International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Enhancing Big Data Retrieval: A Comparative Analysis of Standard and Three-Level Indexing Techniques Using Dictionary Words Dataset

Tara Prakash Gowdar[1], Dr. Paras Nath Singh[2]

Submitted: 29/01/2024 **Revised**: 07/03/2024 **Accepted**: 15/03/2024

Abstract: Utilizing a large dataset of dictionary words, this study examines the way three-level indexing approaches perform when compared to traditional indexing techniques for improving big data retrieval. The research is concerned with assessing the retrieval effectiveness, efficiency, and scalability of these indexing systems in the context of managing huge datasets. The dictionary terms dataset will be subjected to standard indexing and three-level indexing as part of the experimental framework, and the retrieval accuracy and efficiency metrics will be subjected to a thorough comparison study. Particularly in the context of linguistic datasets, the findings provide helpful information on optimizing big data retrieval strategies. This study emphasises the need of sophisticated indexing techniques for organizing and gleaning useful data from huge databases.

Keywords- *Big Data Retrieval, Indexing Techniques, Three-Level Indexing, Standard Indexing, Comparative Analysis, Dataset, Dictionary Words, Retrieval Performance, Efficiency, Scalability, Information Extraction, Linguistic Datasets.*

1. Introduction

In the context of the English language, word indexing, and search are essential because they make it possible to retrieve, analyse, and understand content quickly. The ability to index and search words is essential for people, corporations, and researchers alike in a world where there is an abundance of textual data. Steps in the big data indexing are shown below. Word indexing primarily enables the organised storage and classification of words. An index is made by giving each word a special number or address, serving as a guide for finding terms within a corpus of text. This indexing procedure gives order to the otherwise disorganised world of language, enabling rapid information access when required. It doesn't matter if it's a book, a website, or an entire database—indexing makes sure that words are arranged logically and methodically. Furthermore, efficient search operations are made possible via word indexing. Users can enter keywords or phrases to find relevant information from huge collections of documents or web pages with the aid of search engines and their algorithms. This talent is especially useful for academic study since it allows students to uncover pertinent articles, sources, and references more quickly, which speeds up their understanding of difficult subjects. People can use search engines to discover new ideas, explore different topics of interest, and find solutions to questions in their daily lives. Word indexing assists with information access but also aids in the study and comprehension of textual material. In a text corpus, when we index words, we will be able to investigate the frequency, patterns, and relationships between keywords. Some applications of this skill are in linguistics, computational linguistics, natural language processing, and information retrieval. Acquiring insights into language usage, tracking trends, performing sentiment analysis, and constructing advanced language models are useful for researchers. For improving language understanding, word indexing, and search is critical. Individuals can quickly search unknown terms or concepts utilising dictionaries, online resources, or digital platforms that provide fast definitions and explanations. This ability to quickly seek the meaning and context of words greatly benefits language learners, students, and professionals, encouraging good communication, writing, and reading skills.

[1]*Research Scholar, CMRIT, Bangalore, Karnataka, India*[1]
[2]*Professor, CMRIT, Bangalore, Karnataka, India*[2].
tara.enner@gmail.com, PNSingh2810@gmail.com

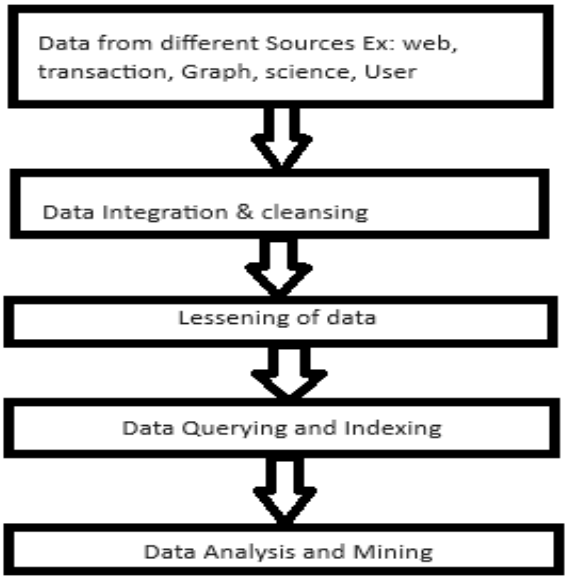

Fig 1: Big data indexing technique.

