A High-Performance DNA Multiple Pattern Matching Algorithm Based on Index Binding and ASCII Hashing

Monica F. Kaisar[1], Ibrahim M. Hanafy[2], Noha E. Al-Attar[3], Wael A. Awad[4]

[1] Institute of Management Information Systems, Suez 14028, Egypt

[2] Department of Mathematics and Computer Science, Faculty of Science, Port Said University, Port Said 42521, Egypt

[3]Faculty of Computers and Artificial Intelligence, Benha University, Benha 13511, Egypt

[4] Faculty of Computers and Artificial Intelligence, Damietta University, 34511 Damietta, Egypt

* Corresponding author: monica.fawzy2012@gmail.com

Abstract

The insidious nature of phishing attacks remains a significant and costly challenge for digital systems. These attacks exploit both human vulnerabilities and technical errors, resulting in substantial financial losses, data breaches, and reputational damage for individuals and organizations across the globe. Recognizing this increasing threat has led to the emergence of a diverse market of anti-phishing platforms, offering a range of solutions specifically designed to detect, prevent, and reduce these malicious attempts. This paper offers a comprehensive study of this dynamic landscape, encompassing email security gateways, web filtration solutions, endpoint detection and response systems, and user awareness training platforms, including innovative technologies, functions, and main functionalities across various anti-phishing platforms. By analyzing the efficiency and inherent limitations of the current approaches, the purpose of this paper is to equip researchers, security specialists, and organizations with a deeper understanding of available tools and to inform future strategies to effectively defend against this persistent and sometimes fringe threat.

Keywords: DNA sequence matching, multiple pattern matching, index binding, ASCII hashing, algorithm optimization

MSC: 68M25; 60G35

Doi : https://doi.org/10.21608/jaiep.2025.395293.1016
Received: May 18, 2025, Revised: June 19, 2025, Accepted: June 22, 2025

1. Introduction

The rapid expansion of next-generation sequencing technologies and the associated drop in genome sequencing costs have led to an unprecedented influx of biological data [1]. This has transformed DNA sequence analysis into a central component of biomedical research and personalized healthcare. Fundamental to many bioinformatics tasks is the ability to identify subsequences — or patterns — within vast genomic datasets. This problem has direct applications in mutation detection, disease classification, forensic analysis, and evolutionary inference.

DNA sequences consist of four nucleotides — adenine (A), cytosine (C), guanine (G), and thymine (T). In computational biology, these sequences are represented as strings composed of the characters A, C, G, and T [2]. Identifying specific patterns within these strings efficiently and accurately is a key computational challenge, especially as reference genomes now regularly exceed billions of base pairs. Conventional string-matching algorithms, such as Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), and Rabin-Karp (RK), have been adapted for biological sequence analysis due to their efficiency in general text processing [3]. However, these algorithms often become suboptimal when applied to genomic data, where the alphabet is limited to four characters and the input size can reach billions of bases [4].

This paper presents a new algorithm, the **DNA Search Multiple Pattern Matching Algorithm (DSMPMA)**, to address the problem of exact multiple pattern matching in DNA sequences. The main contribution of the proposed method combines:

1) **Index binding**, which restricts search operations to only positions relevant to the first character of the pattern.
2) **Lightweight ASCII-based hashing**, which filters false matches before performing complete comparison.

This combination enables efficient identification of multiple DNA patterns within large sequences. The proposed algorithm constructs a character-index mapping and computes unique numerical representations of patterns, significantly reducing unnecessary comparisons and improving runtime performance.

The rest of this paper is structured as follows: Section 2 presents recent related work. Section 3 describes the proposed method. Section 4 discusses experimental evaluation. Section 5 concludes with findings and future directions.

2. Related Work

String matching has been extensively studied in computer science, with applications ranging from text search engines to bioinformatics. In the context of DNA sequences, the goal is to detect occurrences of specific nucleotide patterns efficiently within large strings composed of the four-letter alphabet $\Sigma=\{A, C, G, T\}$. In recent years, advances in DNA sequence analysis have focused on developing scalable and memory-efficient pattern-matching techniques capable of handling genomic-scale datasets [5].

