International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958, Volume-8 Issue-4C, April 2019

A Novel Algorithm for Enhancing Search Engine Optimization

Ritu Sachdeva (Sharma), Sachin Gupta

Abstract*: Google search engine uses Page Rank algorithm to rank websites in their search results. But, Page Rank calculates only the importance of each page rather than relevancy. Whenever a user triggers a query to be searched, searching algorithm finds pages associated with the query term i.e. on basis of relevancy. Then, Page Rank selects the most imperative result. Different Search Engine use different searching algorithm plays an indispensable role in searching relevant results. But efficiency of searching algorithm is calculated by drafting algorithms for specific categories of data like strings. In numerous key approaches, the trie is a recognized fast access method. The proposed algorithm is a search algorithm for path-compressed trie for keyword searching in a database through a search engine. Faster searching optimizes the search engine and speeds up the complete process of creating final results. Thus, greater SEO (search engine optimization), faster will be Page Rank Algorithm.

***Index Terms*: binary trie, LC-trie, Page Rank, PAT, PATRICIA, Search Engine Optimization, Trie**

I. INTRODUCTION

Information Retrieval is used to illustrate the procedure of searching a keyword in a document or multiple documents. To search the relevant content of query triggered by user, Search Engines are designed. Search engine gives the searching potential and approach to document content by using techniques that overdo author names, document titles or other high-level metadata [26]. An amalgamation of techniques and practices by which any website can enhance ranking in search engine results page leads to optimization of search engine or SEO. SEO stands for search engine optimization [5]. It facilitates locating the website appropriately and at ease referencing for the researchers in case of need of site. On-Page SEO and Off-

Page SEO are two principal pillars of SEO. On-Page SEO, provides good content and keywords selection, placing keywords on correct places, giving suitable title to every page [15] or site structure. This makes certain that everything is done on the actual webpage to upgrade the rank while Off-Page SEO incorporates link building and escalating link popularity by capitulating open directories, search engines,

link exchange and social media. Everything is done off the page to enhance the rank [13].
Basically, there are three key processes in finding results after searching. These are crawling, indexing and ranking.

Crawling is a procedure by which search engines find relevant content on the web [35]. For this, search engine uses a program known as crawler, bot or spider. This program uses an algorithmic process to conclude names of sites to crawl and their frequency. While moving through a site, the crawler also identifies and accounts more links on these pages and enlists them for future crawling. This is the way to find a new content.

Indexing: After crawling, all underwent pages are compiled to a substantial index of all the words seen and their location of each page. This compiled content is then stored, organized and interpreted by algorithm of search engine to appraise its significance compared to similar pages [33].

Ranking: Search engine will check for pages within their index that are a closest match after entering a keyword into a search box. Then a score based on an algorithm consisting of hundreds of different ranking signals is allotted to these pages. These pages either images or videos will be displayed to the user in order of score.

The factors [34] influence the searching in Google search engine are Page specific factors, Anchor text of inbound links and Page Rank. Page Rank algorithm was invented by Page and Brin in 1998. Google was the first search engine that used it as a prototype. The aim of this algorithm is to estimate the significance of a webpage based on the interconnection of the web. The rationale behind this algorithm is that a page having more incoming links is more imperative than a page with less incoming links. It is also noted that if a page is linked with a page of high importance, then it is also imperative [8].
Following is simplified model of a search engine:

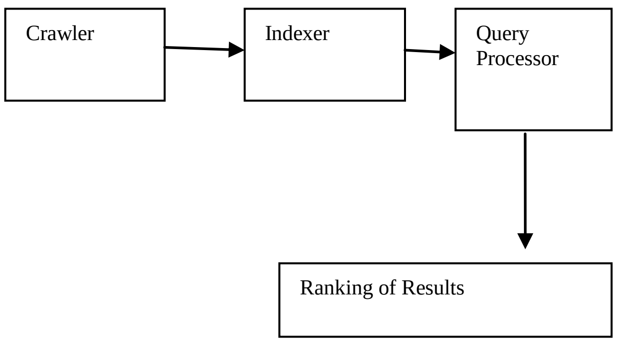

Fig.1: A basic search engine model

Revised Manuscript Received on December 22, 2018.

