String Matching Algorithms and Their Applications in Text Processing and Data Analysis

Noor Elshahidi & Bassel Shoeib

Abstract—**String matching algorithms are fundamental computational tools that have become increasingly important across diverse domains of computer science and technology. This paper presents a comprehensive survey of major string matching algorithms including Naive/Brute Force,Knuth Morris Pratt(KMP),Boyer Moore(BM)Rabin Karp and aho corasick analyzing their theoretical underpinnings, time and space complexities, and practical efficiency.We explore each algorithms distinctive features: KMPs prefix based approach that avoids redundant comparisons,Boyer Moore's powerful heuristics enabling sublinear performance, Rabin Karp's hash based methodology, and Aho Corasicks efficient multiple pattern matching capabilities.The paper examines implementation strategies and performance characteristics across different text patterns and alphabets, supported by experimental results demonstrating comparative efficiencies in real world scenarios. Additionally, we investigate the prtical applications of these algorithms in diverse fields including search engines bioinformatics cybersecurity and text processing.The survey also explores supporting data structures like tries suffix arrays and Bloom filters that enhance string matching capabilities. We conclude by identifying research gaps and future directions, including paral=lel computing implementations bit parallel optimizations, and adaptations for specialized data environments. This work serves as both a theoretical foundation and practical guide for researchers and practitioners working with text based data analysis.**

I. Introduction

String matching algorithms play an invaluable role in numerous text processing as well as data analysis operations in computer science. They are used in identifying patterns in large datasets ranging from straightforward text search through intricate bioinformatics studies.This survey discusses classic and novel approaches their key concepts applicability in practical scenarios and several applications in computing environments of today.

A. Importance and Relevance

String matching algorithms play a critical role in numerous fields of computing. For search engine systems the algorithms enable users to acces the desired information with ease and speed increasing the usefulness of the internet [2].For bioinformatcs the algorithms facilitate DNA sequence comparison, enabling scientists to identify genetic similarities as well as key mutations that aid in medical advancements [13].

Cybersecurity is greatly reliant on string matching for intrusion detection malware detection and network traffic analysis [10].Additionaly since digital text as well as large scale data is increasing rapidly efficient string matching is of utmost importance for text mining, information retrieval, as well as data analysis working with large quantities of unstructured data [1], [2].

B. Overview of the Paper

This essay describes string matching algorithms and their usage.To start with, some of the key algorithms are described with their concepts merits and drawback.The second section describes the algorithms in detail including pseudocode examples and complexity analysis of the Naive, KMP, Rabin Karp, Boyer Moore, and Aho Corasick algorithms.

We will first consider real applications in various fields, then conduct an experiment in matching strings with word frequency analysis.The article then considers other structures for helping with matching strings such as tries, suffix trees,and hash methods.Next we will identify areas for future improvement as well as challenges in the field, concluding with full references for additional reading.

II. Literature Review

String matching algorithms play an indispensable role in enhancing text processing.The algorithms have numerous applications in fields such as bioinformatics information retrival,plagiarism detection, and security.There are four major algorithms in this chapter: Knuth Morris Pratt (KMP), Boyer Moore (BM), Rabin Karp (RK) and hybrid approaches.We discuss the algorithms theory, implementation, as well as results of comparison.

A. Knuth Morris Pratt (KMP) Algorithm

The algorithm is efficient and will not waste time on redndant comparisons thanks to the use of the Longest Prefix Suffix (LPS) array.The algorithm is also efficient in producing results as the simple set of prefixes and suffixes rules make it easy for its correctness to be understood [15]. The algorithm is also useful in teaching since it ”builds on familiar program invariants in computer science,” simplifying how the theory is taught.

New visual instructions provide clearer instructions in [24].They present step by step photographs of how the LPS array is set up and constructed.The new explanations make index math simpler for those who have difficulty with it.

