Journal of Innovative Technology Convergence
Vol. 1, No. 2 December 2019, pp. 33-42

A Study of a Local Search Engine for Desktop PCs using an Enhanced Boyer-Moore Algorithm

Lorlyn S. Sernicula [1], Melinda L. Manalo [2], Gryam Abner M. Niebre [3], Rajean C. Anastacio [4], Jhona P. Alagos [5], Jason P. Sermeno [6*]

Abstract: The volume of digital information and documents stored on desktop computers and over the Internet has been rapidly increasing in the past few years. Thus, search algorithms, methods, and programs are becoming popular to help in finding the intended information or files. This study proposes a string matching algorithm to improve the pattern matching technique of the traditional Boyer-Moore (BM) search algorithm. The proposed enhanced BM algorithm simultaneously scans the text from both sides (left and right) using two windows wherein each window has a size that is equal to the pattern length.

Keywords: Boyer-Moore search algorithm, local search engine, string searching, string matching, Algorithm

1. Introduction

A personal computer (PC) search engine simply refers to an information retrieval program installed on your computer in order to help in finding files and information [1]. The search engine program searches through your folders and files using the keywords that you have entered and returns a list of results (files) that contain the keywords. It also refers to an "offline search engine" as it only searches local files as compared to a web search engine.

The desktop search features of the current operating systems (OS), e-mail programs, word processors, and other applications have fewer capabilities as compared to popular Web search engines

[1] College of Computer Studies, University of Antique, Sibalom, Antique, Philippines
Email: lorlyn.sernicula@gmail.com
[2] College of Computer Studies, University of Antique, Sibalom, Antique, Philippines
Email: melinda.manalo@antiquespride.edu.ph
[3] College of Computer Studies, University of Antique, Sibalom, Antique, Philippines
Email: gryamniebre@gmail.com
[4] College of Computer Studies, University of Antique, Sibalom, Antique, Philippines
Email: rajean.anastacio@antiquespride.edu.ph
[5] College of Computer Studies, University of Antique, Sibalom, Antique, Philippines
Email: jhona.alagos@antiquespride.edu.ph
[6*] College of Computer Studies, University of Antique, Sibalom, Antique, Philippines
Email: jasonpsermeno@gmail.com (Corresponding Author)

Received [May 11, 2019]; Revised [July 15, 2019]; Accepted [August 20, 2019]

© 2019 The Authors.
This is an open access article licensed under the Creative Commons Attribution-NonCommercial 4.0 International License.
To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/4.0/.

Published by InnoCon Publishing.
ISSN 2704-4440

[2]. These features were limited to simple keyword searches of a set of files that usually belongs to a single file type. Desktop search engines face additional challenges such as recognizing which of the many file types it is dealing with, deriving the metadata that the file authors have chosen to include, and being efficient and avoiding imposing substantial processing or memory load on the desktop computer. In addition, the integration of desktop and Web search capabilities into the same application presents security and privacy challenges.

Other notable algorithms commonly used for searching include the naïve algorithm, Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), and the Rabin-Karp (RK) algorithms [3]. The idea of the naïve algorithm is to make a comparison character by character of the text $T[s...s + m - 1]$ for all $s \in \{0, \ldots, n - m + 1\}$ and the pattern $P[0...m - 1]$. This process returns all the valid shifts found. On the other hand, KMP refers to a linear time algorithm wherein the main characteristic is that when a match between the pattern and a shift in the text fails (*i.e.*, a mismatch occurs), the algorithm will use the information given by a specific table and obtained by preprocessing of the pattern in order to avoid re-examining the characters that have been previously checked, thus, limiting the number of comparisons required [4]. The RK algorithm is based on hashing techniques and uses a totally different approach in solving the string matching problem. A hash function $h(x)$ for the pattern $P[0...m-1]$ is computed and then a match is identified by using the same hash function for each substring of length $m - 1$ of the text.

