Comprehensive Study and Comparison of Information Retrieval Indexing Techniques

Zohair Malki
Information Systems Department
The Collage of Computer Science and Engineering in Yanbu
Taibah University,
Saudi Arabia

***Abstract*—This research is aimed at comparing techniques of indexing that exist in the current information retrieval processes. The techniques being inverted files, suffix trees, and signature files will be critically described and discussed. The differences that occur in their use will be discussed. The performance and stability of each indexing technique will be critically studied and compared with the rest of the techniques. The paper also aims at showing by the end the role that indexing plays in the process of retrieving information. It is a comparison of the three indexing techniques that will be introduced in this paper. However, the details arising from the detailed comparison will also enhance more understanding of the indexing techniques.**

Keywords—Information Retrieval; Indexing Techniques; Inverted Files; Suffix Trees; Signature Files

I. INTRODUCTION

Information retrieval refers to the process of obtaining relevant information from an existing database that consists of different data that has been collected together. The current state of information retrieval depicts the existence of two search indexing. The first one is metadata and the second one is full text. Metadata is formally outlined as data about other sets of data [1]. It is more precisely described as information regarding other information that is structured. Metadata tool of information retrieval does not take into consideration the complexity of the search question. It can give relevant results to a simple query like the name of an author of a certain book. It can also provide relevant objects to other queries that are complex, like geographical codes. It is usually mostly utilized in education institutions in libraries other resources with large databases [1]. Catalogs of Libraries represent a remote metadata. Reviews of books, art collections as well as summaries also take the form of remote metadata.

On the other hand, full-text tool of retrieving information refers to the use of techniques that search documents stored from single computer. It can also refer to the use of techniques that search of a document that exists in a collection of documents in a collection of full-text database. A full-text search usually performs an examination of all the words that exists in the documents stored in the attempt of matching criteria of searching [1]. Over the years, it has been commonly used in online searches from databases of bibliography. Most application programs, as well as websites, provide capabilities of the full-text searches. Most search engines of the web usually employ techniques of full-text search. However, there are others that partially index the web pages. The only condition is that the web pages must undergo examination by their indexing systems.

Baeza-Yates and Ribeiro-Neto [2] explain that indexing with full text usually depends on the number of documents. Small numbers of documents can prompt direct scanning of the contents. A strategy known as serial scanning is applied to each query. Serial scanning is the protocol that is usually followed by most tools in the searching process. An example of such tools is Grep, which uses the strategy of serial scanning. Potential largeness of documents or increase of the quantity of queries to search prompts the division of full-text searching process into two stages. The first is indexing and the second one is searching. The first stage of indexing focuses on the scanning of all the existing documents. The stage also sees the building of a search term list. This list of search terms that is usually built at the indexing stage is referred to as an index. However, there are people refer it to as a concordance. The second stage of known as search only references the index in the performance of specific queries. The stage does not reference the text of the documents that are original.

However, this research will focus on a study of the indexing stage and the various techniques that are applied in the process.

There are several indexing techniques in information retrieval. However, this research is going to focus on three indexing techniques namely inverted files, suffix trees, and signature files. The three are the most commonly used techniques in the current world of information retrieval. The process of retrieving information usually begins with a query from a user into the system. A query is a statement that is formal indicating the need for particular information. An example of a query is the search of information in online search engines. Information retrieval queries stated by users do not usually offer a specific object or solution to the problem. Rather, it gives a collection of related objects that match the problem stated in the query. However, the objects have different levels of relevance to the query. Depending on the technique that is used, the relevance of available information is determined with respect to the entered queries. The results given in a form of objects are based on their relevance to the queries. The techniques have proven to the most reliable and usually generate desirable results. However, the indexing techniques differ in many ways. They usually differ in the way

they perform the relevancy tests. They also differ in their simplicity of application. The indexing that is performed by the techniques does not take a similar route. The research in the paper will outline the processes of indexing that the various techniques undertake.

Accuracy is a key factor in retrieving information [3]. Users expect accurate answers from the objects offered in respect to their queries. The accuracy expectation usually cuts across all the information retrieval methods as well as indexing techniques. Users expect to have accurate information no matter the technique that is used in the indexing process. However, it will be shown in the paper that the techniques differ in their accuracy. This research will compare the accuracy levels of the three mentioned techniques.

Despite the high preference of inverted files, suffix trees as well as signature files, they all have limitations. There are various challenges that are associated with each technique of indexing. The level of challenges associated with the application of each technique will be measured in the paper. A detailed comparison of the challenges will offer an understanding about which technique is more limited as compared to the others. Further, the benefits associated with the use of each technique will be outlined. Each indexing technique has benefits that are associated with its use. These benefits and advantages will be critically evaluated and compared. This comparison will offer information about which technique among the three has the most accrued benefits upon its application in the process of retrieving information. Finally, each indexing technique has an objective. The objectives of the various techniques differ across the techniques. This paper will also undertake a study of the main objectives of the techniques which they focus their performance.

