https://doi.org/10.56093/JISAS.V79I3.8
Available online at http://isas.org.in/isa/jisas
JOURNAL OF THE INDIAN SOCIETY OF AGRICULTURAL STATISTICS 79(3) 2025 279–294

Sequence Alignment and Mapping Algorithms used in Big Data Bioinformatics

Prakash Kumar, Raju Kumar, Deepak Singh, Md. Yeasin and Himadri Shekhar Roy

ICAR-Indian Agricultural Statistics Research Institute, New Delhi

Received 16 December 2025; Revised January 2026; Accepted 06 February 2026

SUMMARY

The rapid evolution of next-generation sequencing (NGS) technologies has transformed genomics into a data-intensive science, generating unprecedented volumes of sequencing data and creating major computational challenges for efficient read mapping and alignment. This review focuses on recent advances in mapping and alignment algorithmic strategies that have emerged in response to increasing sequencing throughput, longer read lengths, and the growing demand for rapid and accurate genomic analyses. Special emphasis is placed on lightweight and sketch-based approaches, including MinHash and HyperLogLog, which enable efficient similarity estimation and large-scale dataset comparison, particularly in metagenomic applications. Key indexing and compression techniques such as the Burrows–Wheeler Transform (BWT), FM-Index, and Locality-Sensitive Hashing (LSH) are critically discussed for their roles in reducing memory requirements and accelerating alignment while maintaining high sensitivity and tolerance to mismatches. Alignment tools are compared based on essential performance metrics, including mapping sensitivity, proportion of appropriately paired reads, computational time, memory usage, and robustness in handling tandem repeat-rich reads. This review synthesizes recent methodological innovations aimed at addressing big-data challenges in bioinformatics and provides insights to support the development of next-generation hybrid aligners that integrate emerging sketching and indexing paradigms for scalable and accurate genomic analysis.

Keywords: BWT; FM-Index; HyperLogLog; MinHash and LSH.

1. INTRODUCTION

The advent of next-generation sequencing (NGS) technologies has dramatically increased the number of bases generated per sequencing run while substantially reducing sequencing costs, leading to an unprecedented accumulation of genomic data. Efficient handling and analysis of these massive datasets are essential for exploring the biological properties of organisms (Nowrousian, 2010). Read mapping and sequence alignment represent fundamental steps in NGS data analysis pipelines and are widely applied in transcriptomics and genomics studies (Lee *et al.,* 2013), variant detection, genotype and single nucleotide polymorphism (SNP) calling, and transcript abundance estimation, which forms the basis for most downstream analyses. Tandem repeats are prevalent in genomic sequences and pose significant challenges to accurate read mapping due to alignment ambiguity, whereby reads may align to multiple genomic locations (Treangen and Salzberg, 2012; Torresen *et al.,* 2019).

Most read-mapping tools are developed using either heuristic-based or index-based alignment strategies. Index-based aligners generally offer high accuracy but are computationally intensive, requiring substantial memory and processing time. In contrast, heuristic-based aligners are faster, consume less memory, and are more suitable for mapping shorter reads (Lindner and Friedel, 2012). Since the 1980s, the continuous growth of sequencing data has created new opportunities for biological discovery, but it has also introduced significant computational challenges. Genome sequencing generates large-scale datasets that demand big-data analytical frameworks and high-performance computing solutions (Stephens *et al.,*

Corresponding author: Himadri Shekhar Roy
E-mail address: himadriiasri@gmail.com

2015). The processing and storage of such datasets increasingly rely on distributed and cloud computing infrastructures, which have facilitated the development of novel bioinformatics software tools (Muir *et al.,* 2016; Langmead and Nellore, 2018).

Classical sequence alignment approaches rely on similarity and evolutionary distance optimization. Simple distance measures may be used to estimate similarity; however, weighted scoring schemes that assign different penalties to mismatches and gaps provide more biologically meaningful alignments (Durbin *et al.,* 1998). Optimal global alignment is achieved using the Needleman-Wunsch dynamic programming algorithm (Needleman and Wunsch, 1970), whereas optimal local alignment is performed using the Smith-Waterman algorithm (Smith and Waterman, 1981). These methods were further refined by Gotoh, who introduced affine gap penalties, significantly improving alignment accuracy (Gotoh, 1982). In the 1990s, the development of the Basic Local Alignment Search Tool (BLAST) by Altschul revolutionized bioinformatics by introducing heuristic filtration and indexing strategies to accelerate local sequence similarity searches (Altschul *et al.,* 1990). Despite its widespread adoption over the past three decades, BLAST is not well-suited for mapping the massive volumes of short reads produced by NGS platforms.