The BM algorithm's basic concept is that the match is performed from right to left as shown in Figure 1. This characteristic has allowed the skipping of more characters as compared with the other searching algorithms. For example, if the first character matched of the text is not contained in the pattern $P[0...m - 1]$, then m characters can be immediately skipped [3][5]. It is considered as an efficient string searching algorithm that preprocesses the string being searched for (*i.e.*, the pattern), but not the string being searched in (*i.e.*, the text). Thus, the BM algorithm is very well-suited for applications in which the pattern is much shorter than the text or where it persists across multiple searches [5].

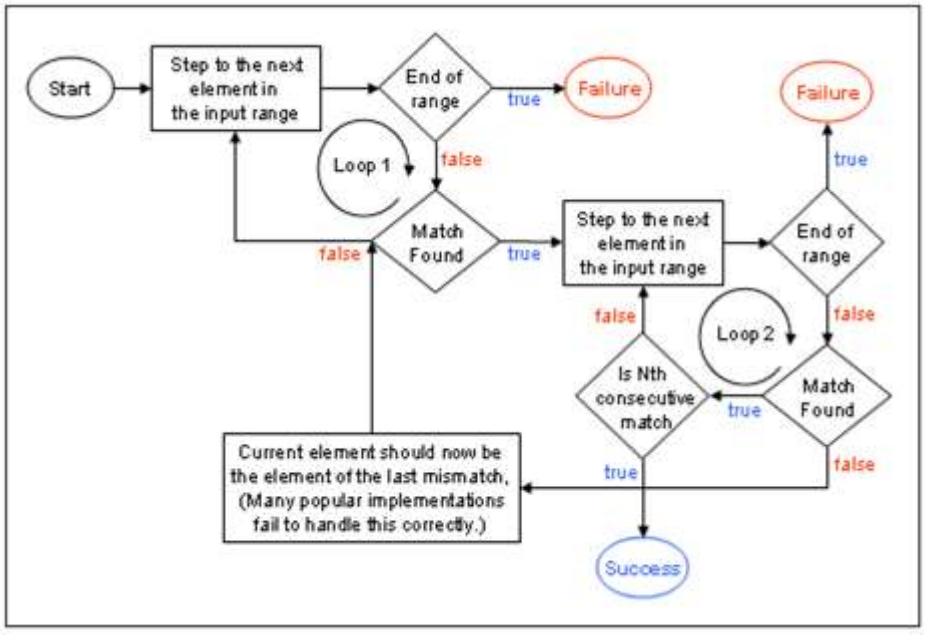


Figure 1. Traditional Boyer-Moore Search Algorithm

This paper aims to enhance the traditional BM search algorithm as a basis for a Local Search Engine intended for desktop computers. The enhanced BM search algorithm specifically attempts to achieve the following objectives: (1) determine the average duration of the traditional and the enhanced BM

algorithm; (2) determine the average accuracy rate of the traditional and the enhanced BM algorithm; and (3) determine the average performance of the traditional and the enhanced BM algorithm.

The remainder of this paper is organized as follows: Section 2 outlines the different related literature as a basis for the design of the proposed enhanced BM algorithm; Section 3 details the conceptual framework for the enhanced BM algorithm; the discussion on the significance of the proposed enhanced BM algorithm is highlighted in Section 4; and Section 5 concludes the study.

2. Related Literature

This section outlines the various literature that was helpful and significant in the design of the proposed enhanced BM algorithm as a local search engine for desktop PCs. These include the improvements on the traditional Boyer-Moore (BM) algorithm, the Search Engine Optimization (SEO), the PageRank, and the DocFetcher desktop search application.

2.1 The Traditional Boyer-Moore Algorithm

String searching and pattern matching algorithms had been popular as the volume of digital information over the Internet and the number of files stored on desktop PCs rapidly increase over the past decades. The performances of these algorithms were evaluated and compared with traditional string searching algorithms such as the Breadth-first search (BFS), Bellman-Ford (BF) algorithm, the traditional BM algorithm, and the Boyer-Moore-Horspool (BMH) algorithm.