Ritu Sachdeva (Sharma), Computer Science Department, Modern Vidya Niketan University, Palwal, Haryana, India
Sachin Gupta, Computer Science Department, Modern Vidya Niketan University, Palwal, Haryana, India

*Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication*

The mechanism of search engine is done via crawler, indexer and a query engine [2]. The information, the user wants, is fetched by triggering a query. When a user triggers a query to be searched, Google selects the results that contain the search term. Selected results are sorted by Page Rank in a hierarchical way from the most important to least important results. Then, results are shown to the user [2].

First, the crawler crawls through the entire world wide web [6]. Then, the crawler moves from one page to another to keep track of the existing links as well as updates the database with new web pages and links. These links are forwarded and organized into the indexer on the basis of relevancy of the query done [14]. The degree of matching a site with a particular query is known as Relevance [4]. It is evaluated by IR score (from page specific factors) and anchor text of inbound links of a page (weighted by position and prominence of the search term within the document) computed by Google. On behalf of this score, the results are operated by Page Rank algorithm. The principal page in Google becomes the most productive page in the result list to be displayed for the client [3]. This IR-score is then united with Page Rank as an indicator for the general importance of the page. To unite the IR score with Page Rank, the two values are multiplied. These values cannot be added, for, pages with a very high Page Rank would rank high in search results even if the page is not linked to the search query. Finally, pages associated to the query are enlisted and the search is done. Then, the results are shown to user.

Answering a query by scanning the entire text is an easy job in case of small web but in large volumes of information, powerful mechanisms are required either in case of searching or against unauthorized disclosure [20]. Its prohibitive costs render it impractical and inflexible. Response time as well as quality of the output determines the efficiency of an information retrieval system. Response time is evaluated on behalf of size and arrangement of corpus as well as type of index used and query posed. The quality of the output is a qualitative and is measured by precision and recall [20]. But, key management becomes problematic in the face of frequent database updates. As data structures are the different approaches meant to store data in the permanent memory. Arrays, Linked lists, hash tables etc. are considered as storage structures meant for storing the data while stacks, queues and priority queues are process-oriented data structures meant for processing the data. Another category of data structure is descriptive data structure meant to describe the nature of the data. Linear lists, binary trees etc. come under this category [22]. The approach used in this research is based on Binary tries and their modifications. A trie is a rooted tree where each child branch is marked with letters in the alphabet $\sum$. A trie node can be stored as binary search tree (BST). Moreover, the branches of a node can also be stored in a BST. The result is in the reduction of total space of the tree to $O(|T|)$ but increase in the query time to $O(|P|\log\sum)$. By take advantage of the basic properties of tries, the complexity of database management as well as cost of these data structure is significantly reduced. This paper proposes a modified form of trie known as LPC-trie. At times, if is difficult to handle large texts by some applications due to inefficiency of main memory. To overcome this problem, a partial data structure that fits in main memory could be utilized that helps to reduce the number of time-consuming accesses to secondary memory. This trie is stored in main memory used as an index for searching into secondary memory. Also, this paper use level-path compressed tries. This data structure is used to search the strings stored in key value form. Indeed, relative compaction, avoidance of management of dynamic memory and traversal during searching only are different aspects of implementation via trees. Thus, tries are suitable where random access is not required. Thus, dynamic tries are used which avoid pointers at some cost in memory management and lead to stable searching.