II. Indexing Techniques

A basic definition of indexing was given in 1988 by Salton [4] as the facilitation of information retrieval accuracy by collecting, parsing and storing data. The accuracy facilitation is performed by use of various methods and techniques. As earlier stated, users need accuracy in the information retrieval process. The indexing process usually has an incorporation mechanism that allows use of concepts from various disciplines. It has been stated that there exists many information retrieval techniques with the common ones being inverted files, suffix trees as well as signature files. This section will discuss each technique into details as well as the way they work.

A. Inverted Files

Inverted files are defined as central components of an indexing algorithm in a search engine. The engine that searches information has a goal of query speed optimization. This means finding documents where a certain word occurs. Then the next step is developing a forward index. The index that is developed plays a role of storing the lists of words in every document. The document is then inverted, leading to a developed inverted index. Sequential iteration is usually required in order to query the forward index. In 2006, Belew [5] suggested that the iteration requires to be performed in each word and document in order to allow the verification of a document that matches the query. Technically, the resource in terms of time and memory that is required in the performance of such a query lacks an aspect of being realistic. However, the structure of the inverted files that is developed lists the documents per every word. This is done in place of listing the vice versa, where the words would be listed per every document. To perform a clear illustration of the inverted file concept, we assume an existing set of documents. Further, we assume that every document in the set is assigned a list that comprises of keywords. These keywords can also be referred to as attributes. We also assume that there are optional weights of relevance for every keyword. With the assumptions, the sorted list of keywords will be the inverted file. Each attribute will have a link to the documents that contains the specific keyword. Fig. 1 shows how the concept of inverted files works [5].

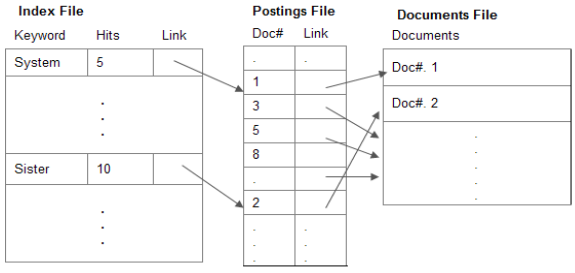

Fig. 1. Inverted files, the index file contains all words in the document and their index, the posting file contains a link to each word with the corresponding frequency, the document file contains the documents

According to Belew [5], the concept of inverted files is mostly used in library systems that are commercial. It is also used in libraries that belong to various education institutions. The reason for the popularity is that inverted files have enhanced efficiency in searching. Basically, the efficiency associated with inverted files is usually necessary when dealing with files that comprise large texts. This is the case for such institutions, which justifies their preference of inverted files.

1) Structures Used in the Inverted Files: There are several structures that are usually used in the implementation of inverted files. The most commonly used structures are sorted arrays, B-trees as well as tries. These structures will be discussed in this section. This will help in giving more information about the concept of inverted files. It will give more understanding of the relationship between inverted files and the various structures that are used in the files' implementation process.

a) Sorted Arrays: The implementation of an inverted file through this structure enables the file to support storage of keywords' lists in a sorted array. This includes several documents that are associated with each attribute. Further, it also includes a link to the documents that usually contain the attributes. Primary storage based systems use a binary search that is standard and are the most commonly used in searching a sorted array. On the other hand, systems that are based on secondary storage usually adapt the sorted array in conformity to their secondary storage's characteristics. Fig. 2. shows the structure sorted arrays as outlined by Barto, et al [1].