NGS technologies generate extremely large numbers of short sequencing reads at low cost (Magi *et al.,* 2010). Mapping these reads to reference genomes comprising billions of nucleotides is computationally demanding and requires substantial computing resources. To overcome the limitations of BLAST in handling NGS-scale data, numerous specialized read-mapping tools have been developed over the past two decades (Brinda, 2016). Prominent examples include BWA and Bowtie2, which employ the BWT-based FM-index and BWA-MEM algorithms for efficient read mapping (Langmead and Salzberg, 2012; Li, 2013). Indexing structures based on the Burrows–Wheeler Transform (BWT) require significantly less memory and time compared to suffix arrays or trees and can be extended for bidirectional pattern searches (Burrows and Wheeler, 1994; Ferragina and Manzini, 2000; Lam *et al.,* 2009; Makinnen *et al.,* 2015; Kucherov *et al.,* 2016; Canzar and Salzberg, 2017).

In metagenomic analyses, where millions of reads must be mapped against thousands of reference genomes, traditional alignment-based methods become increasingly inefficient. Composition-based comparison approaches using fixed-length k-mers offer a faster alternative by estimating sequence similarity based on shared k-mer content, achieving accuracy comparable to BLAST while substantially reducing computational time (Wood and Salzberg, 2014). The use of spaced k-mers further improves accuracy by capturing non-contiguous sequence patterns (Morgenstern *et al.,* 2015). With the emergence of terabyte-scale datasets, traditional indexing strategies are often impractical. To address this challenge, locality-sensitive hashing (LSH) algorithms have been introduced to enable approximate similarity searches through compact sequence sketches (Indyk and Motwani, 1998; Wang *et al.,* 2011).

MinHash, a prominent instance of LSH, is an alignment-free technique that converts sequences into sets of k-mers (shingles) and estimates similarity using the Jaccard index (Broder, 1997). It generates compact sketches by selecting minimum hash values from k-mer sets, enabling rapid comparison of large datasets. Although MinHash performs well for datasets of similar sizes, its effectiveness diminishes for datasets of unequal lengths. This limitation can be mitigated using winnowing and minimizer-based techniques that generate window-based sketches for improved similarity estimation (Schleimer *et al.,* 2003; Robert *et al.,* 2004). Today, thousands of genome sequencing projects worldwide continue to generate massive volumes of data, necessitating increasingly scalable and efficient mapping strategies.

The evolution of alignment methodologies has closely paralleled advancements in sequencing technologies: dynamic programming-based alignments dominated the 1990s, BLAST-like heuristic algorithms emerged during the early 2000s alongside Sanger sequencing, specialized NGS aligners developed during the 2010s, and contemporary metagenomic analyses increasingly rely on alignment-free approaches (Kucherov, 2019). Sequence alignment is broadly classified into pairwise and multiple sequence alignment (MSA). Pairwise alignment employs dot-matrix analyses, dynamic programming algorithms, and heuristic k-tuple methods, each serving distinct analytical purposes in modern bioinformatics pipelines.

1.1 Pairwise sequence alignment

Sequence alignment is used to identify similarities between two protein or nucleic acid sequences and to align them in a manner that maximizes identity and similarity, thereby enabling inference of structural, functional, and evolutionary (common ancestry) relationships. Based on evolutionary origin, sequence relationships are categorized as homologs (genes inherited from a single common ancestor), orthologs (genes diverged by speciation that typically retain the same function in different organisms, e.g., haemoglobin genes across species), paralogs (genes derived from duplication events within a genome that may acquire related but distinct functions, e.g., haemoglobin and myoglobin), xenologs (genes that share similarity due to horizontal gene transfer), and analogs (genes that have evolved independently to perform similar functions through convergent evolution, e.g., subtilisin and chymotrypsin). Three major criteria are used to evaluate sequence alignments: conserved regions, which retain physicochemical properties across evolutionary time; similarity, which reflects the degree of correspondence between sequences based on identity and conservative substitutions; and identity, which denotes exact residue matches between aligned sequences. Pairwise alignment is widely applied to identify homologous sequences, detect duplicated regions and shared domains, and compare genes with their corresponding gene products. The fundamental properties and interpretation of sequence alignments are illustrated in Fig. 1.