For exploring and understanding the English language word indexing and searching are important instruments. Effective information retrieval, knowledge acquisition, and language comprehension can be done as indexing & search give structure, accessibility, and analytical capabilities. The value of word indexing and search in our digital age only grows, as the volume of written content grows drastically, allowing us to make the most of the vast amount of information at our fingertips. It is possible to investigate the frequency, patterns, and relationships between keywords in a text corpus if indexing can be applied. A wide range of applications in domains such as linguistics, computational linguistics, natural language processing, and information retrieval are applications of this analytical power. It enables academics to learn more about how language is used, follow trends, carry out sentiment analysis, and create sophisticated language models. Additionally, word indexing, and search is essential for improving language understanding. People can quickly seek up words or ideas using dictionaries, internet databases, or other digital tools that offer rapid definitions and explanations. Language learners, students, and professionals greatly benefit from this ability to quickly seek for the meaning and context of words, which promotes good writing, reading, and communication skills. For navigating and comprehending the English language, word indexing, and search are crucial tools. They offer organisation, accessibility, and analytical skills that support effective knowledge acquisition, language understanding, and information retrieval. Word indexing and search are more important in our digital age as the amount of textual data keeps increasing exponentially and allows us to fully utilise the abundance of information at our disposal. Even though the word indexing, and search algorithms used today are efficient, there are some issues that needs to be resolved. First off, these algorithms frequently require sorted strings to produce a speedier search result, which adds another level of temporal complexity. Unfortunately, the algorithms lack the capacity to extrapolate data or make educated judgments, which would improve their effectiveness. Furthermore, it is difficult to do speedier search operations since terms in the English lexicon lack distinctive numbers or identifiers. The lack of a consistent system for indexing new words makes search results less accurate and slower as more words are introduced to the language. The way in which homophones and synonyms are handled by the current word indexing and search algorithms is another problem. While synonyms have similar meanings, homophones are words with the same spelling but different pronunciations. Due to the algorithms' frequent inability to discern between these changes, search results are frequently unclear. For instance, depending on the context, a search for the word "bank" can turn up information about banks of rivers or financial institutions. The accuracy and relevancy of search results would increase if the algorithms were enhanced to take contextual clues and semantic linkages into account.

Another difficulty is the absence of support for morphological differences. English words can change morphologically in several ways, including verb conjugation, pluralization, and distinct tenses. However, the current algorithms frequently regard these variants as separate. The current algorithms, however, frequently regard these variations as different entities, necessitating separate searches for each type. The user experience is hampered by this redundancy in addition to slowing down the search process. By identifying and renormalizing these morphological changes, stemming or lemmatization approaches would enable more effective searches. Accurate word indexing and search are also

significantly hampered by the problem of linguistic ambiguity. The algorithms struggle to accurately grasp the nuances and various meanings of the English language. Multiple-meaning words, or polysemous words, can be confusing and produce inaccurate search results. Contextual analysis and semantic disambiguation techniques must be incorporated into the algorithms to better comprehend the intended meaning behind the words and offer more relevant results.

2. Methodology

In this part, we outline the technique used to analyse the effectiveness of four distinct indexing systems—B-tree, Inverted Index, Hashing, and Trie—when used in conjunction with the word-searching algorithm that is provided. The goal is to examine and assess how well each indexing approach performs in terms of search process duration, time complexity, and space complexity.

a) Data Collection and Preprocessing

1) Data Source: The Reuters corpus, a well-used dataset containing a selection of news stories, is utilised as the data source for assessing the indexing methodologies. Using the Natural Language Toolkit (NLTK) library, one can access the corpus.

2) Algorithmic Combination: Combination of the supplied algorithm with each of the four indexing systems—B-tree, Inverted Index, Hashing, and Trie—as described in Section 1. The preprocessing stages must be changed as part of this integration to create the appropriate data structures for indexing.

b) Implementation and Evaluation

1)B-tree indexing: By combining the technique with a B-tree data structure, words can be stored and retrieved quickly. The B-Tree Node class, which is defined in the code, is used to build the B-tree. Throughout the search process, time duration, time complexity, and space difficulty are noted.

2)The inverted index is created by mapping each term to where it appears in the documents. To create the inverted index, the preprocessing phase is modified. The word is looked up during the search process using the inverted index. Performance indicators are gathered, including measurements for complexity and time length.

3) Hashing: Words are mapped to specified indices in an array-like data structure via hashing. To function with the hash-based indexing system, the algorithm is modified. The search procedure entails looking up the target term in the hash table. Measurements of complexity and execution time are kept.

4) Indexing Trie: To store and look up words quickly, a Trie data structure is created. The Trie is populated by adapting the preprocessing stage. Finding the target word requires searching through the Trie. Measurements are made of duration, intricacy, and space.

c) Performance Comparison

1) Comparison of search times: The amount of time it took for each indexing method to find the specified term was noted. The time module is used to calculate the execution time in milliseconds.