II. Preliminaries

Rene de la Briandais [27] proposed Tries in the beginning in 1959 while Fridkin and the term “trie” was invented by Fridkin as an derivative from “Information Retrieval”. A trie is defined as a tree of nodes that supports Find and Insert operations. Tries are broadly used to represent a set of strings [16]. It was introduced by Morroson in 1968 as a renowned data structure. The entire information i.e. contents of each node in the path from the root to the node is stored in the trie except node itself [16]. A key is used to represent each path between the root and a leaf. It is not viable to symbolize any key that is a prefix of another. To overcome this complication, a special marker on each node is set up to spot the node whether it is end of key or not. The end node is marked as dangling node. Trie can keep an implicit ordering of the keys on behalf of ordering of the characters of the alphabet. Due to implicit ordering, lexical predecessor or successor of any key can be easily found. Thus, trie can sort a set of keys (like a binary tree) but unlike binary tree, trie does not exhibit the degenerate O(n2) worst case behavior in case if keys are inserted into a binary tree in a “bad” order. In trie, each node contains information about child pointers as well as a having a flag that contains key value and whether the node is dangling or not [11].

It is rather simple but a disastrous approach for many applications due to in excess of memory consumption from an engineering point of view. As a trie needs almost 35 times more memory to embody this small dictionary the dictionary itself. In today’s scenario, seven megabytes might not give the impression of bulky memory but in case of large problem, this factor can be disastrous.

A. Path compressed trie

Patricia tree is a variation of trie. A PAT tree is a binary tree structure in the field of information retrieval. Full form of Patricia is Practical algorithm to retrieve information coded in alphanumeric”[9]. It is a Path-compressed trie. This trie stores a set of keys that symbolize strings. To determine the path from the root to a key in sequence of characters in the key, the trie is structured. Thus, the length of this path is most proportional to the length of the key.

Thus, in case of short key strings, there is no need of balancing of height of the trie. Since, Patricia trie is the shallowest trie amongst various trie structures, it often used as an index in information retrieval systems. This paper focuses on a specific feature of Patricia that is specifically helpful in an effort to reduce the memory consumption of tries [10].

In a Patricia, a succession of nodes having only one child and are non-dangling nodes are disintegrated into a single transition. To store such information, each node in Patricia trie has a pointer that points to the record information about eliminated nodes [11]. During implementation of Patricia trie, if the key sets is large to be stored, then it is imperative to compress these tries into compact data structure [10]. Implementing this compression results in several characters of a string for each transition instead of just one. It is to be noted that technique of combining leaf nodes can also be used for storage of keys in such kind of trie. Since only sequences of nodes having one child are collapsed, no such collapsed sequence can possibly be a prefix of another. Therefore each transition might represent many characters and the transition begins with the next character in the key and can possibly be valid at each step where only a single child is present. Therefore, the search is straightforward and efficient. Thus PC- trie is used as a binary trie because of its easy implementation. Although it is an improvement over the binary tries as it eliminated the dummy nodes from tree structure resulting in reduction of memory space needed for the storage of the tree, thereby, decreasing the length of the tree. But in case of large database, the problem of high number of memory accessed still exists [23].

B. Level- compressed trie

Level- compressed trie was introduced by Nilsson [28]. It is considered as LC-trie. This trie structure is combination of path and level-compression. It is a trie in which each complete subtree of height h is collapsed into a subtree of height 1 with 2h children. Thus, the root is replaced with a node of degree equal to the size of the largest full sub-tree originated from the root. LC-trie uses adaptive branching i.e. the number of descendants of a node depends on the distribution of the elements. It represents the binary trie in an efficient way [20]. To convert binary trie into LC-trie from a binary trie, first binary trie undergo path-compression and then every node v that is rooted at a complete subtrie of maximum depth k is expanded to create a 2k -degree v' node . The leaves of the subtrie rooted at node v' in the basic trie become the 2k children of v'. This expansion is performed recursively on each subtrie of the basic trie. Small number of levels in the trie results in storage minimization.