1.2.1 Dot matrix analysis algorithm

Dot-matrix analysis is a graphical method used to compare two sequences and identify regions of potential similarity (Gibbs and McIntyre, 1970). In this approach, one sequence is placed along the horizontal axis of a matrix, while the other sequence is arranged along the vertical axis.A dot or symbol is plotted at the intersection of matching residues, whereas mismatches are left blank. As the comparison proceeds across the matrix, continuous diagonal lines of dots indicate regions of similarity and potential alignments.

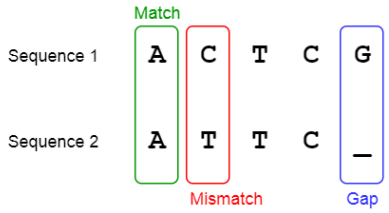

Fig. 1. Pairwise alignment properties.

The figure illustrates the fundamental properties of pairwise sequence alignment, highlighting matched regions (identical or similar residues), mismatched regions, and gap regions introduced to optimize alignment and represent insertions or deletions (indels) between two sequences.

A schematic representation of the dot-matrix method and its working principle is shown in **Fig. 2**. Three characteristic patterns can be observed in dot-plot analyses, as illustrated in **Fig. 3.** (i) partial homology, where only specific segments of divergent sequences show similarity; (ii) long insertions and deletions, which appear as breaks or shifts in diagonal patterns; and (iii) tandem repeats, which produce multiple parallel diagonals.

Pros: Dot-plot analysis provides a comprehensive visual representation of all potential alignments between two sequences, facilitating the identification of insertions, deletions, inverted repeats, and repeat regions. The method is applicable to both nucleotide and protein sequences.

Cons: Dot plots do not provide quantitative measures of alignment quality, lack statistical significance testing, and are limited in their ability to distinguish biologically meaningful alignments from spurious matches.

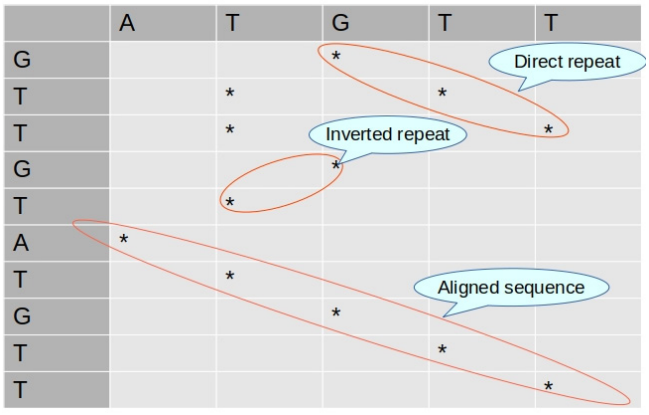

Fig. 2. Dot-matrix analysis algorithm

The figure illustrates three characteristic alignment patterns identified using dot-matrix analysis: (i) aligned (collinear) sequence regions represented by continuous diagonals, (ii) inverted repeats forming anti-diagonal patterns, and (iii) direct repeats appearing as parallel diagonal lines.

1.2.2 Local alignment techniques

Local sequence alignment is used to identify conserved domains and short regions of similarity between partially homologous sequences by focusing on subsequences that yield the highest similarity scores. Unlike global alignment, local alignment detects regions of high similarity within larger, divergent sequences. The Smith–Waterman algorithm is the most widely used method for local alignment and identifies the optimal local alignment by selecting the highest scoring path anywhere within the dynamic programming score matrix.

The Smith–Waterman algorithm proceeds through the following steps:

1. Initialize a scoring matrix for the two sequences.
2. Fill the matrix by computing the optimal score for each cell from position (0,0) to (p,q) using defined match, mismatch, and gap penalties.
3. Store traceback pointers to enable reconstruction of the optimal alignment.
4. Identify the maximum score in the matrix and trace back from this position to obtain the locally optimal aligned subsequences. Let sequences $A=a_1, a_2, \ldots, a_p$ and $B=b_1, b_2, \ldots, b_q$ denote two input sequences of lengths p and q, respectively. The initialization of the scoring matrix is illustrated in **Fig. 4**, and the procedure for computing the optimal local alignment scores is shown in **Fig. 5**.