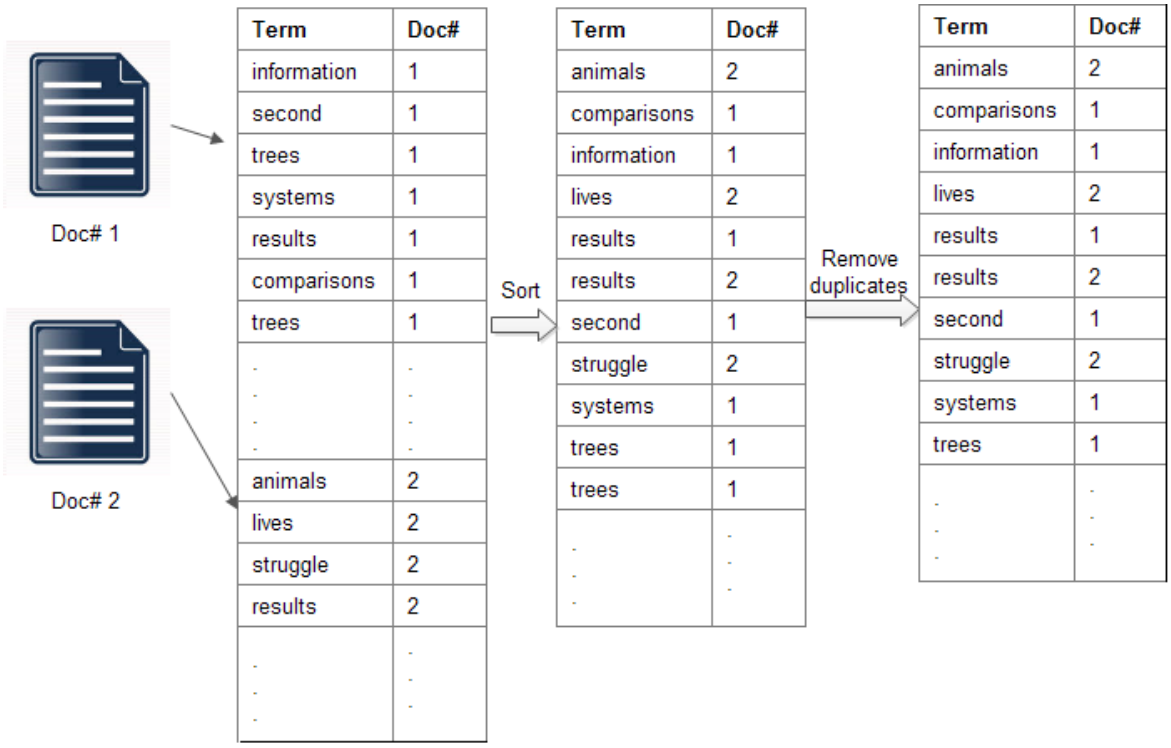

Fig. 2. Sorted arrays, The terms are sorted in lexical ascending order, then the duplicate words are removed

The figure contains two documents, Doc#1 and Doc#2. The words in each document are extracted and inserted in table. The words from both documents are sorted and possible duplication in single document is removed. For example the word trees are repeated in document 1, so it has been removed. The word results are found in both documents so it can't be removed.

The sorted arrays structure has an easy implementation process. It has a reasonable speed that enhances its performance. However, the structure is limited in that it requires frequent update of the index. The frequent updating sometimes is expensive.

b) B-trees: The most common type of the B-tree structure is prefix B tree. It utilizes word prefixes as the primary keys in an index of B-tree. This makes it well structured for the storage of indices that are textual. Every node that is internal usually carries a number of keys that are variable. The shortest word distinguishing the keys stored in the next level is usually named as the key. It is not necessary for the key to be a prefix in the index that is an actual term. The last level in the structure is known as leaf level, as shown in Fig 3. It carries the mandate of storing the attributes with the data associated with them. The order of every node of the prefix B-tree varies because there is dependence on attributes by the internal node keys as well as their lengths [6].

Fig. 3 shows simple Prefix B tree, the first level contains two keys, B and T. The two keys represent separators of the following leaves,

- Words beginning with letter less than or equal B such as Ar and Am,
- Words between B and T such as Co Fi and Ja,
- Words after T such as Un and Wa.

The second level represents other keys for the leaves beneath them, and so on. The last level contains the words of the documents with pointers to the corresponding document.

The B-Tree requires continuous update for maintenance of balance in the tree. The structure has a limitation in that it is not capable of handling many words within the same prefix. The B-tree method is broken down in cases of multiple words. The prefixes that are common usually call for division to avoid space wastage. B-trees usually occupy more space as compared to sorted arrays. However, updates are easier to implement and are faster in comparison with the sorted arrays [6].

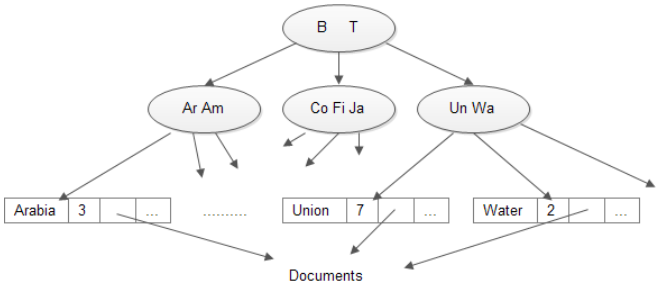

Fig. 3. Binary Tree indicating three levels with keys rpresenting each node

c) Tries: The structure's name was generated from the word retrieval. This structure is widely used to implement inverted files. Digital decompositions of the attributes are highly used by the structure in the representation of the same

keywords [1]. In structures of tries, keys associated to specific nodes are not stored by the nodes. There is similarity of the prefixes of a string in all the descendants of a particular node. There is no necessity of associating values with every node. Values are instead associated with leaves alongside several inner nodes that have correspondence to keys of interest [7]. Fig. 4 shows an example of the Trie structure.

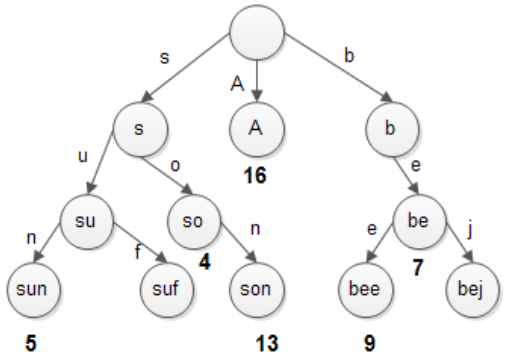

Fig. 4. An example of Trie structure showing no association of values with every node. Valures are instead associated with leaves alongside several inner nodes

The example above shows listing of some keys in the nodes while the values are indicated below the nodes. There is an arbitrary integer value associated with every English word that is complete. The example reveals a Trie as a finite automaton that is deterministic and tree shaped. There is generation of finite language by automaton tries. Further, compression of each trie into a state automaton that is deterministic acyclic is implemented as clearly shown in the above example. Tries are also fast in terms of the time used in implementing inverted files. It is also easy to implement as the example is easy to understand.