It is an improved version of PAT. In certain conditions, it can change branching factor of some nodes. It is faster than PAT, for, LC-tries increase branching factor resulting in the shallow depths of trie. To reduce the size of Patricia trie, LC-trie replace the i highest levels by a single node of degree 2i. This procedure is recursively performed in a top-down manner until total trie is level-compressed. The ultimate result is called as LPC trie i.e. Level-Path Compressed trie. It is a dynamic variant of a static LC-trie. LPC-trie is data structure that stores the data in simple order and supports fast retrieval of elements and efficient nearest neighbor and range searches [1]. These LPC-trie use compression of path followed by level-compression of Patricia tries, so called digital tries. Literature has various implementation of dynamic trie structures [17, 18]. But the methods used previously have drawbacks of consuming more memory than a balanced search tree. To overcome this problem, compression of tries are used. In fact, while implementation, LPC-trie consumes amount of memory equivalent to a balanced BST. Following figures show the process of LPC-trie graphically. Figure 4 presents the final state after implementation of LPC-trie after path and level-compression. The average depth of LPC-trie is expected to be $\Theta(\log \log n)$.

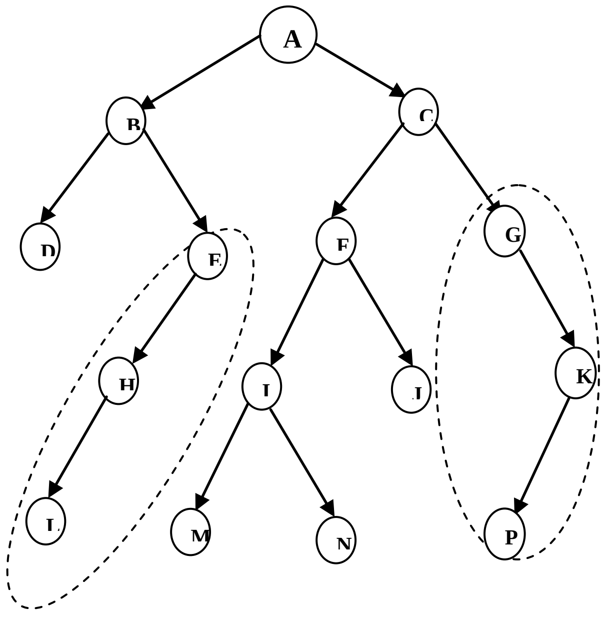

Fig. 2: A Binary Trie

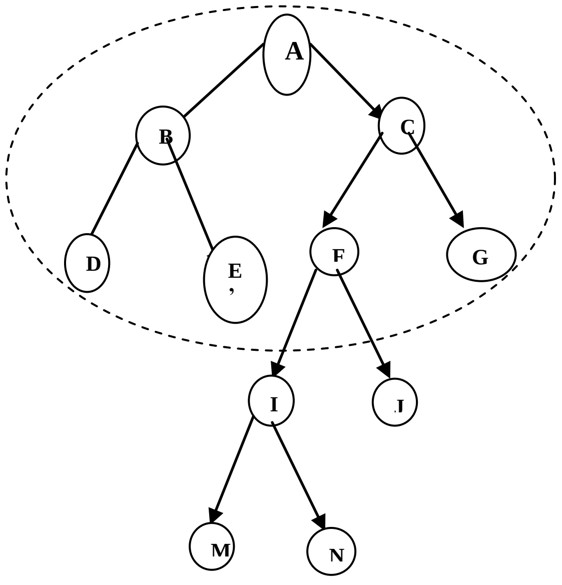

FIG. 3 : Path-Compressed Trie

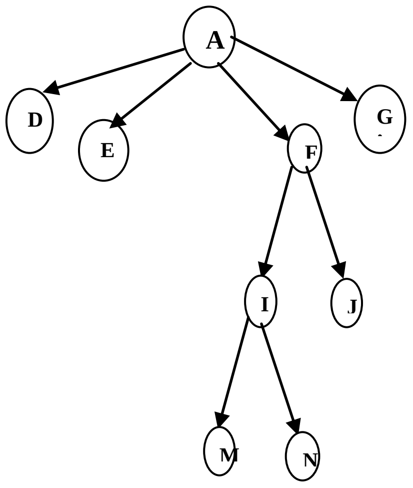

Fig. 4 : Level-Compressed Trie

III. Proposed Algorithm

Stage 1: To find a leaf node that is a single child.