The traditional BM algorithm is considered an efficient string searching technique and has been the standard benchmark for the practical string search literature [5][6]. The algorithm was developed by Robert S. Boyer and J. Strother Moore in 1977. The concept of the algorithm is that it preprocesses the pattern string that is being searched in the text string.

In 1980, Nigel Horspool has simplified the traditional BM algorithm which is related to the Knuth-Morris-Pratt (KMP) algorithm. The result becomes the Boyer-Moore-Horspool (BMH) algorithm or simply Horspool's algorithm which refers to an algorithm for finding substrings in strings. The BMH algorithm preprocesses the pattern in order to produce a table containing, for each symbol in the alphabet, the number of characters that can safely be skipped [7].

In 1990, the Boyer-Moore-Horspool-Sunday (BMHS) algorithm was developed to further improve the BMH algorithm. This algorithm determines the shift amount through the character to the right of the text that is being processed. Just like the BMH algorithm, the BMHS also disregards the good suffix rule and uses only the last occurrence table which does not exclude the last character of the needle, resulting in fewer comparisons and faster execution time [6][8].

In 2010, the BMHS2 was proposed which aims to address the shortcomings of the BMHS algorithm [9]. In addition, another BMHS improvement was proposed that takes advantage of the position information of the last character and its adjacent character in the current attempt window to get a bigger jump distance in each jump in order to make the algorithm more efficient [10].

Recently, there were more optimizations made to the traditional BM algorithm which have been significant in searching programs and applications.

2.2 Search Engine Optimization (SEO)

Search Engine Optimization (SEO) is a methodology used by websites in order to improve their quality and quantity of information and be better indexed by search engines. It aims to improve the website's content by making it more attractive to visitors as well as to search engines. In addition, it aims for a website to be better ranked by one or several targeted search engines, thus, will be appearing

higher in their results list [11]. SEO is also considered as a set of rules that can be followed by website owners to optimize their content for search engines and thus improve their search engine rankings. SEO is essentially important for websites in improving their rank for search results and to get more page views from human users. This search engine rank enables a better and optimized presentation of results to users which will allow them to view the most popular pages among the available pages in the search results [12]. Figure 2 outlines some of the features of SEO that will be helpful to the success of any website.

Figure 2. Search Engine Optimization

2.3 PageRank

PageRank refers to a method in measuring the importance of website pages. PageRank was actually named after Larry Page, one of the founders of Google [13]. According to Google, "PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites".

PageRank is the first and the best-known algorithm used by Google Search to rank web pages in their search engine results. Its algorithm outputs a probability distribution which is used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. The PageRank computations require several iterations through the collection of documents (*i.e.*, any size) to adjust approximate PageRank values to more closely reflect the theoretical true value [14].

2.4 DocFetcher

DocFetcher is an Open Source desktop search application [15]. It allows you to search the contents of files on your desktop computer. It can be thought of as "Google for your local files". DocFetcher may run on Windows, Linux, and OS X platforms. The features of the DocFetcher main user interface are shown in Figure 3. Labeled as (1) shows a text field where queries can be entered; the result pane in (2) displays the search results; the preview pane at (3) shows a text-only preview of the file currently selected in the result pane where all matches in the file are highlighted in yellow; the results can be filtered by specifying the minimum and/or maximum file size in (4); filter by file type in (5); filter by location in (6); and the buttons at (7) are used for opening the manual, opening the preferences, and minimizing the program into the system tray, respectively.

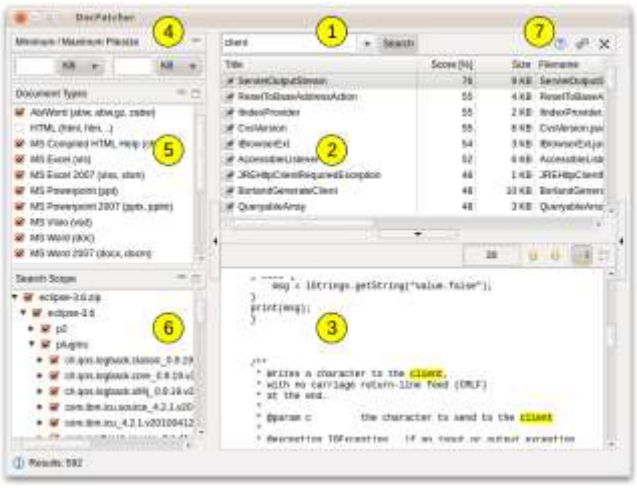


Figure 3. DocFetcher [15]

