
Naïve String Search Algorithm	259	1024	1

Table (2-b): patter=AG (m=2)

Algorithm	Number of occurrences	Number of comparisons	Comparisons per character
MSMPMA	53	1230	1.201
Brute-Force	53	1282	1.252
Trie-matching	53	1284	1.254
Naïve String Search Algorithm	53	1281	1.250

Table (2-c): pattern=CAT (m=3)

Algorithm	Number of occurrences	Number of comparisons	Comparisons per character
MSMPMA	11	1298	1.268
Brute-Force	11	1318	1.287
Trie-matching	11	1321	1.290
Naïve String Search Algorithm	11	1310	1.279

Table (2-d): pattern=AACG (m=4)

Algorithm	Number of occurrences	Number of comparisons	Comparisons per character
MSMPMA	5	1359	1.327
Brute-Force	5	1376	1.344
Trie-matching	5	1380	1.348
Naïve String Search Algorithm	5	1376	1.340

Table (2-e): pattern=AAGAA (m=5)

Algorithm	Number of occurrences	Number of comparisons	Comparisons per character
MSMPMA	2	1375	1.343
Brute-Force	2	1388	1.355
Trie-matching	2	1393	1.360
Naïve String Search Algorithm	2	1387	1.354

Table (2-f): pattern=AAAAAAGG (m=8)

Algorithm	Number of occurrences	Number of comparisons	Comparisons per character
MSMPMA	1	1394	1.365
Brute-Force	1	1409	1.376
Trie-matching	1	1417	1.384
Naïve String Search Algorithm	1	1407	1.374

Table (2-g): pattern=TTCTTAATAAAA (m=12)

Algorithm	Number of occurrences	Number of comparisons	Comparisons per character
MSMPMA	1	1390	1.356
Brute-Force	1	1390	1.356
Trie-matching	1	1402	1.369
Naïve String Search Algorithm	1	1399	1.366

Table (2-h): pattern=GGCTGTTCAACGCTCC (m=16)

Algorithm	Number of occurrences	Number of comparisons	Comparisons per character
MSMPMA	1	1349	1.317
Brute-Force	1	1349	1.317
Trie-matching	1	1365	1.333
Naïve String Search Algorithm	1	1349	1.317

V. Results Analysis

After the implementation of the proposed algorithm, the following points could be concluded from the obtained results in table (3) as follows:

- The number of comparisons per character(CPC) which is equal to:
(Number of comparisons/file size) can be used as a measurement factor, this factor affects the complexity time, and when it is decreased the complicity also decreased.
- From the above results we can see that CPC is always around 1, which means that the complexity depends only at the file size. (O (n)).
- If we take in consideration the number of matching (occurrences), we can say that the complicity is less than O (n), since we need less number of comparisons for the second match and less for the third and so on.
- For small, medium and large files complexity remains without changing and still depends on file size.
- The pattern length does not affect the complexity.
- The pattern length and the multiple patterns matching do not negatively affect the algorithm performance.
- The proposed algorithm can suit any type of files with any size.

However, from the comparison in table (3), between the proposed algorithm and the other three most common algorithms(i.e. Brute-force, Trie-matching, and Naive string), it is clear that the proposed algorithm has shown a good improvement in enhancement that is less number of comparisons and less value of CPC.

The proposed algorithm can be described as quite simple in description and in implementation with following main features:

- Good time complexity
- Unlimited size of the pattern.
- Unlimited size of the text file.
- Multiple patterns matching (finding all the occurrences of the pattern in the text file).
- Multiple skip technique in the matching process.
- The number of comparisons which affects the processing time rapidly decreased after the first match, and the total number of comparisons for all occurrences will be around n(text file size).
- It can be used in different ranges of applications such as: text editors, DNA matching, computer viruses detection, Noise detection (in communication systems).

VI. Conclusion

A new algorithm to search for multiple patterns simultaneously is proposed. The proposed MSMPMA algorithm proves some performance enhancements compared with Brute-Force, Trie-matching, Naïve string algorithms. These enhancements were measured by CPC, and the testing results have shown that MSMPMA algorithm has the minimum value of CPC and less number of comparisons.

References

[1] Aho, A. V., and M. J. Corasick, ‘‘Efficient string matching: an aid to bibliographic Search,’’ Communications of the ACM **18** (June 1975), pp. 333 340.

[2] Altschul S. F., W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, ‘‘Basic local alignment search tool,’’ J. Molecular Biology **15** (1990), pp. 403 410.

[3] Baeza-Yates R. A., ‘‘Improved string searching,’’ Software — Practice and Experience 19 (1989), pp. 257 271 .

[4 Boyer R. S., and J. S. Moore, ‘‘A fast string searching algorithm,’’ Communications of the ACM **20** (October 1977), pp. 762 772.

[5] Commentz-Walter, B, ‘‘A string matching algorithm fast on the average,’’ Proc. 6th International Colloquium on Automata, Languages, and Programming (1979), pp. 118 132.

[6] Haertel, M., ‘‘Gnugrep-2.0,’’ Usenet archive comp.sources.reviewed, Volume 3 (July, 1993).

[7] Horspool, N., ‘‘Practical Fast Searching in Strings,’’ Software — Practice and Experience, 10 (1980).

[8] Hume A., personal communication (1991).

[9] U. Manber, ‘‘Finding Similar Files in a Large File System,’’ Usenix Winter 1994 Technical Conference, San Francisco (January 1994), pp. 1 10.

[10] U. Manber and S. Wu, ‘‘GLIMPSE: A Tool to Search Through Entire File Systems,’’ Usenix Winter 1994 Technical Conference, San Francisco (January 1994),

[11] Wu S., and U. Manber, ‘‘Agrep — A Fast Approximate Pattern-Matching Tool,’’ Usenix Winter 1992 Technical Conference, San Francisco (January 1992), pp. 153 162.

[12] Wu S., and U. Manber, ‘‘Fast Text Searching Allowing Errors,’’ Communications of the ACM **35** (October 1992), pp. 83 91.

[13] M. Roesch. Snort – lightweight intrusion detection for networks. In Proceedings of LISA’99: 13th Systems Administration Conference, pages 229–238, November 1999.

[14] R. S. Boyer and J. S. Moore. A fast string searching algorithm. Communications of the ACM, 20(10):761–772, 1977.

[15] A. V. Aho and M. J. Corasick. Efficient string matching:
An aid to bibliographic search. Communications of the ACM,
18(6):333–340, 1975.
[16] B. Commentz-Walter. A string matching algorithm fast on the
average. Proceedings of ICALP, pages 118–132, July 1979.
[17] S.Wu and U. Manber. A fast algorithm for multi-pattern searching.
Technical Report TR-94-17, Department of Computer Science,
University of Arizona, 1994.
[18] E.P. Markatos, S. Antonatos, M. Polychronakis, and K.G. Anagnostakis.
Exclusion-based signature matching for intrusion detection. In Proceedings of the
IASTED International Conference on Communications and Computer Networks
(CCN), pages 146–152, November 2002.
[19] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood.
Deep packet inspection using parallel bloom filters. In 11th Symposium on High
Performance Interconnects, August 2003.