STEP 1: Find whether a trie exists.

```
int trie_find(trie_t *node, char *key)
{  int i=0;
   trie t  *cur =node;
   int key_len=strlen(key);
   if (node = = NULL)
   return (NOT_FOUND);}
```

Step 2: Find the children of the node for the searched keyword till the length of the keyword.

```
for (i = 0; i <key_len,i++)
{     cur = trie_find_child(cur, key[i]);
   if (cur ==NULL)
   return (NOT_FOUND);}
```

Step 3: Find whether the corresponding child is a leaf node and if it is then check if it is the only child of the parent node. If yes then apply the path compressed function.

```
for (i = 0; i < key_len; i++)
{      if (trieIsEndOfKey(cur))
      {if (right[root]==NULL)
        path_compressed();}
      else
        return (NOT_FOUND);}
```

Stage 2: Algorithm for Path-Compression Function

PATH_COMPRESSED(INFO,ROOT)
The INFO gives the data stored in the parent node ROOT.

```
1. INFO[ROOT]:=INFO[ROOT]+INFO.LEFT[ROOT]
2. Exit.
```

IV. Comparison with other Data Structure

Table 1 shows the complexities of different data structure such as Binary-Search Tree, Simple Trie and LPC-Trie. The result below shows that average complexity of LPC-Trie is less as compared to the other data structures.

TABLE I
Comparison of Complexities of different Data structures

Type of Data Structure	Average Complexity
Binary Search Tree	O (log n)
Simple Trie	Θ(log n)
LPC-Trie	Θ(log log n)

V. ANALYSIS

The complexity of a trie structure can be defined as O(n*m). A few operations like initialization have to be performed whenever a string is traversed and added to the existing structure. It is a constant time operation as time complexity is O(C) where C is a constant. For creation of a new node, it is O(1). Its worst case complexity is (n*m) where m is the length of the longest string. This length will be repeated for all the strings with length m [32].

Keyword searching is the fundamental technique of the information retrieval system which tries solve the large search space problem as the documents to be searched could be of any size [21]. Information retrieval index all the documents' wording(s) on which search would be performed [26]. Indexes are to be build to implement well-organized searches. Indexes can be forward as well as backward. A forward index or simply index is list of documents and which words appear in them i.e. mapping from content to location while backward or inverted index is a list of words and documents i.e. mapping from location to content. The aim of an inverted index is to make fast full text searches at a cost of enhanced processing when a document added to database. It is the most prominent data structure used in systems meant for retrieval of documents on a large scale like search engine. The mechanism of search engine involves collecting, parsing and storage of data to retrieve fast and accurate information retrieval. The technique used by search engines is inverted indexing to trace the web pages having the keyword contained in the users' query but the performance of inverted index depends upon the searching of keyword in the database, maintained by search engines. This performance can be enhanced through the compression techniques. Compression techniques are integrating into modern information retrieval systems results in optimization of compression methods. It results in benefit of different types of posting information to the space and time efficiency of search engine. Here, LPC trie is used to index the keyword up to a certain optimum level.

It is to be observed that increasing the levels of trie will result in increasing the performance of retrieval but won't decrease the memory upto the same level. It is also examined that to search a keyword in the list, LPC-trie requires 56% less number of comparisons with indexing up to level 2 on an averages, as required by binary search.

This proposed method involves the compression of Patricia trie into the pre-order bit-stream. Efficiency in terms of space and time for proposed method is theoretically discussed.

A. Experimental Results

The alphabet size spans from alphabets/ characters a to z (case-insensitive) and a blank or space having a list of 27. To represent LPC-trie, linked list is applied and then implementation is done in Java 7.0.21. Java is a programming language. Well-defined types and semantics as well as automatic memory are salient features of this language. Due to this, it is extensively used language. This programming language sustains object-oriented program design. Keyword searching is done by path compressed and level compressed tries, up to the length of the keyword. After implementation of LPC tries, a list of the keywords were searched randomly from the database and generated in the browser.