3. The Enhanced BM Algorithm

The traditional BM algorithm is considered the most popular and efficient string-matching algorithm in common applications. It is often implemented in text editors using the "*search*" and "*substitute*" commands. It works by scanning the characters of the pattern (*i.e.*, string to be searched) beginning with the rightmost one and performing the comparisons with the text (*i.e.*, a string being searched) from right to left. In case of a mismatch or if there is a complete match of the whole pattern, two recomputed functions (*i.e.*, bad character and good suffix functions) will be used by the algorithm to shift the window to the right [5][7].

The BM algorithm makes use of the following definitions:

- $S[i]$ indicates the character at index i of string S, counting from 1.
- $S[i..j]$ indicates the substring of string S starting at index i and ending at j, inclusive.
- A prefix of S is a substring $S[1..i]$ for some i in the range [1, n], where n is the length of S.
- A suffix of S is a substring $S[i..n]$ for some i in the range [1, n], where n is the length of S.
- The string to be searched for refers to the *pattern* and is denoted by P. Its length is n.
- The string being searched in refers to the *text* and is denoted by T. Its length is m.
- An alignment of P to T is an index k in T such that the last character of P is aligned with index k of T.
- A match or occurrence of P occurs at an alignment if P is equivalent to $T[(k-n+1)..k]$.

The BM algorithm searches for occurrences of pattern P in T by performing explicit character comparisons at different alignments. It utilizes the information gained by preprocessing P in order to skip as many alignments as possible. The operation of the algorithm begins by the alignment $k = n$, in order that the start of P is aligned with the start of T. Then, the characters in P and T are then compared starting at index n in P and k in T, moving backward, from right to left (*i.e.*, the strings are matched from the end of P to the start of P). The matching comparisons continue through either the beginning of P is reached (*i.e.*, a match has occurred) or a mismatch has occurred upon which the alignment is shifted forward (*i.e.*, to the right) based on the maximum value permitted by a number of rules. The comparisons are performed again at the new alignment, and the algorithm iterates until the alignment are shifted past the end of T (*i.e.*, no further matches can be found).

A shift during the course of the comparisons is calculated by applying two rules: the bad character rule and the good suffix rule. The algorithm utilizes the knowledge gained from character comparisons to skip future alignments that definitely won't match. The following principles apply:

- In case of a mismatch, the information of the mismatched text character is used to skip the alignments (*i.e.*, bad character rule).
- In the case of matching some characters, the information of the matched characters is used to skip the alignments (*i.e.*, good suffix rule).
- The alignments must be tried in one direction, then character comparisons must be tried in opposite directions (*i.e.*, for longer skips).

The conceptual framework for the proposed enhanced BM algorithm is depicted in Figure 4. The key search string-matching words were determined using the formula tool by comparing the length of the keywords to the text string words. The search local desktop search results are displayed in response to a query by the user.