2) Time Complexity Analysis: Each technique's theoretical time complexity is covered. While Inverted Index and Hashing offer constant-time lookup with varying overheads, B-tree and Trie have logarithmic search time complexity.

3) Analysis of Space Complexity: Each indexing system's space complexity is covered. While hashing and Trie include space for tree structures and hash tables, respectively, B-tree and Inverted Index need additional space for data storage.

d) Experimental Setup

Hardware configuration and software Environment:

OS Name: Microsoft Windows 11 Pro

Programming language: Python 3.8

Libraries & framework used: NLTK 3.5, sorted containers 2.3.

e) Data Collection and Analysis

Data Points: From the Reuters corpus, a representative sample of search queries is chosen for analysis.

Execution and Metrics: For each query, the four indexing strategies are used to carry out the search operations. For analysis, the execution time and resource usage are logged.

Results Interpretation: For each indexing strategy, the results are graphically displayed and described in terms of time duration, time complexity, and space complexity.

f) Ethical Considerations

Data Usage: The Reuters corpus is a publicly available dataset, and its usage adheres to ethical data usage guidelines.

Code Implementation: The code implementation for each indexing technique respects software licensing and copyright considerations.

The following sections of the research paper will delve into the results and discussion of the performance comparison, drawing conclusions on the effectiveness of each indexing technique in the context of the provided algorithm and outlining potential implications and future research directions.

3. Methodology for Data Collection and Corpus

The English language corpus offered by the Natural Language Toolkit (NLTK) dataset is used to test the methodology. The English language corpus offers a broad range of English texts, including novels, essays, and web content, making it appropriate for a variety of language analysis applications. The corpus includes a sizable and varied collection of English texts that reflect various tenses, genres, and subjects. It is appropriate for analysing the qualities and traits of English words since it offers a complete and representative sample of the English language.

3.2 Methodology for Indexing and Search Algorithms:

General algorithm for search:

1) Build the indexing technique 1 (3-level indexing technique) and indexing technique 2 (It may be trie, hashing, inverted indexing, B-Tree).
2) Preprocess and index data.
3) Search using technique 1 then search using technique 2.
4) Retrieve results.
5) Measure and report performance.

Here's an explanation of each step in the general algorithm for search:

1) Build the Indexing Technique 1 and Indexing Technique 2:

- In this step, you select one indexing technique from among trie, hashing, inverted indexing, B-tree and use 3-level indexing technique to efficiently organize and store your data for quick retrieval. These techniques can be chosen based on the specific requirements and characteristics of your dataset.

2) Preprocess and Index Data:

- In this step, dataset is prepared for indexing. This includes cleaning, normalizing, and transforming the data to make it suitable for the chosen indexing techniques.

- Then build the actual index structures for both Technique 1 and Technique 2. This involves creating data structures, populating them with the data, and organizing them in a way that allows for efficient searching.

3) Search Using Technique 1, Then Search Using Technique 2:

- After indexing the data, the search is performed in two stages: first using Technique 1 and then using Technique 2.

- The search in Technique 1 leverages the primary indexing method (e.g., 3-level indexing) to quickly identify potential matches or candidates in the dataset.

- If the search in Technique 1 doesn't yield the desired results or needs further refinement, you proceed to the second stage of the search using Technique 2.

- Technique 2 is used to refine the search results obtained from Technique 1. It's a complementary indexing method designed to handle specific search criteria or scenarios.

4) Retrieve Results:

- In this step, you retrieve the search results from both Technique 1 and Technique 2.

- You may need to combine or compare the results from both techniques to ensure accuracy and completeness in the final set of results.

- Depending on your application, you might have different strategies for presenting or ranking the results.

5) Measure and Report Performance:

- Finally, assessment takes place for of the search process in terms of speed, efficiency, and accuracy.

- The time taken to execute each search stage are measured (Technique 1 and Technique 2) and recorded.

- The correctness of the results is evaluated and how well they match the search criteria can be observed.

- The performance metrics are reported, allowing to analyse which indexing technique performed better for this specific use case.

This general algorithm for search provides a structured approach to optimize search operations by combining different indexing techniques, which can be especially valuable when dealing with large and complex datasets. It helps improve search efficiency and accuracy by leveraging the strengths of multiple indexing methods.