B. Suffix Trees

A suffix tree is a Trie that is compressed and contains suffixes of the given texts as the keys that belong to them as well as their values as the positions present in the text. The idea of compressing tries makes suffix trees be referred to as tries. Consequently, the sub-trees are referred to as sub-tries. The concept of suffix trees was developed in the year 1973 by Weiner [8]. The first online construction of suffix was to be developed by Ukkonen [9].

The running time associated with the algorithm was ranked as one of the fastest at that time. However, the algorithms were all linear-time for a size alphabet that was constant [10]. Generally, they had a running time of $O(n \log n)$.In 1997, Farach [11] designed an algorithm of suffix tree construction that had optimism for all alphabets. It was the first algorithm of linear-time for strings that were drawn from integers of the alphabet, in a range that was polynomial. This was the foundation of new algorithms that have been later developed in the construction of both suffix trees as well as suffix arrays.

Assuming a suffix tree for the string S and length n, the definition must meet several requirements [12]. Firstly, there must be exactly n leaves that are numbered from 1 to n in the tree. Every node that is internal must also have at least two children with the exceptional of the root. The labeling of the edge is done with non empty S substrings [13]. Any two edges that start out of a node should have strings labels that start with a different character. This condition means that it is not possible for a suffix to be a prefix that is proper for another. The digit that comes last in the data is a, and it appears two times in the data. Lastly, suffix S[i..n] is spelt out by the string obtained after the concatenation of string labels. These labels are the one present in the path of root to leaf.

Let us assume a string s = peeper. The non empty suffixes of the string will be peeper, eeper, eper, per, er and lastly r. Developing a suffix tree for the string peeper will comprise a compressed trie containing elements peeper, eeper, eper, per, er and r. The alphabet of the string is e, p, and r. This means that the radix of the trie that is compressed is 3. Fig. 5 based on [10] indicates a Trie for suffixes of the word "peeper".

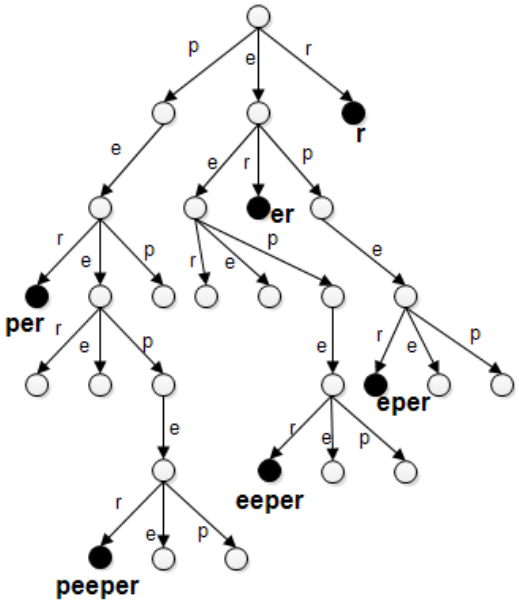

Fig. 5. Trie representation of the word "peeper", the compressed version the Trie is done with eliminating leaf white nodes

The use of suffix trees is applied when solving multiple string problems occurring in free text search as well as text editing. Suffix trees are also used in computational biology as well as other areas of application. However, there are several primary main applications of suffix trees. Firstly, they perform a search of a string in O (*m*) complexity. In such an application, *M* represents the substring's length. However, there is a mandatory requirement that there be time O (n) sufficient for building the string's suffix tree. Secondly, suffix trees are used to find the repeated string that is longest. It is also applied in the process of finding the common substring that is longest. Lastly, the longest palindrome in a string is found through application of suffix trees [14].

The above mentioned applications are useful as they expand the use of suffix trees. They enable them to be used in real life processes. For example, they are widely used in bioinformatics applications. They are also widely used in the search of DNA patterns as well as sequences of proteins.