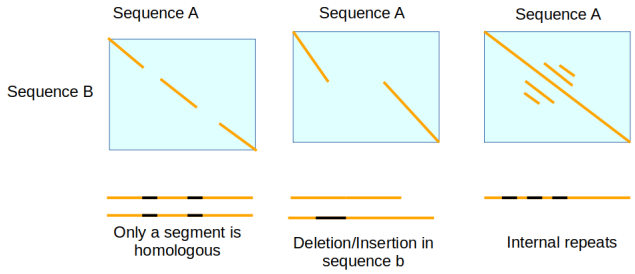

Fig. 3. Diagram illustrating various alignment patterns

The figure depicts three characteristic situations observed in dot-plot analyses: (i) partial homology in divergent sequences, where only specific segments align; (ii) long insertions and deletions indicated by disruptions in diagonal patterns; and (iii) tandem repeats represented by multiple parallel diagonals.

At each step of the alignment process, the pairing score can be calculated in three possible ways:

(i) both sequences contribute one residue, resulting in either a match or a mismatch;
(ii) sequence A contributes a residue while sequence B introduces a gap; and
(iii) sequence B contributes a residue while sequence A introduces a gap. The optimal score for each cell (x,y) in the scoring matrix is computed as:

BestScore [x,y] = max{BestScore [x−1,y−1] + Match (x,y), BestScore [x−1,y] − GapPenalty, BestScore [x,y−1] − GapPenalty, 0}

	-	C	G	T	G	A	A	T	T	C	A	T
-	0	0	0	0	0	0	0	0	0	0	0	0
G	0											
A	0											
C	0											
T	0											
T	0											
A	0											
C	0											

Fig. 4. Initialization of the scoring matrix

The figure illustrates the initialization procedure for constructing the scoring matrix, showing the calculation steps used to assign initial values prior to dynamic programming-based local alignment.

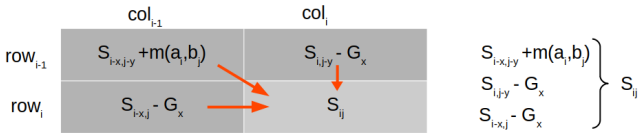

Fig. 5. Calculation of the optimal local alignment score

The figure illustrates the dynamic programming procedure used to compute the optimal (maximum) score in the scoring matrix for local sequence alignment.

A numerical example illustrating the calculation of the best score is provided in Fig. 6.

Pros: The Smith–Waterman algorithm identifies optimal local alignments with high accuracy and has a very low probability of missing biologically relevant similarity. It allows users to define substitution matrices

and gap penalty schemes, providing flexibility to model different evolutionary scenarios.

Cons: The method is limited to pairwise local alignment, is computationally expensive in terms of time and memory, and becomes impractical for large-scale datasets, making it less suitable for high-throughput NGS applications.

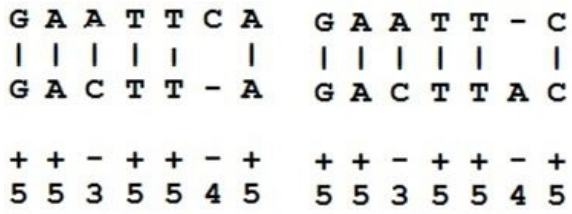

Fig. 6. Numerical example illustrating the calculation of the best score

The figure presents a numerical example demonstrating the computation of the optimal local alignment score using a scoring scheme of match = 5, mismatch = −3, and gap penalty = −4.

1.2.3 Global alignment techniques

Global alignment is used to align two sequences over their entire lengths and is most appropriate when the sequences are of similar length and are expected to share overall similarity. In this approach, alignment is performed from the beginning to the end of both sequences to obtain the best possible end-to-end correspondence. The Needleman-Wunsch (NW) algorithm is the classical method for global alignment and identifies the optimal alignment by tracing back from the highest scoring cell in the final row or column of the scoring matrix. The procedure begins with the construction of a scoring matrix, where the first row and column are initialized using cumulative gap penalties. The matrix is then filled by calculating optimal scores for each cell based on match, mismatch, and gap penalties. The dynamic programming process yields the final scoring matrix, from which the optimal global alignment is obtained through traceback. The complete NW pairwise alignment process is illustrated in **Fig. 7**, and the resulting aligned sequences are shown in **Fig. 8**.