I. Introduction to English Word Indexing Techniques

Information retrieval systems use a variety of indexing techniques and algorithms to effectively index and retrieve English words. Here are several methods that are frequently used:

1) One essential method utilized by information retrieval systems is inverted indexing. It entails building an index system that associates words with the texts or chapters where they appear. Each word has a list of document identifiers or

pointers attached to it, making it easy to quickly search for and retrieve documents that include a certain phrase. Due to its effectiveness in managing huge text collections, inverted indexing is commonly utilized.

2) The Term Frequency-Inverse Document Frequency (TF-IDF) method is A statistical weighting method called TF-IDF is used to evaluate a word's significance within a collection of documents. Each word is assigned a weight depending on its frequency in a document and its inverse frequency across the entire collection. Words that are more prevalent in a particular text but less prevalent across the board are given higher weights. In ranking algorithms, TF-IDF is frequently used to emphasize significant phrases in search results.
3) N-Gram indexing involves breaking down words into contiguous groups of N letters, or subwords. For managing imperfect string matching and incomplete matches, this method is especially helpful. The words "hello," for instance, would be indexed as "hel," "ell," and "llo" in a trigram index. Effective word searching for words with missing characters, variants, or misspellings is made possible by N-Gram indexing.
4) Algorithms: Compression techniques are used to shrink the index, enabling quicker retrieval, and requiring less storage. Variable Byte Encoding, Golomb coding, and Elias Gamma coding are a few common compression algorithms. These techniques take advantage of word frequency statistics and spaces between posting lists to achieve efficient compression without sacrificing retrieval speed.
5) Language-Specific Techniques: A few indexing strategies consider the linguistic nuances of the English language. Stemming algorithms, for instance, condense words to their basic forms (e.g., "running" to "run") while capturing word variants. Lemmatization takes it a step further and reduces words to their dictionary or basic form (for example, "mice" becomes "mouse"). These methods aid in overcoming word morphological problems and enhancing search recall.
6) Advanced Ranking Algorithms: Ranking algorithms are just as important in evaluating the relevancy of search results as indexing techniques are. Search results are scored and ranked using a mix of phrase frequency, document length, and inverse document frequency by algorithms like Okapi BM25 (Best Matching 25). The value and relevance of pages are sometimes determined using techniques like PageRank and link analysis, which were initially developed for web searches.

These are only a few illustrations of the indexing techniques and algorithms employed by English word information retrieval systems. The features of the text collection, the search needs, and the desired trade-offs between indexing effectiveness, retrieval accuracy, and storage considerations are only a few of the variables that influence the technique choice.

Although the current approaches to word indexing and search in information retrieval systems have shown promise, they also have drawbacks and room for development. Here is a breakdown of their advantages, drawbacks, and prospective improvement areas:

Strengths:

1. Effective Retrieval: Current methods offer effective retrieval of pertinent texts or passages, including inverted indexing and compression algorithms. They make it possible to obtain information quickly via keyword searches, making them appropriate for managing big text collections.

2. Scalability: A variety of indexing techniques are built to manage scalability, enabling effective indexing and retrieval even for enormous amounts of textual material. Compression algorithms and distributed indexing are two methods for managing and analysing massive document collections.

3. Language-Agnostic Approach: Several approaches, such TF-IDF and inverted indexing, are language-agnostic and so usable with many languages, including English. Due to their adaptability, they can be widely used in a variety of language contexts.

4. Robustness to variants: Methods like stemming algorithms and N-Gram indexing consider word variants as well as partial matches, misspellings, and morphological variances. They do this by considering various word renderings, which improves the memory and coverage of search results.

Limitations:

1. Semantic Gap: Current methods frequently concentrate on surface-level characteristics like word frequencies and character sequences but fall short of accurately capturing the semantic linkages between words. When dealing with polysemous words or intricate semantic settings, this restriction may lead to less precise search results.

2. Managing synonymy (words with similar meanings) and polysemy (words with many meanings) continue to be difficult. Retrieval algorithms may have trouble correctly identifying and retrieving pertinent information

when different words are used to represent the same idea or when a single word has several meanings.

3. Contextual Understanding: Generally speaking, contextual information like word relationships, grammatical structures, or semantic meaning is not fully captured by existing approaches. As the context in which words appear greatly influences how they are interpreted, this restriction may have an impact on the accuracy and relevancy of search results.

4. Handling Noisy or Ambiguous inquiries: When handling noisy or ambiguous inquiries, when the user's intent is unclear, existing solutions may have trouble. These issues can be resolved by using strategies like query extension, relevancy feedback methods, or user engagement.