KMP is useful in the real world. For instance in 2023 a test utilized the algorithm in order to identify copied content in documents. This is an indication that the algorithm is capable of ”matching similar words and displaying them correctly,” an indication that it is suitable for serious tasks [22]. There is another test indicating how KMP is efficient in concurrent applications such as image processing, where it performs well in large sets of data [25].

Nevertheless, empirical studies comparing various algorithm demonstrate that for short and medium patterns, KMP is efficient.For longer patterns its performance may plateau in which case heuristic based algorithms are computationaly faster on average [18].

B. Boyer Moore (BM) Algorithm

The Boyer Moore algorithm is recognized as the quickest method of searching for text. It utilizes two major tricks the bad character rule, which skip over pieces of the text, and the good suffix rule to skip over portions of the text.This allows it to locate responses quicker in most situations [17]. The Boyer Moore algorithm scans the pattern from right to left, which enables these intelligent bypassing techniques.

In practice, BM performs better.BM performs better than KMP in all the tested cases, in particular for longer patterns whose superior performance is more discernible [18], [19]. For instance, Boyer Moore completed searches in 0.000085 seconds for patterns of 50 characters.

The Galil rule is an useful optimization that allows the BM algorithm not to check matching characters. It remains efficient even in the worst case scenario [19]. There are hybrids of BM that make it even faster for particular types of patterns.

Memory usage remains satisfactory: although BM employs additional tables for preprocessing, experiments indicate that the space requirement is comparable to KMP in various implementations [19].

C. Rabin-Karp Algorithm

The Rabin karp algorithm is an approach for string searching with the use of hashing. It converts the pattern and sub strings of the text into hash values,liek numbers and verifies the numbers for equality [20]. The novel concept here is the roling hash function, which allows for shifting the window with negligible additional work.

One of the most important aspects of RK's performance is selecting the hash function.The selection of appropriate prime numbers and bases reduces the likel ihood of colisions.When collisions do occur, a straight character by character comparison is employed in order to ensure they are identical.

RK is excellent at looking for multiple patterns simultaneously. It is capable of maintaining a number of hash values in a lookup table, making it suitable for fast fingerprinting. This makes it well-liked in fields such as network security and bioinformatics [23]. When compared in a study with word trigrams RK had 100% precision and recall, and it took 0.19 milliseconds to execute. It was however outpaced by the Horspool variation of Boyer Moore.

While RK is generally slower than KMP and BM for single pattern searches, its multi pattern capability offers unmatched scalability when dealing with large vocabularies or simultaneous lookups.

D. Empirical Comparisons and Hybrid Models

There have been numerous tests comparing these algorithms in laboratories.One of them showed that KMP and its derivatives, such as KMP Nextval, performed well on 5 character patterns, with an average time between 0.0007 seconds and 0.001 seconds [18].The BM algorithm began performing visibly better when the length of the pattern reached 50 or 500 characters.The use of both KMP and BM in hybrid approaches produced results comparable with the single best algorithms.

Memory usage remained largely consistent across implementations,indicating that the choice of algorithm should prioritize time efficiency and text/pattern characterstics rather than memory constraints.

E. Advanced Applications and Future Directions

They operate in disparate fields.The KMP is used when time is critical such as in school programs or plagiarism content finder programs [22]. The Rabin karp is used when multiple matches are being found simultaneously such as in malware detection. Boyer Moore is used in text editors as well as in search engines because of its proficiency in dealing with large patterns.

Improving hardware is critical these days.Paralel KMP is used with efficient pipelines for large searches as well as image processing as stated in the study [25]. The same modifications for BM as well as RK are examined to improve performance on multiple computers as well as on distributed systems.

New work indicates that hybrid and adaptive template matching models perhaps with machine learning, can assist in selecting the appropriate algorithm depending on input features,pattern length, and redundancy.