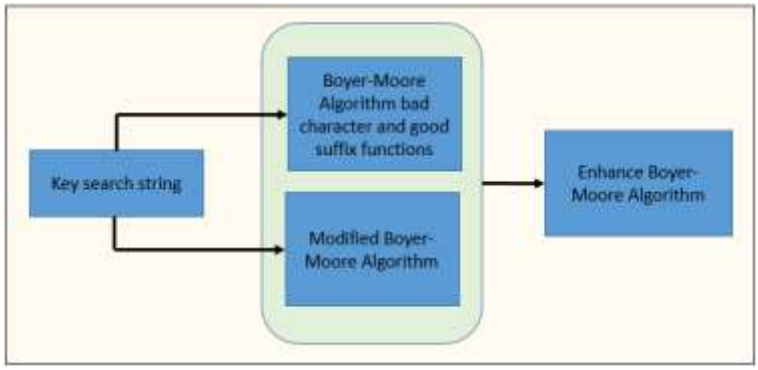


Figure 4. Enhanced BM Search Algorithm Conceptual Framework

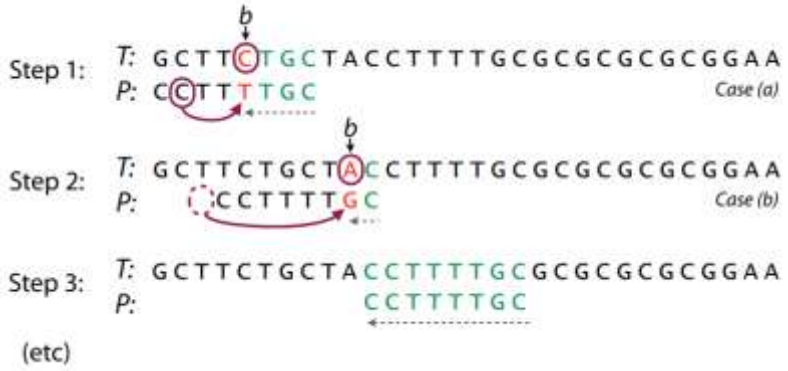
```
for (int i = 0; i <= indexOfTextToStopSearching; i += skip){
    skip = 0;
    intindexOfLastCharacterOfPattern = patternLength - 1;
    for (int j = indexOfLastCharacterOfPattern; j >= 0; j--){
        charcurrentCharacterInPattern = pattern.charAt(j);
        charcurrentCharacterOfTextBeingCompared = text.charAt(i + j);
        if (currentCharacterInPattern != currentCharacterOfTextBeingCompared)
            inttoSkip = j - right[currentCharacterOfTextBeingCompared];
        skip = Math.max(1, toSkip);
        intindexOfNextCharacter = i + j + 1;
        if (indexOfNextCharacter<= (textLength - 1) &&
            text.charAt(indexOfNextCharacter) == ' ')
            skip += (patternLength - skip) + 1;
        break;
        if (skip == 0)
        return i; // found
        return -1; // not found
}
```

Figure 5. The Implementation of Enhanced BM String Matching Algorithm

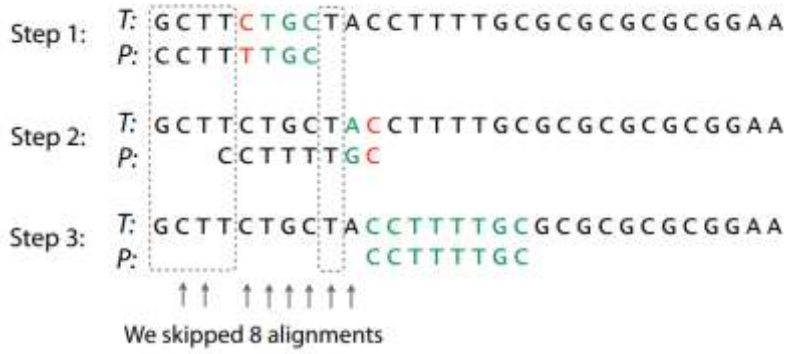
The algorithm depicted in Figure 5 shows the implementation of the enhanced BM string matching algorithm that can perform better with the traditional BM search algorithm.

3.1 Bad Character Rule

Whenever a mismatch occurs, let *b* be the mismatched character in *T*. Then, the alignments are skipped until (a) *b* matches its opposite in *P*, or (b) *P* moves past *b* [16]. The bad character rule process is depicted in Figure 6.