Areas for Development

Some areas for development in indexing & searching are:

1. Semantic Search: To achieve more accurate and context-aware search results advances in strategies for capturing and exploiting semantic linkages between words can be used. To reduce the semantic gap and increase retrieval accuracy we can use word embeddings, knowledge graphs, and semantic analysis together.

2. Personalization and User Intent: Research might concentrate on figuring out user intent and tailoring search results to each user's preferences, browsing habits, and demographic data. The usefulness of search results may be improved by incorporating machine learning techniques and user feedback methods.

3. Integration of multimodal data sources in the indexing and search process is crucial as information retrieval expands beyond text to incorporate images, audio, and video. Search experiences that are more thorough and multimodal can be made possible by techniques that combine textual and visual/auditory features.

4. Real-Time Indexing and Updating: Research can examine methods for indexing and updating information instantly as data streams and real-time information become more common. In dynamic situations, retrieval systems would then be able to deliver timely and current search results.

Improvements can be made to word indexing and search algorithms in the English language domain to improve their accuracy, relevance, and contextual understanding by addressing these research gaps and concentrating on the shortcomings of current methodologies.

4. English Word Indexing Techniques

To efficiently organise and retrieve data from huge datasets, indexing techniques are essential. To enable quicker and more precise word searches, a variety of indexing techniques have been developed for English terms. We will examine and contrast four well-known indexing methods in this discussion: the inverted index, the B-tree, the hash-based index, and the trie.

1. Inverted Index: One popular method for retrieving text is the inverted index. Each distinct word in the dataset is mapped to a list of document identifiers or locations where the word appears using a dictionary. Quick word-based searches are made possible by inverted indexes, which quickly spot documents that contain certain terms.

Principle and Algorithm:

Tokenization: The text is divided into individual words or tokens.

Term Frequency: Count the number of times each term appears in a document.

Inverted Index Construction: Create a mapping from terms to documents or positions to create an inverted index.

Example: Think of a collection of three texts: "Document A: The sky is blue," "Document B: The ocean is vast," and "Document C: The sky meets the ocean." "Document A" and "Document C" would be listed as instances of the word "sky" in the inverted index.

Table 1: Inverted Index

Token	**Document Id**
The	A, B, C
Sky	A, B
Is	A, B
blue	A
meets	C
Ocean	B, C

vast	B

Advantages:

Word-based searches that work well.

Because it stores term-document associations, it is memory-efficient.

Disadvantages:

Ineffective at finding phrases.

Large vocabulary usage consumes a lot of RAM.

Performance Inverted indexes excel at speedy word retrieval but may take longer to respond to queries with several terms or phrases.

2. B-Tree: Self-balancing tree structures known as B-trees are utilised for indexing. They offer search, insertion, and deletion operations with logarithmic time complexity, making them ideal for database applications. A B-tree's ability to have numerous keys and child nodes makes for a balanced structure.

The Algorithm and the Principle:

Node Splitting: When a node has more keys than it can handle, it splits into two nodes.

Balancing: Ensures that the tree's height is maintained in a balanced manner.

Search: Uses a binary search strategy to find keys.

For instance, a B-tree with the keys [4, 8, 12, 16, 20] could speed up key searches.

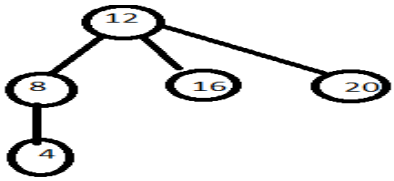

Fig 2: B-tree

Advantages:

Structure with balance for effective operations.

Appropriate for ordered data and range queries.

Disadvantages:

Owing to node overhead, and not being memory efficient.

During insertions and removals calls for reorganisation. B-trees are good at handling frequent updates and range-based searches, making them excellent for dynamic datasets.

3. Hash-Based Index: Hash-based indexing uses hash functions to associate keys with locations in a data structure. It is renowned for its quick access times but is susceptible to collisions.

The Algorithm and the Principle:

Hash Function: Creates fixed-size values from keys using the hash function.

Collision Handling: Handles situations where various keys provide the same hash value.

Storage and Retrieval: Locate values using their hash addresses.

A hash-based index, for instance, can be used to swiftly find information related to English terms.

Advantage:

Quick single-key retrieval access times.

Suits static datasets well.

Disadvantage:

Collisions may happen, which can slow down retrieval.

For range queries, ineffective.

Hash-based indexes are the best choice for lookups in settings with steady data since they are excellent at quick single-key retrieval.