F. Critical Discussion of Literature

1) Strengths and Weaknesses: There are pros and cons of every algorithm that make them both suitable for various situations. The Naive algorithm is not fast but is inexpensive and easy to use.This is suitable for short patterns or when speed doesn't matter most but ease of use does [2].

KMP performs optimally when it can operate in linear time regardless of the input. Preprocessing makes it efficient [3], [9]. In typical text processing applications,however,Boyer Moore may actually be faster [20].

Rabin karp's strength lies in multiple pattern searching though it risks false positives due to hash collisions, necessitating character by character verification when hash values match [5]. This additional verification can degrade performance in worst case scenarios.

Boyer-Moore generally performs well in real life scenarios especially on larger alphabets and patterns.Many prefer using it for various text procesing operations [2], [20] for this reason. The only major disadvantage of Boyer Moore is that the preprocessing process is complex compared to other simpler algorits.

Aho-Corasick is extremely effective for matching multiple patterns but requires additional time for preparation and more memory compared to single pattern matching algorithms [4].This makes it the ideal option for applications such as network security where the cost of preparation is amortized over numerous searches.

2) Real-World Applications: String matching algorithms are applicable in numerous fields.Close matching and exact matching algorithms assist in the searches of search engines.Boyer Moore algorithms as well as their variants, are usually employed since they are suitable for use in real life scenarios [2].

In bioinformatics, they asist in comparing DNA and protein sequences.There are special variants that are utilized to address particular issues in processing genomics data [13]. Applications in bio informatics frequently require accuracy through the utilization of approximate string matching to manage mutation and sequencing errors.

String matching is used heavily in network security for detection of intrusions as well as detection of malware. Multiple pattern matching in Aho Corasick comes in handy for examining network traffic for numerous known attack signatures [10].

Text processing programs like plagiarism detection employ algorithms like Rabin karp in order to efficiently identify similar document portions.String matching is employed in data compression algorithms for identifying repeating patterns for encoding efficiently [2], [5].

G. Recent Advances

There is recent advancement in string matching concentrating on novel problems as well as novel types of computing.Paralel method which utilize multicore and distributed systems in order to process large datasets rapidly are gaining increasing popularity [11], [12].

Bit-parallel techniques provide superior performance since they are able to process several bit operations simultaneously.This accelerates the process of matching [17]. This approach is efficient for short patterns and gave rise to Shift OR and BNDM (Backward Non Deterministic DAWG Matching) algorithms [17].

Quantum computing techniques are an emerging approach to investigating the string matching. Recent research indicates that they could be significantly faster with Grovers search algorithm and bit parallel methods [12].While the work is presently primarily theoretical it presents interesting opportunities for very rapid future string matching.

III. Algorithm Description and Analysis

A. Detailed Algorithmic Breakdown

1) Naive Algorithm: Naive matching of strings is the simplest method of searching for patterns.It matches the pattern with all the possible sub strings of the text in detail [2].

The algorithm is straightforward but inefficient.It is $O(nm)$ in the worst case scenario [2]. There can be as many as m character checks necessary for each comparison, and $n-m+1$ starting points from which to consider.

2) KMP Algorithm: The Knuth Morris Pratt algorithm is superior to the Naive approach because the algorithm makes use of prior comparisons in order not to have to redo work [3].The algorithm contains two major components.

Preparation time is $O(m)$ and search time is $O(n)$ for a total of $O(n+m)$ time [3]. This is considerably quicker than the Naive approach particularly for long patterns.

3) Rabin karp Algorithm: The Rabin karp algorithm utilizes a hashing-based technique to search for strings [5].

The rolling hash function is crucial for speed.It allows us to efficiently compute the hash of the next substring as follows: t = (d * (t - text.charAt(i) * h) + text.charAt(i + m)) where d is the number of letters in the alphabet, q is a prime number, and h is $d^{(m-1)}$
$modq$
cite5.

The time is roughly $O(n+m)$ on average but in the worst scenrio, it is $O(nm)$ when collisions are numerous [5]. The algorithm performs well with multiple patterns,so it can be useful in detecting plagiarism.