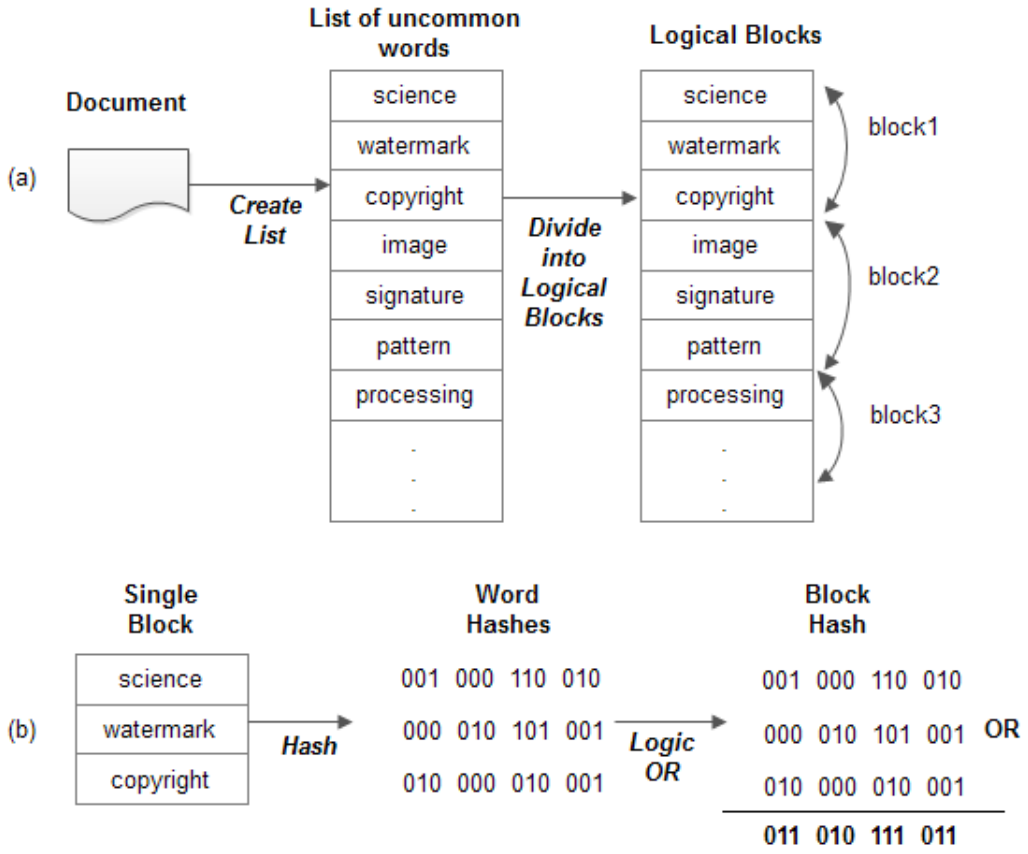

Fig. 6. The signature file is created by hashing every uncommon word to a given number of bits with fixed width

However, the sequences must be viewed as characters that are long stringed. According to Callan et al [15] the greatest advantage that makes suffix trees popular is their ability to make long searches with minimal mismatches. This makes them candidates to be used in data compression whey they enable the finding of data that is repeated. Lastly, most search engines also use suffix trees in the process of clustering data.

C. Signature Files

A signature file is a technique of indexing that usually creates a filter that is "dirty". An example of such a filter is the Bloom filter that keeps all the existing documents matching to the query entered by a user and also hopes to keep the ones that do not match the criteria. This is done through creation of a signature for every file which is typically a version of a hash code [16]. Therefore, a signature is an abstraction of a record which has been mapped. Signature files are generated through two main methods: Word signatures as well as Superimposed coding. The word signature approach involves hashing of identifiers which are basically words of a record to a bit pattern. The patterns or word signatures later form the record signature through concatenation. On the other hand, Superimposed generation of signed signatures involves hashing every uncommon word to a given number of bit positions, say *S* with a width that is fixed, say *F*, Fig. 6. Superimposing, through bitwise OR is performed on the resulting signatures for the generation of the record signature [17].

Fig. 6 shows a document is processed by creating list of uncommon words. A stop list of common words must be created to remove such words from further processing. Common words have no effect on defining the document character. The word list is divided into logical blocks. Each logical block, as shown in Figure (b), is hashed by hashing each single word. The block signature is obtained by logical ORing the word hashes. The main idea of Bloom Filter [18] k-th order Bloom filter has k independent hash functions $H_1(x)$, $H_2(x), \ldots\ldots H_k(x)$, that maps a word to a hash value in the range 0 to N-1, where N is the length of the hash bits. Formally,

$$H_i(X_j)=y \ , \quad 1 \le i \le k; \quad 1 \le j \le D; 0 \le y \le N-1; \qquad (1)$$

Where, *Xj* is the *jth* word in the uncommon list, *D* is the number of uncommon words in each document. The following procedure is applied

1) A has table of N bits size is created and all of its bits are set to zero.

2) For each word in the word's list, its k hash values are calculated, and accordingly the corresponding bits are set to 1. Thus for example if $H_i(X_j)=68$ for some (I,j), then the sixty-eighth bit of the hash table is set to 1, if the bit is already 1, then no change will be done.

When searching for specific keyword, the keyword's k hash values are calculated. If all the corresponding values in the

hash table are all set to 1, then a matching is found, otherwise no match.

A signature approach that is naive would involve a uniform and a random hashing. Same *S* bit positions are hashed by any given *n*-gram [10]. A possibility exists that two *n*-grams that are different will most probably hash too the same position of bit. The occurrence of hashing same bit position by different *n*-grams is referred to as collision. The possibility of collision is contributed by the fact that the *F* chosen in most cases is usually less than the number of unique *n*-grams in total.

III. Comparison of Indexing Techniques

This section will focus on the comparison of the three discussed techniques of indexing in information retrieval. It is clear from the previous discussion that the techniques differ in many aspects. This is despite the fact that they work towards yielding same results. All the techniques are aimed at indexing and undertaking successful information retrieval. However, the approaches that are used by then various techniques are different. The techniques also vary in their performance as well as their stability. The techniques also vary in terms of their limitations as well as advantages. This section will critically focus on the performance and stability of each technique and compare them with the rest. It will also compare the limitations of each technique and make a detailed comparison. This will enhance the understanding and knowledge about the three indexing techniques.