B. Space Complexity

A simple trie is a n-ary trie but PAT and LC- trie is a binary trie. So, implementing PAT and LC-trie generates much better space reduction than simple trie structure. Figure 5 gives the comparison of space complexity of PAT, LC-trie, and simple trie to examine the memory consumption.

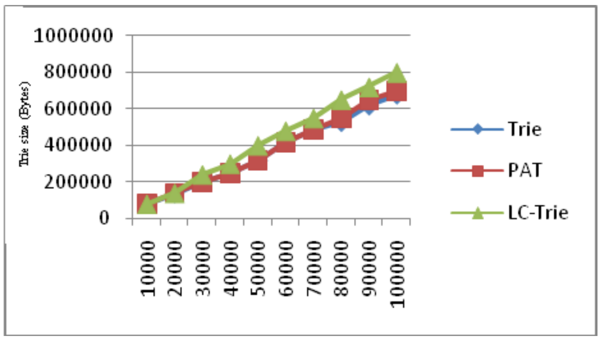

Fig.5: The evaluation of space complexity

It is observed that LC-trie can lessen the useless space more than both PAT and simple trie. Simple trie is the best choice for small set of word lists as it can reduce the useless space but, LC-trie is best choice for the large set of word lists as it consumes less space. It is also concluded that LPC-trie (Combination of PAT and LC) outperforms than other approaches for a large set of word lists.

C. Average Depth

The searching speed of trie structure gives its average depth. Greater average depth of trie structure results in slow searching speed. This result concludes that the searching speed of LC-trie is faster than both PAT and simple trie. Thus, this approach can preserve the searching speed of trie structure.

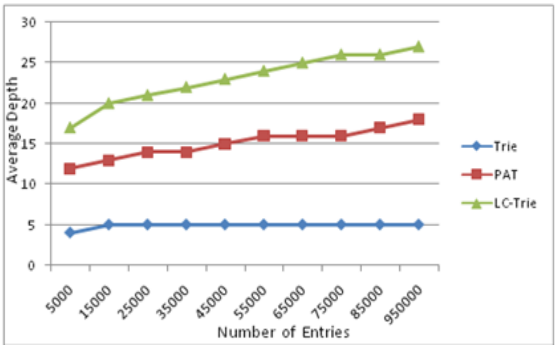

Fig. 6: Average Depth of PAT, LC-Trie & Trie

D. Communication delay

There is an aspect in page rank computation regarding updating rank values. It constitutes the following scenario:

- Each page communicates via exchanging its value with the pages that are directly connected to it through inlinks as well as outlinks and computes its own pagerank value
- To calculate pagerank (Point 1) , at any random time, initiation of communication by the pages can be done i.e. there is not a fixed order among the pages as well as the web (the centralized agent) which resolves the pages to update their values
- The degree of computation performed at each page is very mild

Due to this, there is delay in communication. There are following cases w.r.t. searching and ranking the web pages where the communication delay occurs. These are:

Case 1: One-to-one page updation

Case 2: Simultaneous updates (when the pages locally update their values by communicating linked pages)

Case 3: Due to sparsity inherent in the web as millions-trillions data leads to load in computation load at the pages and the pace of communication among the pages.

Communication delay leads to lower page loading, high latency period, and ultimately less search engine optimization. Less SEO affects ranking of websites.

The web undergoes an aggregation and forming groups of web pages based on any criteria i.e. either Number of inlinks, outlinks etc. Following equation determines the communication pattern at time k as each group computes the time average of its own state [31]:

$$N_i(k) = \begin{cases} 1, & \text{if node i sends its value to nodes in } V_{i,l}, l=1,2,\ldots, g_i \\ 0, & \text{if node i does not communicate} \end{cases}$$

The time taken from clicking the link to exhibiting the required content from the web page on the requesting browser is considered as Page Load while the minimum delay required for the data browsing the network between the web server and the web user's browser and vice versa is referred as Latency period.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

The approach used in this research paper can solve this problem of communication delay. LPC-trie is a highly compressed trie. It includes compression for both level and path. The memory requirement is somewhat equal to a balanced BST (binary search tree), but the expected average depth is lesser. Due to less average depth, the pages ceases update in their pagerank values after converging to a predefined level. These values are communicated to the linked pages. After that, no more computation as well as communication for this page is required. This results in reduction in time and ultimately the cost, of getting update page rank values. Thus, faster updation of rank values of web pages results in less Page Load as well as Latency period.