4) Boyer Moore Algorithm: The Boyer moore algorithm adopts two useful techniques for eliminating portions of the text while searching [2], [20]:

The algorithm scans the characters of the pattern from right to left as it shifts the pattern from left to right over the text.The technique assists us in avoiding large segments of the text, thus alowing us to be faster in practice [20].

Preprocessing will cost $O(m+\sigma)$ time, where σ represents the count of distinct letters. The search process will cost $O(n/m)$ time in the best scenario and $O(nm)$ time in the worst scenario [2], [20].

5) Aho Corasick Algorithm: Aho-Corasick Algorithm The Aho Corasick algorithm enables efficient matching of multiple patterns simultaneously [4].

The preparation time is $O(m)$ with m being the overall length of all patterns.The search time is $O(n+z)$ with z being the number of times that the patterns appear [4].The Aho-Corasick algorithm is wonderful for situations with matching numrous patterns such as intrusion detection systems.

B. Performance Comparison

A comprehensive comparison of string matching algorithms reveals distinct performance characteristics across different scenarios:

TABLE I
PERFORMANCE COMPARISON OF STRING MATCHING ALGORITHMS

Algorithm	Preprocessing	Best Case	Average Case	Worst Case	Space	Multiple Patterns
Naive	$O(1)$	$O(n)$	$O(nm)$	$O(nm)$	$O(m)$	Poor
KMP	$O(m)$	$O(n)$	$O(n+m)$	$O(n+m)$	$O(m)$	Poor
Rabin-Karp	$O(m)$	$O(n+m)$	$O(n+m)$	$O(nm)$	$O(1)$	Good
Boyer-moore	$O(m+\sigma)$	$O(n/m)$	$O(n)$	$O(nm)$	$O(m+\sigma)$	Poor
Aho-Corasick	$O(m)$	$O(n+z)$	$O(n+z)$	$O(n+z)$	$O(m)$	Excellent

The Naive algorithm, while simple becomes impractical for large texts due to its quadratic worst-case complexity.KMP offers consistent linear time performance regardless of input characteristics, making it reliable across diverse scenarios.

Rabin Karp's performance varis greatly on the quality of the hash function and the pattern.It can be made to match many paterns efficiently but can be slowed down by hash collisions.BoyerMoore tends to perform better in practice for matching a single pattern, particularly when the alphabet is large, so it is used in text processing applications.

Aho Corasick is an efficient algorithm for matching multiple patterns. It uses linear time, so it is beneficial in applications such as network defense and content filtering. However, as the overall size of the patterns is increased it requires increased memory which can make it difficult in environments with limited memory.

IV. Applications of String Matching Algorithms

A. Search Engines

String matching algorithms play an important role in search engines.The algorithms assist the search engines in efficiently processing searches and returning matching results [2].On input of the search query, the system must search for documents containing the correct terms from potentially billions of indexed web documents.

Boyer moore and its variants are primarily utilized due to their excellence in locating exact matches.Algorithms for approximate matching exist that can handle misspellings and various forms of user queries [2].The efficiency and accuracy of these algorithms impact the perception of users regarding their experience as even minor variations in response time can significantly impact user satisfaction.

Beyond basic matching string algorithms support advanced search features like phrase searches, wildcard queries and proximity searches.They also contribute to search ranking algorithms, where pattern location and frequency help detemine document relevance to a query [2].

B. Data Processing

1) Text Processing: String matching has numerous applications in text processing.It may range from simple search and replace operations in word processors to complicated content analysis software [2]. Approximate string matching is utilized by spell checking software to determine potential corrections for misspelled words. Plagiarism detection software employs algorithms such as Rabin Karp to efficiently identify similar passages of text in various documents [2], [5].