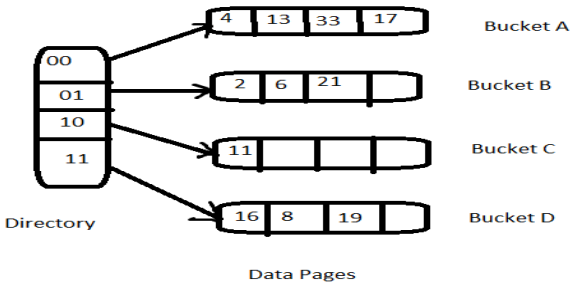

Fig 3: Hash based index.

4. Trie (Prefix Tree)

Strings may be stored and retrieved effectively using trie structures. They are especially helpful for dictionary searches and auto complete suggestions.

The Algorithm and the Principle:

Each node in the node structure represents a character, while paths stand in for strings.

Prefix Search: Navigates character-based nodes to find strings.

Compression: To conserve memory, data can be compressed.

Example: A trie can swiftly determine any word that has a particular prefix as its first letter.

Advantages:

Efficient for searches based on prefixes.

Suitable for dictionaries and auto complete applications.

Disadvantages:

Huge vocabulary-demanding on memory.

Non-prefix searches are slower.

Characteristics of performance:

Tries perform well in prefix-based searches but may use more RAM for large vocabulary sets.

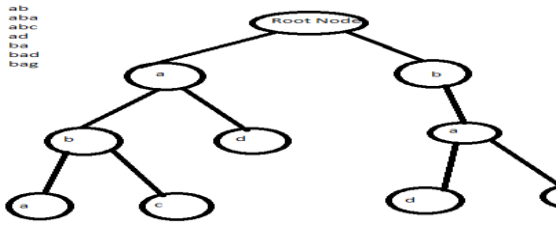

Fig 4: Trie Data Structure

English Word Search Algorithms

1. Exact Match Search: -

-Principle: Exact Match Search only returns results that are a perfect match to the search query.

- Algorithm: To locate precise matches, an exact match search often compares the search query to the terms that have been indexed directly.

- Effectiveness: When users are aware of the exact term they are looking for, an exact match search is quite effective. Although it guarantees accurate results, if the search term is misspelt or otherwise altered, it might not find the desired documents.

2. Fuzzy Search:

- Principle: Fuzzy search gets results that are comparable to the search query, considering spelling mistakes, typos, and other minor variations.

- Algorithm: The Levenshtein Distance (edit distance) algorithm, which counts the number of edit operations (insertions, deletions, and replacements) required to change one word into another, is a popular fuzzy search algorithm.

- Effectiveness: Fuzzy search works well when the search query has typos or other variants. Even when a user makes a small input error, it can still produce useful results.

3. Probabilistic Search: In contrast to exact matches or fuzzy similarities, probabilistic search finds results based on the likelihood that they are relevant.

- Technique: The Vector Space Model (VSM), which describes documents and queries as vectors in a multidimensional space, is one popular technique used in probabilistic search. To determine relevance, it computes the cosine similarity between the query and document vectors.

- Effectiveness: Probabilistic search works well for determining the relevance of documents. It considers a document's overall content and is capable of handling synonyms, different word orders, and a wider context. It might not be as useful, though, if customers are asking precise questions.

- Comparison and Analysis: Exact match search is simple and accurate, but it is not flexible enough to handle different user queries.

- Fuzzy search offers greater versatility and may accommodate minor typos or query changes. On the other hand, it can result in false positives and less relevant results.

- Probabilistic search is excellent at ranking documents according to relevance, considering both user queries and the total content of documents. Complex search circumstances can be handled with more success using it. However, it uses more processing power and might not work as effectively for requests that are brief or ambiguous.

The selection of a search algorithm is based on both user behaviour and the unique search requirements. Exact match search is suitable for applications where accuracy is important, and users are required to deliver precise queries. When dealing with user input problems or when users may have spelling issues, fuzzy search is helpful. When ranking results based on relevance is important, notably in information retrieval systems or search engines, probabilistic search is advantageous.

Many search engines combine these methods to offer a thorough search experience. For instance, a search engine may use fuzzy search to handle typos, exact match search for exact queries, and probabilistic search to rate results based on relevance. Users can obtain precise and pertinent search results using this combo, which also supports flexible search queries.

5. Experimental Results and Analysis

In this section the experimental results will be presented, and they will be compared. Comparison will take place in terms of time complexity and space complexity.

In all the searched word “external” is searched.