VI. CONCLUSION & FUTURE WORK

To fetch the web pages in response to the query triggered by the user as well as query keywords existed in the information database, searching algorithm of any search engine is responsible. But implementing trie data structure plays a significant role for enhancing the performance of proposed algorithm. This technique excels for fast retrieval of keyword to be searched as well as fetching of web pages containing the keyword. Thus, LPC-trie is the best choice. In it, number of comparisons is reduced to search a keyword by 56% on an average as compared to algorithm with binary search. Also, the major trouble of all web-site owners is the low pagerank, traffic, usage and visibility of the website on the search engines. So the users require different search engine optimization (SEO) techniques to get better pagerank, traffic, usage as well as visibility of their websites to get the top positions of search engine result page (SERP) [29]. Further for pragmatic analysis, this work will be extended by applying some analytical optimization techniques such as linear and non-linear. Also, same work will be extended by applying machine learning algorithm such as Support Vector Machine (SVM) for better results. Moreover, Updating individual page rank values results in multi-agent consensus problem as well as link failures. Future work will try to give emphasis on such problems.

Acknowledgement

I would like to show my gratitude to Dr. Sachin Gupta, Head (CS department), MVNU, Palwal for sharing their pearls of wisdom with me during the course of this research. I am immensely grateful to Dr. Rajesh Kumar Tyagi, Professor, SRM Institute of Science and Technology for his comments and feedback on the research paper. I am also thankful to “anonymous” reviewers for their so-called insights.