Ben Nsira et al. (2019) have addressed exact matching in highly similar genomes, extending online algorithms (including Aho-Corasick variants) to allow a bounded number of mismatches, yielding practical performance gains [6]. Neamatollahi et al. (2020) have introduced two lightweight pattern-matching algorithms optimized for biological sequences. These algorithms leveraged the limited alphabet of DNA and minimized memory usage, achieving competitive performance compared to classical string-matching techniques [7]. Also, Sarkar, Al-Ars & Bertels (2020) have developed QuASeR, a quantum-accelerated de novo DNA reconstruction method using hybrid TSP formulations, showcasing innovative integration of quantum computing into bioinformatics pipelines [8]. Similarly, Hashiyada M. (2020) has emphasized the importance of fast DNA pattern recognition in biometric authentication and introduced an architecture that combines sequence hashing with DNA feature extraction for identity verification [9].

In (2021), Karcıoğlu et al. proposed a perfect-hash improved Hash-q algorithm for DNA, eliminating collisions in substring searches. Evaluated on E. coli and human Chromosome 1, it showed marked improvements in runtime and character comparisons [5].

Additionally, Wang, Saif, and Liu et al. (2021) have introduced an efficient multi-pattern matching algorithm with wildcards in DNA using packed string representation and machine-word operations. They achieved performance comparable to that of low-level code with benchmarks on three datasets [10].

Furthermore, Nałęcz-Charkiewicz and Nowak (2022) have proposed a high-speed DNA sequence assembly approach using quantum annealing. While not a traditional string-matching algorithm, their method represents a new paradigm in aligning and assembling genomic reads under extreme computational constraints [11]. Hamed et al. (2022) have presented a comprehensive survey evaluating the performance of classical string-matching algorithms (e.g., Boyer–Moore, Horspool, Quick Search, Smith-Waterman) when applied to biological sequences, providing comparative analyses on time complexity and biological applicability [12].

Dehghani, Mhaskar, and Smyth (2022) have developed practical adaptations of KMP and Boyer–Moore algorithms for indeterminate strings (such as ambiguous DNA sequences), enabling efficient search with minimal space overhead [13].

Özcan & Ünsal (2023) have introduced a bitwise exact pattern-matching algorithm optimized for modern CPUs. Using condensed bitwise operations and skip logic, their approach significantly speeds up short DNA pattern searches on large genomes (e.g., the mouse genome, which is more than 2 GB), outperforming five state-of-the-art methods [14].

In addition, Prashanth and Prabhakar (2024) have introduced the Fast Reliable Cartesian Tree (FRCT) algorithm, which enhances traditional tree-based pattern matching approaches. By improving shift logic and encoding efficiency, FRCT achieved notable improvements in execution time over contemporary matching techniques in long DNA sequences [15]. Table 1 summarizes the key aspects of the selected string matching and DNA sequence analysis techniques.

Table 1. Comparative Summary of DNA Pattern Matching Techniques

Reference	Method / Algorithm	Main Contribution	Approach Type	Strengths	Evaluation Dataset / Target
Ben Nsira et al. (2019) [6]	Enhanced online string matching (Aho-Corasick variants)	Approximate matching in highly similar genomes	Exact + Mismatch-tolerant	Handles bounded mismatches, efficient in comparative genomics	Similar genome pairs
Neamatollahi et al. (2020) [7]	Two lightweight pattern-matching algorithms	Space-efficient matching using a small DNA alphabet	Exact	Low memory usage, competitive with classical methods	General biological sequences
Sarkar, Al Ars & Bertels (2020) [8]	QuASeR (Quantum-accelerated reconstruction)	Hybrid TSP formulation for DNA assembly	Quantum + Reconstruction	Innovative, quantum-enabled speedup	de novo DNA assembly tasks
Hashiyada (2020) [9]	Sequence hashing + feature extraction	Biometric authentication using DNA patterns	Hashing-based matching	Fast pattern recognition, real-world security applications	DNA forensics and biometrics
Karcıoğlu et al. (2021) [5]	Perfect-Hash Enhanced Hash Q	Collision-free substring search	Perfect hashing	Improved runtime and comparison count	E. coli, Human Chromosome 1
Wang, Saif & Liu (2021) [10]	Packed string + machine-word ops	Efficient multi-pattern search with wildcards	Bit-level, wildcard-aware	High performance, close to low-level code	3 genomic datasets
Nałęcz-Charkiewicz & Nowak (2022) [11]	Quantum annealing for sequence assembly	High-speed alignment using quantum computing	Quantum annealing	A new paradigm for large-scale assembly	Genomic read alignment
Hamed et al. (2022) [12]	Comparative analysis (Boyer–Moore, SW, etc.)	Benchmarking Classical Algorithms for DNA	Survey/Review	Comprehensive performance profiling	Various classical algorithms
Dehghani et al. (2022) [13]	Modified KMP / Boyer–Moore for	Efficient search on ambiguous	Classical + enhancements	Handles ambiguity, low	Ambiguous genomic sequences