Table 2: Comparison of combination of standard indexing methods with 3-level indexing

Method used	Time taken for preprocessing	Time taken for search
Hashing with 3-level indexing	5195.16 millisecond	0.284194 millisecond
Inverted index with 3-level indexing	5205.214 millisecond	1.047372 millisecond
Trie with 3-level indexing	5673.471 millisecond	0.07295 millisecond
B-Tree with 3-level indexing	6076.89 millisecond	0.087976 millisecond
3-level indexing	4844.588 millisecond	0.162124 millisecond

Time complexity for Hashing:

Building hash table: O(N*M), where N is the number of documents and M is the average number of words per document.

Searching in hash table: O (1) average case (constant time), O(N) worst case.

Space complexity for hashing:

Hash Table storage: O(N*M), where N is the number of words and M is the average number of documents per word.

Time Complexity for Inverted Index:

Building Inverted Index: O(N*M), where N is the number of documents and M is the average number of words per document.

Searching in Inverted Index: O(K), where K is the length of the search key.

Space Complexity for Inverted Index:

Inverted Index storage: O(N*M), where N is the number of words and M is the average number of document IDs per word.

Time Complexity for Trie:

Building Trie: O(N*M), where N is the number of documents and M is the average number of words per document.

Searching in Trie: O(K), where K is the length of the search key.

Space Complexity for Trie:

Trie storage: O(N*M), where N is the number of nodes in the Trie and M is the average length of words.

Time Complexity for B-tree:

Building B-tree: O (NM log (NM)), where N is the number of documents, M is the average number of words per document, and log (NM) is the height of the B-tree.

Searching in B-tree: O(log(N*M)) average case (height of the B-tree).

Space Complexity for B-tree:

B-tree storage: O(N*M), where N is the number of words and M is the average number of document IDs per word.

Time Complexity:

Preprocessing: O(N*M), where N is the number of documents and M is the average number of words per document.

Searching in the list: O(M), where M is the number of words in the samelen list.

Space Complexity:

Storing the samelen list: O(K), where K is the number of words satisfying the conditions.

Based on the data presented and the time and space complexity analysis for each indexing method, we may make the following observations and analyses:

Preprocessing Time: Among the methods evaluated, "Hashing with 3-level indexing" had the shortest preprocessing time, requiring 5195.16 milliseconds.

The preparation timings for "Inverted index with 3-level indexing" and "Trie with 3-level indexing" are 5205.214 milliseconds and 5673.471 milliseconds, respectively.

"B-Tree with 3-level indexing" takes the longest to preprocess, at 6076.89 milliseconds.

The "3-level indexing" method has a preprocessing time of 4844.588 milliseconds.

Search Time: - "Trie with 3-level indexing" has the shortest search time, lasting only 0.07295 milliseconds.

"Hashing with 3-level indexing" searches in 0.284194 milliseconds.

The search times for "B-Tree with 3-level indexing" and "3-level indexing" are around 0.087976 milliseconds and 0.162124 milliseconds, respectively.

"Inverted index with 3-level indexing" takes the longest to search, with a duration of 1.047372 milliseconds.

Indexing strategy Comparison: Choosing an indexing strategy has a substantial impact on both preprocessing and search times.

Hashing and Trie-based indexing algorithms have faster search speeds than others, making them suited for speedy retrieval jobs.

While efficient in preparation, inverted indexing has a longer search time in this context.

B-Tree indexing has substantially longer preprocessing and search times, indicating that it may be less efficient for this specific dataset and search query.

Complexity Analysis: Hashing has an O (N*M) time complexity for generating the hash table and an O (1) time complexity for searching.

Inverted Index and Trie have O(N*M) time complexity for creating and O(K) time complexity for searching (where K is the length of the search key).

B-Tree has a greater temporal complexity for building and searching (O (NM log (NM)) and O(log(N*M)) average cases, respectively).

The "3-level indexing" without identifying the method most likely combines various indexing approaches to strike a balance between preprocessing and search times.

Space Complexity: All indexing methods have a space complexity of O(N*M) due to the storage required for the index structures, where N is the number of documents and M is the average amount of words per document.

Discussion and conclusion

The experimental results and the study of time and space complexities offer important new perspectives on how different indexing techniques function in relation to the dataset and search query at hand.

1.Preprocessing Time: Preprocessing times for various indexing techniques varied greatly. The quickest preprocessing technique is hashing with 3-level indexing, which takes 5195.16 milliseconds. This is a result of how effectively hash tables are made. In contrast, the preprocessing time for the B-Tree with 3-level indexing is the longest at 6076.89 milliseconds. The effectiveness of data preparation is directly impacted by the indexing strategy chosen.