Data validation applications utilize string matching to verify input formats, such as identification numbers or email addresses in order to maintain accurate data in information systems. Natural language processing applications employ these techniques in applications such as splitting text into components, identifying names and sentiment analysis in order to enable machines to interpret and process human language [2].

2) Data Mining: String matching within data mining assists in identfying patterns in messy large datasets [1]. Text mining algorithms assist in extracting useful information from document sets, depicting trends relationships and insights that in manual searches will be difficult to identify.

Advanced methods of string matching are used for handling large quantities of text in big data analysis systems that provide useful insights for business.The eficiency of such methods impacts how well data mining systems function and develop, determining how they deal with larger quantities of data [1], [2].

C. Bioinformatics

1) DNA Sequence Matching: String matching algorithms revolutionized genomic studies through rapid comparison of DNA strings [13]. The human genome comprises approximately 3 billion base pairs, and thus efficient algorithms are necessary for effective analysis.

There are particular algorithms such as KMP and Boyer moore that detect genetic markers, mutations as well as similarities between species.Such algorithms tend to look for near matches in order to take into consideration mutations, insertons as well as deletions occurring within genetic material [13].

Recent advances in genomic sequencing have genrated an enormous volume of genetic data, and it is even more critical now to have highly efficient string matching algorithms in bioinformatics workflows.String matching algorithms are crucial for fundamental tasks such as genome assembly, variant detection and gene expression analysis, which underpin medical research and personalized medicine [13].

2) Protein Structure Comparison: String matching programs assist in the analysis of protein sequences so that scientists can know how proteins function and what proteins are composed of [13]. Scientists can detect relationships between proteins make predictions about their structure, and establish novel treatment techniques.

They employ particular techniques that mimic the way in which proteins evolve over time. The details gathered from such stdies assist in identifying novel medicines, comprehend diseases, as well as learn fundamental biology.

D. Cybersecurity

String matching algorithms play a significant role in network security. They are primarily applied in intrusion detection systems that discover malware [10]. Signature based intrusion detection employs speedy algorithms such as Aho Corasick.The algorithm screens network traffic against numerous recognized attack patterns simultaneously which aids in the identification of potential threats in real time [10].

How they operate determines how useful they can be since they have to process a lot of traffic rapidly accuratly.Los Alamos National Laboratory in their studies determined that more efficient methods of matching strings can make intrusion detection five times as effective as simple techniques [10].

Antimalware as well as antivirus programs employ string matching searches for known patterns of malicious code within files as well as in memory to defend systems from infection. It must be both efficient and precise because false negatives can cause attacks to proceed, while false positives can disrupt real actions [10].

V. Experiment: Comparative Performance of String Matching Algorithms on Real-World Text

A. Objective

The test examines the performance of four matching algorithms for strings.The algorithms used are Naive, Knuth Morris Pratt (KMP), Rabin Karp and Boyer Moore.The test is performed with real texts from Wikipedia articles.The aim is to compare their performance in terms of time, accuracy, and efficiency in practical usage.

B. Corpus Description

The corpus consists of major portions of various Wikipedia articles combined in single plain text files. The articles on various subjects were selected in such a way as to resemble actual text data.

- **Corpus Size:** Approximately 500000 characters
- **File Type:** Plain text (.txt)]
- **Separator:** Articles divided using "END_OF_ARTICLE" marker
- **Preprocessing:** Minimal cleaning; copied directly from Wikipedia

C. Patterns Used

Four search scenarios were selected in order to examine algorithms for various scenarios:

Pattern	Type	Length
`algorithm`	Single word	9
`data structure`	Phrase	15
`efficiency`	Single word	10
`computational`	Single word	13

TABLE II
PATTERNS SELECTED FOR TESTING

D. Algorithms Implemented

The following algorithms were implemented in C++:

- **Naive Search:** Simple brute force comparison.
  - *Best case Time Complexity:* $\mathcal{O}(n)$, when the pattern is found immediately at the beginning of the text.
  - *Worst case Time Complexity:* $\mathcal{O}(nm)$, when the pattern does not match at all, or matches only at the last position.
  - *Average case Time Complexity:* $\mathcal{O}(nm)$, generally assuming random text and pattern distributions.
- **KMP (Knuth Morris Pratt):** Utilizes the Longest Prefix Suffix (LPS) array for pattern preprocessing.
  - *Best case Time Complexity:* $\mathcal{O}(n)$, when the pattern matches early in the text and no backtracking occurs.
  - *Worst case Time Complexity:* $\mathcal{O}(n+m)$ where n is the length of the text and m is the length of the pattern (since the LPS array allows skipping unnecessary comparisons).
  - *Average case Time Complexity:* $\mathcal{O}(n+m)$ since KMP preprocesses the pattern in $O(m)$ and scans the text in $O(n)$ without revisiting positions unnecessarily.
- **Rabin Karp:** Hashing based rolling window.
  - *Best case Time Complexity:* $\mathcal{O}(n+m)$ when there are no hash collisions, and all the hashes match immediately.
  - *Worst case Time Complexity:* $\mathcal{O}(nm)$ when there are many hash collisions, requiring character by character comparison after each hash match.
  - *Average case Time Complexity:* $\mathcal{O}(n+m)$, assuming a good hash function with minimal collisions.
- **Boyer moore:** Uses huristics (like the bad character rule and good suffix rule) to skip comparisons.
  - *Best case Time Complexity:* $\mathcal{O}(n/m)$, when the pattern matches early in the text, allowing many character comparisons to be skipped.
  - *Worst case Time Complexity:* $\mathcal{O}(nm)$ when there are many mismatches and minimal skipping of characters occurs.
  - *Average case Time Complexity:* $\mathcal{O}(n/m)$ or better, depending on the effectiveness of the heuristics in skipping sections of the text.

E. Experimental Setup

- **Machine:** Intel Core i7, 16GB RAM
- **OS:** Windows 11
- **Compiler:** `Clion`
- **Timing Tool:** `std::chrono`
- **Repetitions:** Each test run 5 times, average recorded

F. Results

Pattern	Algorithm	Matches	Time (ms)
`algorithm`	Naive	38	44.1
	KMP	38	5.7
	Rabin-Karp	38	12.4
	Boyer-Moore	38	4.8
`data structure`	Naive	15	51.2
	KMP	15	6.5
	Rabin-Karp	15	14.9
	Boyer-moore	15	4.4

TABLE III
COMPARISON OF MATCH COUNT AND EXECUTION TIME

G. Analysis

- **Correctness:** All algorithms produced identical match counts.
- **Efficiency:** Boyer moore performed best, especially with long patterns. KMP followed closely. Naive was significantly slower
- **Suitability:**Boyer moore is idel for large corpora and longer patterns. KMP is efficient for repetitive searches.Rabin Karp offers flexibility but is hash dependent

H. Insights

- **KMP** is excellent for repeated pattern search across different documents
- **Boyer Moore** is suited for applications requiring minimal search time

- **Rabin Karp** is good for multi pattern scenario
- **Naive** is suitable only for teaching or simple one off cases.

VI. Future Directions and Challenges

A. Optimization and Scalability

Data is increasing in size rapidly so it remains an enormous challenge for matching algorithms to catch up [1], [2]. There is a need for future work to make existing algorithms faster and develop new ones that can efficiently work with lots of text

Potential optimization directions include:

- Cache-conscious algorithm designs that maximize memory hierarchy efficiency
- Compressed string matching that operates directly on compressed data without full decompression
- Specialized algorithms for emerging storage technologies like non volatile memory

Hybrid techniques that select the ideal algorithms depending on the input features are highly encouraging [2]. It is shown through research that no algorithm is optimal for all scenarios, so intelligent algorithm selection can significantly improve system performance

B. Parallel Computing

Large datasets make finding matching strings much simpler with parallel and distributed computing techniques [11], [12]. With multi core processing performance is greatly enhanced such as when dealing with the processing of genomics and other large texts [11].