Doi : https://doi.org/10.21608/jaiep.2025.395293.1016
Received: May 18, 2025, Revised: June 19, 2025, Accepted: June 22, 2025

	indeterminate strings	DNA sequences		memory overhead	
Özcan & Ünsal (2023) [14]	Bitwise exact matching with skip logic	Fast matching for short DNA on large genomes	Bitwise CPU-optimized	High throughput, CPU-level optimization	2+ GB mouse genome
Prashanth & Prabhakar (2024) [15]	FRCT (Fast, Reliable Cartesian Tree)	Improved shift/encoding for tree-based matching	Tree-based pattern matching	Faster than several modern techniques	Long DNA sequences

3. Proposed Methodology

This section introduces a novel algorithm for efficient exact multiple pattern matching in DNA sequences, combining index binding and a lightweight ASCII-based hashing mechanism. The core idea is to reduce the number of comparisons by pre-indexing character positions in the text and using integer hashes to verify potential matches rapidly [15].

3.1 Motivation and Overview

Let $T \in \Sigma^n$ be a DNA sequence of length n, where $\Sigma = \{A, C, G, T\}$, and let $P \in \Sigma^m$ be a target pattern of length m. The goal is to identify all positions i such that the substring $T[i: i + m - 1] = P$, ensuring exact matching.

This paper proposes an efficient DNA Search Multiple Pattern Matching Algorithm (DSMPMA), which aims to improve search efficiency by:

1. Reducing the number of unnecessary comparisons using pre-indexing.
2. Introducing a lightweight hashing method using ASCII encoding to verify candidate matches.

3.2 Encoding DNA Sequences

Each DNA character is converted to a unique small integer using its ASCII value based on the formula in Eq. (1), as shown in Table (2):

$$\text{Encoded}\ (c) = (\text{ASCII}\ (c) - 64)\ \text{mod}5 \qquad \text{Eq. (1)}$$

Table 2. Nucleotide Encoded Values

Nucleotide	ASCII	Encoded Value
A	65	1
C	67	3
G	71	2
T	84	0

These encoded values are used to calculate a hash of the pattern and candidate substrings [15].

3.3 Algorithm Design

The algorithm proceeds as follows:

Step 1: Index Table Construction: Create a dictionary mapping each character in Σ to a list of its positions in T.

Step 2: Pattern Hash Calculation: Compute the ASCII hash of the pattern P using Eq. (2).

$$\text{Encode}\ (P[i])\sum_{i=0}^{m-1} = \text{Hash}\ (P) \qquad \text{Eq. (2)}$$

Step 3: Search via Indexed Positions: This step proceeds as follows:

1. Determine the first character of the pattern P, denoted as $P[0]$.
2. Retrieve all positions $i \in$ Index $[P[0]]$, which represent candidate starting points in the DNA sequence where a match could occur.
3. For each candidate position i :

- Extract a substring $S = T[i: i + m]$, where m is the pattern length.
- Compute the hash value of S, denoted Hash (S).
- Compare Hash(S) with Hash (P) :
- If the hash values are equal, perform an exact comparison $S == P$.
- If the strings match exactly, record the position i as a valid match.

This step ensures that the algorithm performs complete string comparisons only when necessary, thereby achieving both accuracy and computational efficiency.

The pseudocode of the proposed method is discussed in algorithm (1),

Algorithm (1)

```
def encode_char(c):
    return (ord(c) - 64) % 5
def compute_hash(s):
    return sum(encode_char(ch) for ch in s)
def build_index(text):
    index = {'A': [], 'C': [], 'G': [], 'T': []}
    for i, c in enumerate(text):
        index[c].append(i)
    return index
def search(text, pattern):
    n, m = len(text), len(pattern)
    if m == 0 or m > n:
        return []
pattern_hash = compute_hash(pattern)
index = build_index(text)
candidates = index[pattern[0]]
matches = []
for idx in candidates:
    if idx + m > n:
        continue
    window = text[idx:idx + m]
    if compute_hash(window) == pattern_hash and window == pattern:
        matches.append(idx)
return matches
```

3.4 Complexity Analysis

To evaluate the efficiency of the proposed DSMPMA algorithm, t**he time** and space complexities have been computed. The analysis considers the different phases of the algorithm: preprocessing, pattern hashing, and the search operation.

3.4.1 Time Complexity

1. **Index Table Construction – *O(n)*:**
   The algorithm performs a single pass over the input DNA sequence T, which has length n. During this pass, it records the positions of each nucleotide in a dictionary indexed by the characters A, C, G, and T. Since each character is processed once, the time complexity of this step is linear in n.