A. Performance Comparison

The performance comparison between inverted files, suffix trees as well as signature files can take several dimensions. However, the main parameter used in determining the performance of the techniques is the processing time of the various techniques [19]. This is the time that is taken for a system using a particular technique to give response to a query raised by a user. The comparison of the performance between the various indexing techniques in this section will be based on the response time. This will help to effectively determine the performance of each indexing technique.

1) Inverted files: The structure was developed with a primary goal of optimizing the speed of the query. The structure's performance is based on an iteration of a developed inverted index. Querying the forward index is the main reason as to why the iteration is necessary. However, as discussed, it would be technically unrealistic to take the time required for the iteration. Inverted files have several approaches of performance that enhance the response time that is required.

Firstly, developed inverted files usually list the documents on basis of "per every word". Secondly, inverted files have a special performance approach that is known as skipping. This involves introduction of synchronization points which are additional locations that usually offer a platform for the commencement of decoding to the inverted list [19]. The index in the inverted files contains both the difference in document number as well as the difference in bit address. This results to the capability of inverted files to be stored as sequence that is compressed. It is the compression capability that enhances the performance of inverted files and highly saves on the processing time [2].

The results of an experiment on 100 documents from the internet with applying skipping on the inverted files retrieval technique is shown in Fig. 7. The experiment executed some queries on the documents, the figure shows the CPU time required to process those queries.

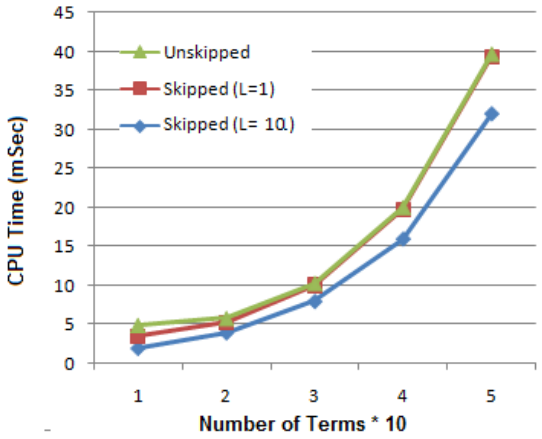

Fig. 7. The effect of skipping on the inverted files indexing performance

In the figure, L represents the skipped index

2) Suffix Trees: A suffix tree has already been expressed in the previous section as a compression of sub tries. Therefore, the performance of suffix trees is based on basis of compression. It is clear that suffix trees support compression and indeed perform when compressed. The processing and running times associated with the algorithm are one of the fastest. Its performance is ranked as one of the most efficient taking processing time as a parameter. The running time of suffix trees is generally given as $O(n \log n)$.

There are several reasons for the efficient performance in terms of time. The first reason is the support of insertion. This is put as a condition in any dynamic suffix tree. The second reason is the ability to perform deletion. Lastly, suffix trees carry special capability to perform modification of strings. These are the unique traits with suffix trees that makes the technique's performance to stand out among all other indexing techniques. There is no other indexing techniques whose performance involves insertion, deletion as well as strings modification in the manner that suffix trees perform.

3) Signature Files: The performance of the signature files is largely based on unique signature development for every file. This development of signatures as explained in the previous section is done through word signatures as well as superimposed coding. An evaluation of the procession time as a parameter to determine the efficiency in performance reveals several things. Firstly, there is a possibility of slowness as compared to other techniques due to the concatenation due to the word signatures. Secondly, the time taken to respond to queries raised by users can increase when using signature files technique due to the sequential nature of the files [20]. This is mostly the case for the files that use superimposed coding other than word signatures.

There are special features associated with the technique that make it fast and pass the efficient performance test. The technique like others supports compression in order to enhance its performance. This enables the technique to be well placed to improve the time for processing objects in the indexing process. The technique also utilizes partitioning. This is usually the most unique feature that is associated with signature files as a technique of indexing. There are not many techniques that are known for both vertical as well as horizontal partitioning.

B. Stability Comparison

The aspect of stability is explained in the ability of various techniques to handle the files that contain information which the users are looking to retrieve. Stability in the field of information retrieval is simply the variance that is associated with the results of the queries of various queries. This means the relationship between the objects provided by a certain technique with the query that is entered by a user. The relevance of the results in respect to the queries forms the basis of stability discussion. This section will compare the results that are given upon the use of various techniques. This will shed more light about the stability of suffix trees, signature files as well as inverted files. Generally, variance is usually measured by undertaking a balance between the risks and rewards. The risks are the threats to a technique in performing and giving the desired results [3]. On the other hand, the rewards are the desired objects that can be obtained by using a certain technique of indexing in the process of retrieving information. However, this is usually challenging as there must be a clear way of determining the rewards as well as the risks. The study of stability best illustrated in a risk/reward curve as shown in Fig. 8. Algorithm A dominates algorithm B. The figure shows two algorithms that appear identical in terms of mean average precision (MAP) gain may have very different risk profiles.

1) Inverted Files: Inverted files have a considerably desirable trade off between the risks and the rewards. They are seen as one of the most stable indexing techniques. The index construction in an inverted file was explained by Kanaan et al [20] as shown in Fig. 9.