Alignment properties are illustrated in Fig. 9 with emphasis on gap handling.

Pros: The Needleman–Wunsch algorithm is well suited for aligning two sequences of comparable length that share high overall similarity. It performs end-to-end (global) alignment, ensuring that the entire length of both sequences is considered. The method allows users to define substitution matrices and gap penalty schemes, providing flexibility to model different evolutionary relationships.

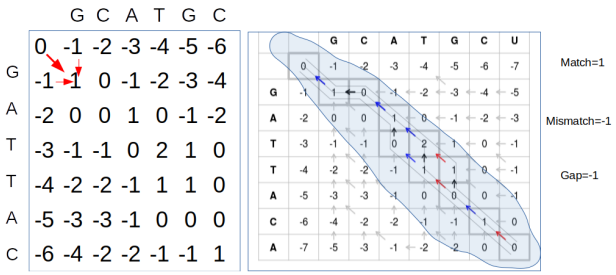

Fig. 7. Scoring matrix construction and traceback in Needleman–Wunsch alignment

The figure illustrates the calculation steps involved in constructing the scoring matrix and performing traceback to obtain the optimal global pairwise sequence alignment using the Needleman–Wunsch algorithm.

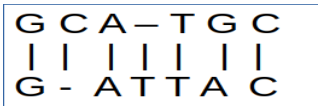

Fig. 8. Pairwise sequence alignment result.

The figure shows the final global alignment of the given pair of sequences obtained using the Needleman–Wunsch algorithm, with a calculated optimal alignment score of 4.

Cons: The algorithm is computationally expensive in both time and memory, making it impractical for large-scale datasets. Its dynamic programming framework involves a large number of computational steps, leading to slow performance when aligning long sequences. Additionally, the method requires appropriate gap penalty selection to achieve efficient and biologically meaningful alignments.

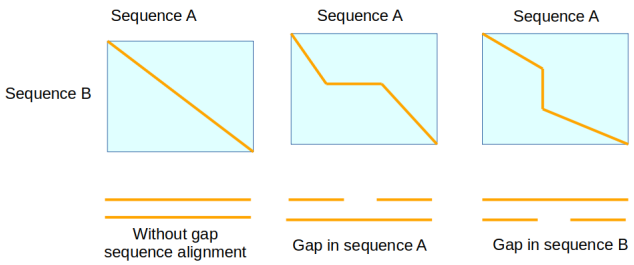

Fig. 9. Representation of gaps in pairwise alignment

The figure depicts both ungapped and gapped alignments between different sequences, highlighting the introduction of gap regions to represent insertion and deletion (indel) events and to optimize alignment quality.

1.2 Progressive Alignment Methods for Multiple Sequence Alignment (MSA)

Progressive alignment is a heuristic strategy widely used for multiple sequence alignment (MSA) (Feng and Doolittle, 1987). This approach proceeds in two main stages. First, evolutionary relationships among sequences are estimated by constructing a guide tree based on pairwise alignment scores. Pairwise alignments are initially computed using dynamic programming algorithms, and evolutionary distances are calculated for all $(N(N-1))/2$ sequence pairs, where N is the total number of sequences. These distances are used to generate a distance matrix, which is subsequently subjected to clustering to produce a guide tree. In the second stage, the guide tree directs the progressive construction of the MSA by iteratively aligning sequences or groups of sequences, starting with the most closely related pairs and gradually incorporating additional sequences. Each new sequence is aligned to an existing alignment based on the highest alignment score, and this process continues until all sequences are incorporated into the final alignment. Well-known implementations of this method include ClustalW and its graphical user interface version, ClustalX. A schematic representation of the progressive alignment strategy is shown in **Fig. 10**.