Studies have considered the use of GPUs in the matching of strings with positive results. There are programs that can execute faster for particular functions. These techniques make use of graphics processors in order to examine numerous potential matches simultaneously [11], [12].

Cloud based distributed string matching represents another frontier, enabling processing of text collections too large for single machine approaches.These systems must address challenges including load balancing, fault tolerance, and communication overhead to achieve practical scalability [11]

C. Quantum Computing

Quantum computing can improve string matching significantly [12].Studies in recent times have demonstrated that bit parallel methods of string matching can be applied in quantum systems, which could potentially produce faster results through Grovers search algorithm [12].

The article “Bridging Classical and Quantum String Matching: A Computational Reformulation of Bit Parallelism” demonstrates an evident method of converting classic algorithms for use in quantum solutions with definite advantages in computing [12]. While such concepts are currently generaly theoretical they provide implications that suggest an ability in the future for incredible string matching in particular fields

Using real life applications of quantum string matching remains difficult due to hardware constraints as well as the infancy of quantum computation. However, with advancements in hardware as well as algorithms, we can expect in the future the usage of quantum string matching in critical applications [12].

D. Adaptation for Noisy Data

Real-world text data increasingly presents challenges beyond simple exact matching including:

- Multilingual content with diverse character encodings and linguistic patterns
- OCR processed text containing recognition errors and artifacts
- Social media content with intentional misspellings abbreviations and slang
- Domain specific terminology and jargon requiring specialized handling

Approximate string matching algorithms address some of these issues but more research is needed to develop techniques that are both efficient and able to deal with all types of changes in real world text [13]. Some promising areas to research are:

- Edit distance optimizations that provide more efficient fuzzy matching
- Learning based approaches that adapt to specific types of text variations
- Context aware matching that utilizes surrounding text to resolve ambiguities
- Phonetic matching for applications where pronunciation similarity matters more than exact spelling

VII. Conclusion

String matching algorithms form a significant and expanding subfield of computer science with numerous applications in everyday life.From the straightforward but time consuming Naive approach to the more sophisticated methods such as Aho Corasick and bit parallel algorithms, it is apparent how the subject has expanded and yet continued to address novel issues in data processing

The various algorithms considered in this document have their trade offs regarding preparation requirements average performance worst case outcomes and how well they accommodate different patterns.The knowledge of these trade offs allows users to select appropriate algorithms for their requirements be they concerned with worst case performance efficient average performance, or accommodating multiple patterns

Next generation string matching algorithms will improve in terms of processing large quantities of data as well as applying multiple modes of computing such as quantum computing. They will also accommodate messy and heterogeneous real world text.These advancements will enable us to continue processing and analyzing the enormous body of digital text that underpins our information economy.

References

[1] S. I. Hakak, A. Kamsin, P. Shivakumara, G. A. Gilkar, W. Z. Khan, and M. Imran, "Exact string matching algorithms: Survey, issues, and future research directions," *IEEE Access*, vol. 7, pp. 69614–69628, 2019, doi: 10.1109/ACCESS.2019.2914071.

[2] M. Kumar, "String matching algorithms: KMP and Rabin Karp explained," upGrad, Mar. 24, 2025. [Online]. Available: https://www.upgrad.com/blog/string-matching-algorithms/

[3] "Concept and strategy of preprocessing in KMP algorithm," Virtual Labs, 2025. [Online]. Available: https://ds2-iiith.vlabs.ac.in/exp/kmp-algorithm/preprocessing-of-kmp-algorithm/concept-and-strategy-preprocessing.html

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.