The diagram shows the possibility of deriving an inverted file upon completion of the Trie structure. The structure as indicated enables access to the file in main memory. This is the basis of the strength and stability of inverted files. The reason is that every entry has a reference position of the posting file which is usually held in storage that is secondary. This brings out an aspect of back up and easy tracing of entries. This has few risks associated with it and results in a stable indexing process.

2) Suffix trees*:* A suffix tree is built with a high threshold of stability. The construction of suffix trees is performed with the principle that every string is supposed to be padded a marker symbol that is out of alphabet and unique. This serves the purpose of ensuring that any suffix in the construction does not become a substring of the other. Since the building of the suffix trees involves leaves, every suffix has a representation by a leaf that is unique. This means that the reward risk assessment is passed by suffix trees. The risk that is associated with most techniques is false results. However, the suffix trees eradicate that by ensuring that every suffix is served by only one leaf. Therefore, stability is maximized in suffix trees.

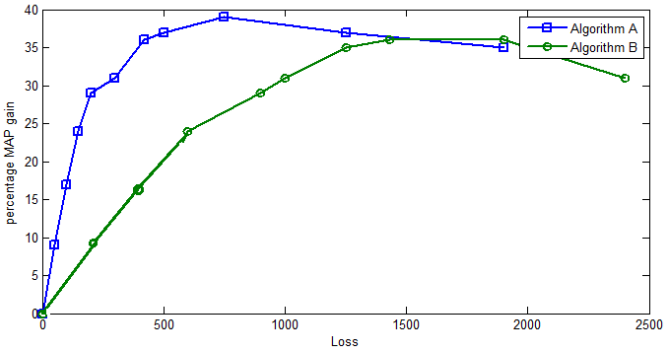

Fig. 8. Risk/reward curve showing query expansion. The curve shows two retrieval algorithms compared in performance. Algorithm A performance is better than Algorithm B.

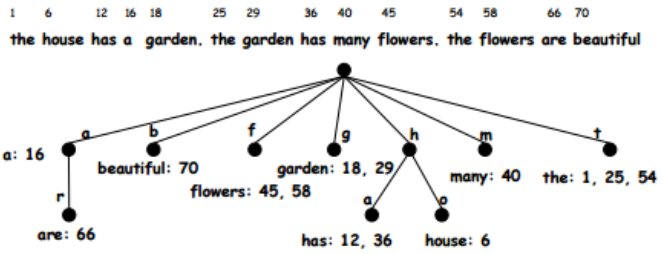

Fig. 9. The index construction in an inverted file. The figure based on [20]

C. Limitations Comparison

Despite the many strong points that the discussed indexing techniques have, they usually have various limitations that make it hard for them to perform optimally. However, the indexing techniques are not limited in the same way. They usually have different limitations in their performance. This section will focus on the limitations of every indexing technique. The limitations will then be compared to offer a synopsis of the limitations and the ability of each technique to overcome the challenges.

Inverted files have their share of disadvantages that usually pose a challenge to their efficiency in offering optimum indexing. This in turns affects their application and usage by most users across the world. Firstly, the technique has a limitation in that there is difficulty in the update of insertions especially of new records [1]. This usually requires moving of proportions of files that are large. Secondly, random access of any system by use of the inverted file technique is usually slow. In some cases, files are usually considered to be organized in a sequential manner even when there is no order to a certain key. This creates the possibility of false objects because sometimes acquisition date can be regarded to as the key value.

On the other hand, Robertson and Sparck [21] suggest that suffix trees are limited in that they usually require a lot of space due to the nature of their construction. The internal pointers in the tree usually require more space for storage. This

is in comparison to most of the other techniques that consume considerably less space. Suffix trees also have a challenge in that it is necessary for them to be built in an order of reverse. This means that characters have to be added from the input's end. Lastly, the nature of the tree works against it because the string's length can be a single variable. This could occur within the same class that the segments of the leaf belong. This works against it because the side to side co existence of suffix trees could be impossible because there would be similarity of the class of leaf segments.

Lastly, signature files indexing technique has a share of practical problems in its performance. There are many methods that are used by signature files to enhance operation. The variety in the methods of operation also expands the limitations of the signature files. Firstly, signature files' performance is known to deteriorate as the files grow [22]. This simply means that signature files have a limitation of performance in files that are large. Secondly, the technique has a disadvantage because in case the keywords number in every document is large, then a huge hash table must be made. It might also lead to usual queries touching a proportion of the database that is large. Lastly, the signature files technique is limited in handling queries that are not conjunctive [23]. This difficulty in dealing with non- conjunctive queries limits the performance of signature files. This mostly happens when signature files utilize the method of Gustafson's in the process of indexing.

IV. CONCLUSION

The research in this paper has clearly achieved a critical analysis of indexing techniques. It has offered information about the construction of various techniques such as inverted files, Suffix trees and Signature files. In addition the paper introduced detailed structures that make up these techniques. The research has also given more understanding of the building of the structures and the way that they work. The paper has detailed few benefits that are associated with the use of every technique. The speed, as well as the space that is required for the various techniques to optimally operate, has been outlined. This has provided the basis of the comparison that has been done between the various techniques.