REFERENCES

1. Stefan Nilsson, Matti Tikkanen, ”Implementing a dynamic compressed Trie”Proceeding WAE’98, Saarbricken, Germany, (1998), Ed. Kurt , pp. 1-3
2. Heinz, S. Zobel, J. & Williams, H.E., “Burst tries: A fast, efficient data structure for string keys”, ACM Transactions on Information Systems 20(2), 192-223
3. Aanchal Kakkar et al, “Search Engine Optimization: A Game of Page Ranking, (2015) 978-9-3805-4416- 8/15 IEEE
4. John B. Killoran, “how to use search engine optimization techniques to increase website visibility”, (2013), IEEE transaction on professional communication vol 56 no. 1
5. Santosh Kumar Ganta, “Search Engine Optimization through Web Page Rank Algorithm”,(2011), ISSN: 0976- 8491, IJCST Vol. 2, Issue 3
6. Renalyn C. Antonio et al, “i-search: Document searching using Page Rank Algorithm”(2015), ISSN: 2278- 5299, Int. Journal of Latest Research in science and Technology, Volume 4, Issue 1: Page no. 70-74
7. N. Kaur & J. Kaur, “Development of Ranking algorithm for Search Engine Optimization”,(2014), International Journal of Engineering Research & Technology, India, ISSN: 2278-0181, Vol-3
8. K. Shum, “Notes on Page Rank Algoritnm”, (2013), ENGG2012B Advanced Engineering Mathematics
9. D.R. Morrison,” PATRICIA- practical algorithm to retrieve information coded in alphanumeric (1968), Journal of the ACM, 15, 514-534
10. Masami Shishibori et al, “A key search algorithm using the com pact Patricia Trie”, (1997), IEEE International Conference on Intel ligent Processing systems, Beijing, China
11. Niloufar shafiei, “Non-blocking Patricia Tries with Replace Operations”, (2013), IEEE, 33rd International conference on Distributed Computing Systems
12. M. Shishiburi et al., “An Efficient Method of Compressing Binary Tries”,(1996), IEEE, 7803-3280-6
13. Ayush Jain, “The Role and Importance of Search Engine and Search Engine Optimization”,(2013), IJETICS Vol. 2, Issue 3, ISSN-2278-6856
14. Shipra Kataria & Pooja Sapra, “A Novel Approach for Rank Optimization using Search Engine Transaction Logs”,(2016), 978-9-3805-4421-2/16, IEEE
15. Search Engine Optimization – tutorialspoint, simply easy learning, http://www.tutorialspoint.com/seo /seo_tutorial.pdg
16. Neil V. Murray et al, “Reduced implicate tries with updates”,(2008) Oxford University Press, Vol. 20, The author
17. P. Flat, “On the performance evaluation of extendible hashing and trie searching” (1983), Acta Informatica, 20: 345-369
18. E. Fredkin, “This memory” (1960), Communications of the ACM, 3.490-500,
19. Arne Andersson et al, “Efficient implementation of suffix trees”(1995), Software –Practice and Experience, VOL. 25(2), 129-141, CCC 0038- 0644/95/020129-13
20. Neha Mangla et al, “Context based Indexing in Information Re trieval System using BST”(2014), International Journal of scientific and research Publications, Volume 4, Issue 6, ISSN 2250-3153.
21. V. R. Kangavalli, G. Maheeja, “A study on the usage of data structures in Information retrieval” (2016), National Conference on Innovations in Communication and Computing Technologies
22. Monther Aldwairi et al. , “IP Lookup using two- level indexing and B-trees”,(2010), International Conference on Internet Computing, ICOMP ,Las Vegas Nevada, USA
23. Roberto Grossi et al, “Fast compressed tries through path decompositions” (2014), ACM journal of experimental algorithms, Vol. 19, No. 1, Article 1.8,
24. Erik Demaine, “Advanced data structures”(2012), Lecture L16 , Spring 2012
25. Isara Nakavisute, “Optimizing information retrieval (IR) time with doubly linked list and binary search tree (BST)” (2015), international journal of advanced computational engineering and networks, ISSN 2320- 2106, Volume-3, Issue-12
26. Rene De La Briandais, “File searching using variable length keys”, (1959), western joint computer conference, p.295-298, San Francisco, California
27. Andersson, A, Nilsson, S. “ Improved Behaviour of Tries by Adap tive Branching. IPL, 46(6):295-300
28. Hatab, Rayhan,” Improve Website Rank Using Search Engine Optimization(SEO)” (2014), Thesis Submitted to Faculty of Computer & Information Al-Madinah International University
29. Atish Das Sarma , Anisur Rahaman Molla , Gopal Pandurangan § Eli Upfal ,” Fast Distributed PageRank Computation”, ∗Appeared in Theoretical Computer Science (TCS) (2015), volume 561, pages 113- 121
30. H. Ishii, R. Tempo and E. Bai, "A Web Aggregation Approach for Distributed Randomized PageRank Algorithms” (2012), IEEE Transactions on Automatic Control, vol. 57, no. 11, pp. 2703-2717, doi: 10.1109/TAC.2012.2190161.

31. M. Thenmozhi, H. Srimathi, "An Analysis on the Performance of Tree and Trie based Dictionary Implementations with Different Data Usage Models" (2015), Indian Journal of Science and Technology, Vol 8(4), 364–375, ISSN (Online) : 0974-5645 , DOI: 10.17485 /ijst/2015/v8i4/59865
32. Sangita Karmakar and Soumen Swarnakar, "New Concept based Indexing Technique for Search Engine" (2018), Indian Journal of Science and Technology, Vol 10(18), DOI: 0.17485/ijst/2017/ v10i18/ 114018, 0974-6846 ISSN (Online) : 0974-5645
33. https://www.searchmetrics.com/knowledge-base/ranking-factors/
34. S. Hussien, "Factors Affect Search Engine Optimization" (2014), IJCSNS International Journal of Computer Science and Network Security, VOL.14 , No.9