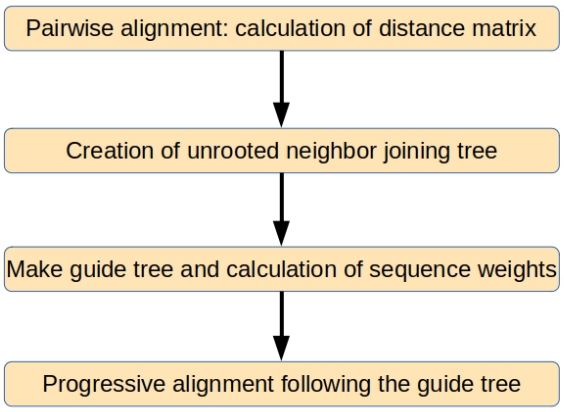

Fig. 10. Systematic diagram of the progressive alignment algorithm

The figure illustrates the sequential steps involved in progressive multiple sequence alignment, including pairwise distance calculation, guide tree construction, and iterative alignment of sequences or sequence groups.

Pros: Progressive alignment algorithms are effective for aligning large numbers of sequences and are computationally efficient for MSA tasks. They offer flexibility in selecting substitution matrices and gap penalty schemes at different stages of alignment based on evolutionary distances inferred from the guide tree. For example, fewer gaps can be permitted in conserved domains, while more gaps can be allowed in variable or loop regions of protein sequences.

Cons: Progressive alignment does not guarantee a globally optimal solution, particularly when datasets contain many partial or highly divergent sequences. Errors introduced during early alignment steps can propagate through subsequent stages, leading to suboptimal final alignments. The accuracy of the alignment may degrade as evolutionary divergence among sequences increases; however, refinement procedures can be applied to partially correct initial alignment errors.

1.3 Iterative Methods

Iterative alignment techniques employ a refinement-based approach in which previously generated pairwise or multiple alignments are repeatedly realigned as new sequences are incorporated into a multiple sequence alignment (MSA). Unlike progressive alignment, iterative methods allow correction of earlier alignment errors during successive refinement cycles, thereby improving overall alignment accuracy. The primary objective of this approach is to minimize alignment inconsistencies by repeatedly optimizing the alignment as additional sequences are added. Pairwise distances among all sequences are first computed and subjected to single-linkage clustering to construct a guide tree. Sequences within the same cluster are aligned initially, followed by alignments between clusters. This iterative realignment process continues until convergence is achieved. The workflow of iterative alignment is illustrated in **Fig. 11**.

Iterative alignment methods aim to optimize a general objective function by maximizing overall alignment quality through repeated refinement of the alignment. In this approach, subsets of sequences (sub-MSAs) derived from the original dataset are

realigned iteratively to improve the global alignment score. Pairwise distances are recalculated at each iteration, allowing continuous refinement of the guide tree and alignment, which leads to improved accuracy compared with progressive alignment methods. A prominent example of this strategy is the Multiple Sequence Comparison by Log-Expectation (MUSCLE) algorithm, which updates distance measures and realigns sequences across successive iterations to enhance alignment quality. Other examples of iterative alignment methods include DIALIGN (Ait *et al.,* 2013) and MUSCLE (Edgar, 2004).

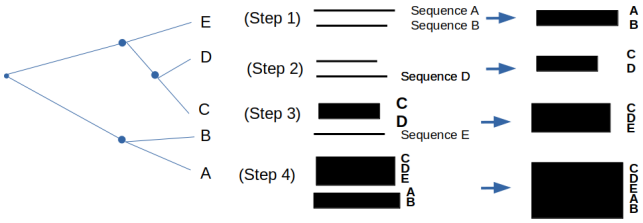

Fig. 11. Iterative alignment methods

The figure illustrates the major steps involved in iterative multiple sequence alignment, including distance calculation, clustering, alignment refinement, and repeated realignment to improve overall alignment quality.

1.4 Burrows-Wheeler transforms (BWT)

The Burrows–Wheeler Transform (BWT) is a reversible data transformation technique that rearranges an input string into a form that is highly amenable to compression and is therefore also referred to as a block-sorting compression method. BWT does not require the storage of additional information to allow exact reconstruction of the original sequence. To address memory constraints associated with long sequencing reads, simplified or naïve implementations of the BWT can be employed. A key characteristic of the BWT is its ability to transform the input sequence into a string containing long runs of repeated characters, which can be efficiently compressed using standard compression algorithms.