The comparison done in this paper has taken the dimension of performance, stability as well as limitations. The performance of inverted files, suffix trees, as well as signature files, is compared in the paper by using the processing time as the parameter. The paper has done a comprehensive comparison of the time taken for every technique to respond to various queries that are raised by users. On the other hand, the stability of every technique of indexing has been discussed and compared to other techniques. The parameter used for the stability discussion is the measure of rewards and risks associated with every technique. Lastly, the paper has undertaken a comparison of limitations and challenges of every technique. This comparison has helped in knowing the challenges a user would get by using a certain indexing technique.

TABLE I. COMPARISON OF INVERTED FILES, SUFFIX TREES AND SIGNATURE FILES INDEXING TECHNIQUES

Indexing Technique	Capabilities	Limitations
Inverted Files	• Compression • Skipping • Iteration	• Difficult update of insertions • Slow random access
Suffix Trees	• Compression • Insertion • Deletion • Modification	• Requires a lot of space • Require reverse building • String's length can be a single variable
Signature Files	• Compression • Vertical Partitioning • Horizontal Partitioning	• Limited performance for large files • Must build huge hash table in case of large keywords in each document • Difficulty in dealing with non-conjunctive queries.

REFERENCES

[1] A. Barto, et al." Learning to Act Using Real-Time Dynamic Programming". *Hoboken:Rutledge Press* pp.112-123.(2015)

[2] R. Baeza-Yates and B. Ribeiro-Neto, "Indexing Techiques," in *Modern Information Retrieval*. A. Wesley, Ed. New York: Wiley, (1999).

[3] K. Darwish, "Building a Shallow Arabic Morphological Analyzer in one Day," in *Acl Workshop on Computational Approaches to Semitic Language*. Illinois: Sage, PP. 47-57. (2002)

[4] G. Salton, "Automatic Text Processing," in *The Translation Analysis and Retrieval of Information by Computer*. Washington: Cambridge, Addison-Wesley publishers, 3(2), pp. 45-70. (1988)

[5] R.K. Belew, "Adaptive information retrieval," in *Machine Learning in Associative Networks*. Michigan: University of Michigan Press., pp. 78-83. (2006)

[6] BAYER, R., and K. UNTERAUER "Prefix B-Trees." *ACM Transactions on Database Systems*, 2(1), 11-26. (1977)

[7] Manolis Terrovitis, Spyros Passas, Panos Vassiliadis, Timos Sellis " A Combination of Trie-trees and Inverted Files for the Indexing of Set-valued Attributes " *CIKM'06, ,* Arlington, Virginia, USA. November 5–11,(2006)

[8] Weiner, P. "Linear pattern matching algorithms" *14th Annual IEEE Symposium on Switching and Automata Theory*, pp. 1–11, doi:10.1109/SWAT.1973.13. (1973),

[9] Ukkonen, E. "On-line construction of suffix trees" *(PDF). Algorithmica 14 (3): 249–260.* doi:10.1007/BF01206331. *(1995).*

[10] H. Chen, "Machine learning for information retrieval" in *Neural Networks, Symbolic`Learning, and Genetic Algorithms*. New York: Elsevier Science Inc., pp. 17-31. (2008)

[11] Farach, Martin "Optimal Suffix Tree Construction with Large Alphabets" , *38th IEEE Symposium on Foundations of Computer Science (FOCS '97)*, pp. 137–143. (1997)

[12] Wikipedia, "Suffix Trees", *Online https://en.wikipedia.org/wiki/Suffix_tree* (2014)

[13] S Ghwanmeh et al. "Comparison Between Inverted and Signature Files Based on Arabic Documents," in *International Journal of Applied Science and Computations*. Hobeken:Rutledge Press., pp. 174-193. (2005)

[14] Sartaj Sahni "Data Structures, Algorithms, & Applications in Java, Suffix Trees" *Online*, (1999)

[15] J. Callan et al. " TREC and Tipster Experiments with Inquiry," in *Information Processing and Management*. New York: Cambridge University Press., pp. 117-122. (2008)

[16] W. Croft and D. Harper, "Using probabilistic models of document retrieval without relevance information". *Maidenhead: Open University Press,* , pp. 12-24. (2009)

[17] William B. Frakes and Ricardo Baeza-Yates " Information Retrieval: Data Structures & Algorithms" *Online*, (2015).

[18] William Stallings, Lawrie Brown "Computer Security: Principles and Practice, 3rd Edition" *Pearson publications*, (2014)

[19] G. Kanaan, "Comparing Automatic Statistical and Syntactic Phrase", in *Indexing for Arabic Information Retrieval*, Chicago: Cambridge, , pp 34-60. (1997)

[20] G. Kanaan, et al. "Indexing for Successful Retrieval," in *Information Retrieval Techniques.* Illinois: Oxford University Press., pp. 18-25(2006)

[21] E. Robertson and K. Sparck, "Relevance Weighting of Search Terms," in *Journal of theAmerican Society for Information Science*. Washington: Rutledge., pp. 45-54. (2004)

[22] R. Attar and A.S. Fraenkel, "Local Feedback in Full-Text Retrieval Systems". *Washington:Sage Press*., pp. 53-67.(2007)

[23] G. Salton and M. J. McGill, "Indexing Techniques Comparison," in *Introduction to modern information retrieval*. New York: McGraw-Hill., pp. 67-89. (2003)