2. Search Time: Trie with 3-level indexing has an impressively fast search time of 0.07295 milliseconds. Trie indexing is hence a fantastic option for quick data retrieval. The Inverted index with 3-level indexing, on the other hand, requires the most time to search, taking 1.047372 milliseconds. For applications that require real-time or nearly real-time results, the search time is a crucial consideration.

3. Comparison of Indexing Strategies: The indexing approach selected has a big impact on how long searches and preprocessing take. Speed-sensitive applications can benefit from the faster search times of hashing and trie-based indexing techniques. While efficient for preprocessing, inverted indexing falls short when it comes to search speed. The lengthy preparation and search durations of B-Tree indexing stand out, indicating that it might not be the ideal option for this dataset and query.

4. Time Complexity Analysis: The time complexity analysis highlights the performance gaps even further. Hashing allows constant-time search (O(1)), but creating the hash table takes O(N*M) time. Trie and Inverted Index both require O(N*M) for construction and O(K), where K is the length of the search key. The most resource-intensive technique is B-Tree, with construction times of O(NM log(NM)) and average case search times of O(log(N*M)). Despite not disclosing the precise mechanism employed, the "3-level indexing" strategy seems to establish a balance between preprocessing and search durations.

5. Space Complexity: Due to the storage needs for each index structure, all indexing algorithms have a space complexity of O(N*M). This shows that they all use roughly the same amount of RAM, which is directly related to the volume of papers and the average word count per document in the dataset.

6. Conclusion

The efficiency of information retrieval systems is, thus, significantly influenced by the method of indexing that is selected. The quickest search times are provided by hashing and trie indexing, which makes them appropriate for use in situations where quick data retrieval is crucial. But they have their own preprocessing time expenses. While inverted indexing is effective for planning, it might not be the greatest option when search speed is crucial. Although flexible, B-Tree indexing takes a lot of time for both preprocessing and searching, making it less appropriate in some situations. Preprocessing and search durations can be balanced out by using the "3-level indexing" method, which mixes different indexing strategies without revealing the precise strategy utilised. An indexing method may be selected depending on the individual requirements of an application that links with the trade-off between preprocessing efficiency and search performance. The requirements of the application, the size and makeup of the dataset, and the intended trade-offs between preparation time, search speed, and

space complexity should all be taken into consideration when choosing an indexing method.

References

[1] Ali Selamat, Nicholas Akosu (2014). Word-length algorithm for language identification of under-resourced languages, Journal of King Saud University - Computer and Information Sciences Volume 28, Issue 4, October 2016, Pages 457-469 http://dx.doi.org/10.1016/j.jksuci.2014.12.004.

[2] Julia Fomina, Denis Safikanov et.al (2020). Parametric and semantic analytical search indexes in hieroglyphic languages Procedia Computer Science, Volume 169, 2020, Pages 507-512 https://doi.org/10.1016/j.procs.2020.02.218.

[3] Kamran Kowsari, Kiana Jafari meimandi et.al (2019) Text Classification Algorithms: A Survey, doi:10.3390/info10040150, Information 2019, www.mdpi.com/journal/inform.

[4] Antonio Ferrandez, Jesus Peral (2019), MergedTrie: Efficient textual indexing PLoS ONE 14(4): e0215288. https://doi.org/10.1371/journal. pone.0215288.

[5] Vimal P Parmar, C K Kumbharana (2017), Implementation of Trie Structure for Storing and Searching of English Spelled Homophone Words, International Journal of Scientific and Research Publications, Volume 7, Issue 1, January 2017 ISSN 2250-3153.

[6] Rahat Yeasin Emon, Sharmistha Chanda Tista (2019), An Efficient Word Lookup System by using Improved Trie Algorithm P. N Singh, Tara G P (2021), Searching String in Bigdata- A better Approach by applied machine Learning, SN Computer Science (2021) 2:192.

[7] A Gani, A Siddiqa et.al, 2015, A survey on indexing techniques for big data: taxonomy and performance evaluation. DOI 10.1007/s10115-015-0830-y.

[8] Fatima Binta Adamu, Adid Habbal et.al (2015), A survey on big data indexing strategies, The 4th International Conference on Internet Applications, Protocols and Services (NETAPPS2015 DOI:10.13140/RG.2.1.1844.0721).

[9] Dalia A. Abd Al lattif, Murtadha M. Hamad, A comparative Study of Indexing Techniques effect in big data system storage Optimization, 2nd Al-Noor International Conference for Science and Technology (2NICST2020), Baghdad, Iraq, https://doi.org/10.1109/NICST50904.2020.9280309 .