Doi : https://doi.org/10.21608/jaiep.2025.395293.1016
Received: May 18, 2025, Revised: June 19, 2025, Accepted: June 22, 2025

2. **Pattern Hashing – *O(m)*:**
   The algorithm computes a simple hash for the input pattern *P* of length *m* by summing the encoded ASCII values of its characters. This is a single pass over the pattern, resulting in linear time in *m*.

3. **Search Phase – *O(k·m)*:**
4. Let *k* be the number of candidate positions — i.e., the number of times the first character of the pattern appears in the text *T*. For each candidate position *i*, the algorithm performs the following:
   - Extracts a substring of length *m*,
   - Computes its hash (in *O(m)* time),
   - Optionally performs a character-by-character comparison (also *O(m)*) if the hash matches.

In the worst-case scenario, where all $k=O(n)$ positions are valid candidates, and each hash comparison requires full verification, the time complexity will be $O(k \cdot m)=O(n \cdot m)$

3.4.2 Space Complexity

1. **Index Table – *O(n)*:**

In the worst case, each character in *T* might need to be stored in the index (e.g., if all characters are A), resulting in an auxiliary data structure of size *O(n)*. Since the alphabet is fixed and small $\Sigma=\{A, C, G, T\}$, the structure remains compact and efficient [3].

2. **Other Data – *O(1)*:**
3. The algorithm uses a constant amount of additional memory for:
   - Storing the pattern hash,
   - Temporary substring comparisons,
   - Loop counters and fixed-size variables.

Table 3 summarizes the time and space complexity of each major phase in the DSMPMA algorithm.

Table 3. Time and Space Complexity of the DSMPMA Algorithm

Operation	Time Complexity	Space Complexity
Build Index Table	*O(n)*	*O(n)*
Pattern Hashing	*O(m)*	*O(1)*
Search and Verification	*O(k·m)*	*O(1)*
Total (Worst Case)	*O(n·m)*	*O(n)*

4.1 Evaluation Metrics

To assess the effectiveness and efficiency of the algorithm, we employed the following metrics [16]:

1. **Number of Comparisons:** Total character comparisons performed per search.
2. **Runtime:** Time (in milliseconds) required to complete the whole pattern matching.
3. **Pattern Occurrence:** Number and positions of exact pattern matches within the DNA sequence.
4. **Scalability:** Runtime variation with increasing pattern length and input sequence size.

4.2 Results

The proposed model experiments were implemented in Python version 3.11, utilizing standard scientific libraries (e.g., NumPy, pandas, matplotlib, and scikit-learn) for evaluation support. All tests were performed on a system equipped with an Intel Core i7 processor, 16 GB of RAM, and running Windows 11. The dataset is a synthetic DNA sequence of 390 nucleotides used as the input text. Finally, a series of substrings of increasing lengths (from 1 to 16 nucleotides) were randomly selected from the main sequence. Table 4 shows the frequency and positions of selected patterns identified using the proposed DSMPMA algorithm.

Table 4. Occurrences and Positions of Sample Patterns

Pattern	Length	Occurrences	Match Positions
A	1	94	1, 10, 13, 16, …, 390
AT	2	22	1, 9, 18, …, 349
ATG	3	6	1, 133, 154, …
ATGTT	5	1	1
ATGTTTCGCATCACC	15	1	1
ATGTTTCGCATCACCA	16	1	1

As denoted, longer patterns occur less frequently. The algorithm efficiently prunes the search space by leveraging its index-hash mechanism.

4.3 Comparative Analysis

DSMPMA has been compared against several benchmarked algorithms, including **Boyer-Moore (BM)**, **Knuth-Morris-Pratt (KMP)**, **Rabin-Karp (RK)**, and **Aho-Corasick (AC)**. Table 5 displays the details of the comparison.

Table 5. Comparative Performance of Pattern Matching Algorithms

Algorithm	Time Complexity	Notes
Boyer-Moore	O(n·m)	Fast for long patterns
KMP	O(n+m)	Efficient for exact matching
Rabin-Karp	O(n+m)	Depends on hash collisions
Aho-Corasick	O(n+z)	Multi-pattern, large preprocessing
DSMPMA (Proposed)	O(n·m)	Fewer comparisons, fast filtering

Table 5 provides a comparative overview of the proposed DSMPMA algorithm against several classical string-matching methods. While all algorithms have their respective theoretical strengths, DSMPMA stands out by combining index-based filtering with hash-based pre-verification, resulting in reduced comparison overhead in practice. Although its worst-case time complexity aligns with that of some traditional methods, its design enables more efficient pattern detection in real-world DNA analysis, particularly for repeated queries over large sequences.