(a) Labeling a Mismatch with *b*



(b) Skipped Alignments during the Process

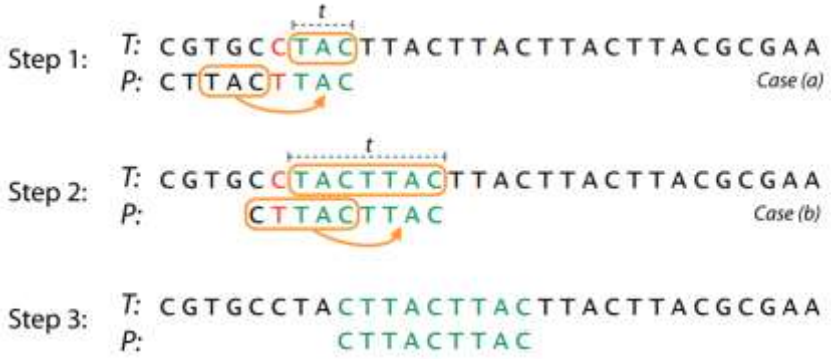
(c) Bad Character Rule Preprocessing

Figure 6. The BM Algorithm Bad Character Rule [16]

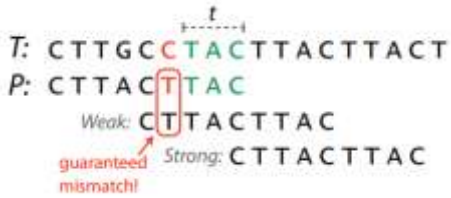
A | Σ |-by-*n* table must be built as soon as the pattern *P* is known. That is if *b* is the character in *T* that has mismatched and *i* is the mismatch's offset into *P*, then, the number of skips is specified by the element in b^{th} row and i^{th} column.

3.2 The Good Suffix Rule

In the good suffix rule, let *t* refer to the substring of *T* that matched a suffix of *P*. Then alignments can be skipped until (a) *t* matches opposite characters in *P*, or (b) a prefix of *P* matches a suffix of *t*, or (c) *P* moves past *t*, whichever of these conditions happens first [16]. The good suffix rule operation is depicted in Figure 7.



(a) The Weak BM Algorithm Good Suffix Rule Operations



(b) Stronger Good Suffix Rule Operations

Figure 7. Variations of a Good Suffix Rule [16]

Just like with the bad character rule, the number of skips that are possible using the good suffix rule can be predetermined into a few tables. The matching process depicted in Figure 7(a) is using the weak good suffix rule, while the strong good suffix rule is depicted in Figure 7(b).

4. Discussions

A local search is a concept that refers to the use of search engines that allow users to perform locally constrained searches against a structured database of local file listings. Local search engines were used in searching for files that were locally stored on laptops or desktop PCs. These search engines were using the Boyer-Moore algorithm or other string search algorithms. These programs are essentially beneficial, for example, if a particular text will be searched from a vast amount of storage, then it would take too much time in searching. Texts can also be searched over transmission paths having a limited string.

There are two algorithms that are being used to enhanced string searching, that is, the bad character function and the good suffix function. There have been a lot of optimization applications that were developed for string searches. In this paper, enhanced performance of the BM algorithm was considered as a basis for string searches in desktop PCs. The search process utilizes a table of predetermined skipped alignments to find the probability of having an occurrence of a pattern in the given input text and to find if the probability will lead to a successful or unsuccessful match.

The proposed enhanced BM algorithm will provide a significant and flexible way of searching in local desktop PCs to meet the user's demands. The speed and reliability of the enhanced BM algorithm will be determined and compared with the traditional BM algorithm. This algorithm can assist in efficiently finding solutions while performing a local search which is slowly progressing. It can lessen the waiting period while searching from vast storage and a large volume of files in desktop PCs. The comparative analysis of the proposed algorithm as compared with the traditional BM search is outlined in Table 1.