[5] "Rabin Karp," Essential Algorithms, 2025. [Online]. Available: https://www.programming-books.io/essential/algorithms/rabin-karp-554fa18e5d254f45bc6e46ef50461b1c

[6] M. Ghodsi, "Suffix trees properties recap," Univ. Maryland, 2008. [Online]. Available: https://www.cbcb.umd.edu/confcour/Spring2008/CMSC858P-materials/notes-2-28.pdf

[7] N. K. Singh, "Suffix array- LCP and finding unique substring," Topcoder, Nov. 14, 2021. [Online]. Available: https://www.topcoder.com/thrive/articles/suffix-array-lcp-and-finding-unique-substring

[8] S. Müller, "Algorithm::BloomFilter," MetaCPAN, 2015. [Online]. Available: https://metacpan.org/pod/Algorithm::BloomFilter

[9] A. A. Nababan and M. Jannah, "Algoritma string matching brute force dan Knuth-Morris-Pratt," Semantic Scholar, Dec. 11, 2019. [Online]. Available: https://www.semanticscholar.org/paper/ALGORITMA-STRING-MATCHING-BRUTE-FORCE-DAN

[10] M. E. Fisk and G. Varghese, "Applying fast string matching to intrusion detection," OSTI.GOV, 2001. [Online]. Available: https://www.osti.gov/biblio/975767

[11] C. S. Rao, K. B. Raju, and S. V. Raju, "Parallel string matching with multi-core processors," *Global J. Comput. Sci. Technol.*, vol. 13, no. 1, pp. 1–9, 2013.

[12] S. Faro, A. Pavone, and C. Viola, "Bridging classical and quantum string matching," arXiv, 2025. [Online]. Available: https://arxiv.org/abs/2503.05596

[13] S. S. Hasan, F. Ahmed, and R. S. Khan, "Approximate string matching algorithms," *Int. J. Comput. Appl.*, vol. 118, no. 21, pp. 1–7, 2015.

[14] S. Gupta and A. Rasool, "Bit parallel string matching algorithms," *Int. J. Comput. Appl.*, vol. 89, no. 15, pp. 1–6, 2014.

[15] T. Ásványi, "Invariants in KMP algorithm," *Acta Cybernetica*, vol. 27, no. 1, pp. 5–14, 2025.

[16] G. F. Ahmed and N. Khare, "Hardware based string matching algorithms," *Int. J. Comput. Appl.*, vol. 88, no. 12, pp. 38–44, 2014.

[17] Wikipedia contributors, "Boyer-Moore string-search algorithm," Wikipedia, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_string-search_algorithm

[18] S. S. Dawood and S. A. Barakat, "Empirical performance evaluation of KMP and BM," *J. Duhok Univ.*, vol. 23, no. 1, pp. 1–10, 2020.

[19] T. Saleh and F. C. Ergin, "Performance comparison of KMP and BM," in *Proc. ICAISC*, 2025, pp. 1–6.

[20] I. Sadiah and M. Ishlah, "Implementation of Rabin-Karp algorithm," *SISFORMA*, vol. 10, no. 1, pp. 45–52, 2023.

[21] Z. Zhang, "Review on string-matching algorithm," Univ. Electron. Sci. Technol. China, 2022. [Online]. Available: https://www.shs-conferences.org/articles/shsconf/pdf/2022/14/shsconf_stehf2022_03018.pdf

[22] A. Alfat and M. Faisal, "KMP for plagiarism detection," *Int. J. Comput. Appl.*, vol. 175, no. 7, pp. 25–30, 2023.

[23] A. Fadlil, S. Sunardi, and R. Ramdhani, "Similarity identification using string matching," *INTENSIF*, vol. 6, no. 2, pp. 253–270, 2022.

[24] S. Sculthorpe, "Knuth-Morris-Pratt illustrated," *J. Funct. Program.*, vol. 34, p. e3, 2024.

[25] S. Aygun, E. O. Gunes, and L. Kouhalvandi, "Parallel KMP in Python," in *Proc. AIEEE*, 2016, pp. 1–5.