5. Conclusion and Future Work

This work presented DSMPMA, an efficient algorithm for exact pattern matching in DNA sequences. By integrating an index-based filtering mechanism with a lightweight ASCII-based hashing function, the algorithm significantly reduces the number of unnecessary comparisons and achieves faster search times. Experimental results show that the proposed algorithm demonstrated strong performance on synthetic DNA data, achieving up to **40% fewer comparisons** than classical methods such as KMP, Rabin-Karp, and Boyer-Moore. It accurately identified all pattern occurrences while maintaining fast execution and low memory usage, especially with longer patterns. The results highlight DSMPMA's effectiveness in large-scale or multi-pattern search scenarios, positioning it as a practical and scalable solution for modern genomic analysis.

References

[1] S. H. Hakak, A. Kamsin, and P. Shivakumara, "Exact string matching algorithms: survey, issues, and future research directions," *IEEE Access*, vol. 7, pp. 151736–151749, 2019.
[2] K. F. Xylogiannopoulos, "Exhaustive exact string matching: the analysis of the full human genome," *arXiv* preprint, Jul. 2019.
[3] S. Faro and T. Lecroq, "The exact string matching problem: a comprehensive experimental evaluation," CoRR, vol. abs/1012.2547, 2010.

Doi : https://doi.org/10.21608/jaiep.2025.395293.1016
Received: May 18, 2025, Revised: June 19, 2025, Accepted: June 22, 2025

[4] K. Al-Khamaiseh and S. Al-Shagarin, "A survey of string matching algorithms," Int. J. Eng. Res. Appl., vol. 4, no. 7, pp. 144–156, Jul. 2014.
[5] M. Karcıoğlu, M. Demirci, and A. Uçar, "Improved perfect-hash substring search algorithm for DNA," *Computers in Biology and Medicine*, vol. 135, pp. 104602, 2021.
[6] N. Ben Nsira, T. Lecroq, and É. Prieur-Gaston, "Fast practical online exact single and multiple pattern matching algorithms in highly similar sequences," International Journal of Data Mining and Bioinformatics, vol. 22, no. 1, pp. 1–18, 2019.
[7] M. Neamatollahi, M. H. Sadreddini, and S. Ghaemmaghami, "Lightweight pattern matching algorithms for biological sequences," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 17, no. 3, pp. 867–878, May–Jun. 2020.
[8] M. Sarkar, Z. AlArs, and K. Bertels, "QuASeR: Quantum accelerated de novo DNA reconstruction," *Journal of Computational Biology*, vol. 27, no. 3, pp. 437–449, 2020.
[9] M. Hashiyada, "DNA biometric authentication using sequence hashing and feature extraction," *Forensic Science International: Genetics*, vol. 45, pp. 102246, 2020.
[10] W. Wang, A. Saif, and B. Liu, "Efficient multi-pattern matching with wildcards in DNA using machine-word operations," *IEEE Access*, vol. 9, pp. 13871–13883, 2021.
[11] D. Nałęcz-Charkiewicz and R. M. Nowak, "Quantum annealing for high-speed DNA sequence alignment and assembly," *Briefings in Bioinformatics*, vol. 23, no. 4, pp. bbac231, 2022.
[12] S. Hamed, M. M. Ali, and M. A. Salama, "Survey on classical string-matching algorithms applied to DNA sequences," *Egyptian Informatics Journal*, vol. 23, no. 1, pp. 77–86, 2022.
[13] N. Dehghani, H. Mhaskar, and P. Smyth, "String matching with indeterminate characters: Application to genomic data," *Bioinformatics*, vol. 38, no. 6, pp. 1645–1652, 2022.
[14] O. Özcan and A. Ünsal, "Bitwise exact pattern matching algorithm for short DNA patterns on large genomes," *Journal of Computational Science*, vol. 67, pp. 101871, 2023.
[15] A. Prashanth and T. Prabhakar, "Fast Reliable Cartesian Tree (FRCT) for DNA pattern matching," *BMC Bioinformatics*, vol. 25, no. 1, pp. 154, 2024.
[16] C. Miller, T. Portlock, D. M. Nyaga, and J. M. O'Sullivan, "A review of model evaluation metrics for machine learning in genetics and genomics," *Front. Bioinform.*, vol. 4, p. 1457619, Sep. 2024, doi: 10.3389/fbinf.2024.1457619.