Table 1. Comparative Analysis for the Proposed Enhanced BM Search

Algorithm Capabilities	Algorithm	
	Boyer-Moore Algorithm for Desktop Search	**Proposed Enhanced Boyer-Moore Algorithm for Local Desktop Search**
Ability to search a string	√	√
Ability to search different files	√	√
Ability to reduce search runtime	×	√
Ability to lessen the duration time while checking the content	×	√

5. Conclusion

This paper has proposed an enhancement of the traditional BM string searching algorithm to form the basis for a local search engine for desktop PCs. The enhanced algorithm runtime depends on the size of the queue, that is, the number of index values in the queue which shows the number of expected patterns in text *T*. As compared with the traditional BM algorithm, the proposed enhanced algorithm for local desktop search will be capable of reducing the runtime length as well as lessen the time duration while performing a search on the contents of a file.

In the future, a prototype implementation for the proposed enhanced algorithm will be designed to further investigate its robustness and efficiency as compared with other string searching algorithms. The algorithm will be implemented using Java program.

References

[1] J. Simoes, "*What is a Computer Search Engine*", lookeen.com, www.lookeen.com/blog/what-is-a-computer-search-engine (Accessed March 8, 2019).

[2] B. Cole, "*Search Engines Tackle the Desktop*", Computer, vol. 38, no. 3, March 2005, pp.14-17, doi: 10.1109/MC.2005.103.

[3] M. Gou, "*Algorithms for String matching*", student.montefiore.ulg.ac.be, www.student.montefiore.ulg.ac.be/~s091678/files/OHJ2906_Project.pdf (Accessed March 8, 2019).

[4] D. E. Knuth, J. H. Morris Jr., V. R. Pratt, "*Fast Pattern Matching in Strings*", SIAM Journal on Computing, vol. 6, no. 2, 1977, pp.323–350, doi: 10.1137/0206024.

[5] R. S. Boyer, J. S. Moore, "*A Fast String Searching Algorithm*", Communications of the ACM, vol. 20, no. 10, October 1977, pp.762-772, doi: 10.1145/359842.359859.

[6] A. Hume, D. Sunday, "*Fast string searching*", Software: Practice and Experience, vol. 21.no. 11, November 1991, pp.1221-1248, doi: 10.1002/spe.4380211105.

[7] N. Horspool, "*Practical fast searching in strings*", Software: Practice and Experience, vol. 10, no. 6, June 1980, pp. 501-506, doi: 10.1002/spe.4380100608.

[8] D. M. Sunday, "A very fast substring search algorithm", Communications of the ACM, vol. 33, no. 8, August 1990, pp.132-142, doi: 10.1145/79173.79184.

[9] L. Xie, X. Liu, G. Yue, "*Improved Pattern Matching Algorithm of BMHS*", in Proc. of 2010 International Symposium on Information Science and Engineering (ISISE), December 2010, pp.616-619, doi: 10.1109/ISISE.2010.154.

[10] J. Yuan, J. Yang, S. Ding, "*An Improved Pattern Matching Algorithm Based on BMHS*", in Proc. of 2012 11th International Symposium on Distributed Computing and Applications to Business, Engineering & Science, Guilin, China, October 19-22, 2012, pp.441-445, doi: 10.1109/DCABES.2012.115.

[11] A. Gandour, A. Regolini, "*Website Search Engine Optimization: A Case Study of Fragfornet*", Library Hi Tech News, vol. 28, no. 6, August 2011, pp.1-16, doi: 10.1108/07419051111173874.

[12] J. Y. Lemos, A. R. Joshi, "*Search engine optimization to enhance user interaction*", in Proc. of 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, February 10-11, 2017, doi: 10.1109/I-SMAC.2017.8058379.

[13] "*Facts about Google and Competition*", web.archive.org, www.web.archive.org/web/20111104131332/https://www.google.com/competition/howgooglesearchworks.html (Accessed March 8, 2019).

[14] "*Page Rank Algorithm and Implementation*", geeksforgeeks.org, www.geeksforgeeks.org/page-rank-algorithm-implementation/ (Accessed March 8, 2019).

[15] "*DocFetcher*", sourceforge.net, www.docfetcher.sourceforge.net/en/index.html (Accessed January 8, 2021).

[16] B. Langmead, "*Boyer-Moore*", www.cs.jhu.edu/~langmea/resources/lecture_notes/boyer_moore.pdf (Accessed March 8, 2019).