The transformation is performed by generating all cyclic rotations of the input string, sorting these rotations lexicographically, and then extracting the last character of each sorted rotation to form the transformed string. Because identical characters tend to cluster together after sorting, the resulting string is highly compressible. For example, given the input string “banana,” a unique sentinel character “$” is appended to obtain “$banana” prior to transformation. The detailed working steps of the BWT procedure are illustrated in **Table 1**.

Table 1. BWT steps

First step: Now create the cyclic arrangement of string would be -->	**Second step:** Sort this cyclic arrangement of strings alphabetically would be -->	**Third step:** Finally we can get BWT output as the last column
$banana	$banana	$ | banan | a
a$banan	a$banan	a | $bana | n
na$bana	ana$ban	a | na$ba | n
ana$ban	anana$b	a | nana$ | b
nana$ba	banana$	b | anana | $
anana$b	na$bana	n | a$ban | a
banana$	nana$ba	n | ana$b | a
		F L

last column= ‘annb$aa’ (BWT string)

and first column = ‘$aaabnn’

Pros: The BWT rearranges sequences into a format that is highly amenable to compression by grouping identical or similar characters into contiguous blocks, thereby increasing the probability that repeated characters occur adjacently. This compression-friendly structure significantly reduces storage requirements. The BWT is fully reversible, allowing the original sequence to be accurately reconstructed from the transformed sequence.

Cons: Sorting all cyclic rotations of long sequences is computationally intensive in both time and memory, making the classical BWT implementation less suitable for very large datasets and not particularly hardware efficient.

Suggestion: BWT can be effectively used as a data indexing technique that enables searching within large data blocks using relatively small memory footprints. Its efficiency can be significantly improved by integrating the FM-index, which facilitates fast and memory-efficient substring searches. A prominent application of this approach is the Burrows–Wheeler Aligner (BWA) (Li and Durbin, 2009).

1.5 FM index

Ferragina and Manzini (2000) first proposed the FM-index as a space-efficient data structure for fast pattern searching in large texts. The FM-index builds upon the Burrows–Wheeler Transform (BWT) and enables efficient substring queries while using substantially less memory. For a given sequence *s*,

all cyclic rotations of the sequence are first generated and sorted lexicographically. The first column of the sorted matrix is denoted as F(*s*), and the last column corresponds to BWT(*s*). The i^{th} element of BWT(*s*) represents the character that precedes F(*s*)[*i*] in the original sequence *s*.

The FM-index utilizes position-wise base counts to construct a tally (occurrence) matrix, which is fundamental to the Last-to-First (LF) mapping operation. The LF-mapping is defined as:

c=p[i]

lo=firstcol[c][0]+TallyDict[c][lo−1]

hi=firstcol[c][0]+TallyDict[c][hi−1]

where *c* is thei^{th} character of the search pattern *p*, *lo* and *hi* denote the current lower and upper bounds of the search interval, and firstcol is a dictionary storing the starting and ending positions of each character in F(*s*). TallyDict contains cumulative occurrence counts of each character in BWT(*s*).

For example, given the sequence *s*="ATGCCATAAAAA", the resulting BWT(*s*) is "AAAAATC$CGTAA". The corresponding *firstcol* dictionary is {'A':(1,8), 'C':(8,10), '$':(0,1), 'G':(10,11), 'T':(11,13)}, and the *tally* dictionary is {'A':[0,0,1,2,3,4,5,5,5,5,5,6], 'C':[1,1,1,1,1,1,2,3,4,5,5], '$':[0,1,1,1,1,1,1,1,1,1,1,1]}, derived from the base composition of *s* (A: 7, C: 2, T: 2, G: 1, $: 1).

Here is the refined and scientifically improved version:

Pros: The FM-index is highly memory efficient because it does not require storage of the original text or large auxiliary data structures, making it suitable for indexing very large genomic sequences.

Cons: Query operations may require a relatively large number of search steps, which can increase computational time, particularly for long patterns.

Suggestion: The performance of the FM-index can be enhanced by employing multi-step (*n*-step) searching strategies rather than single-step searches. Increasing the step size *n* can significantly reduce the number of iterations required during pattern matching, thereby improving overall search speed. A notable implementation utilizing this approach is Bowtie2 (Langmead and Salzberg, 2012).

1.6 Sketch-based methods

With the rapid growth of sequencing datasets from gigabyte (GB) to terabyte (TB) scales, traditional indexing strategies become increasingly impractical, as they require large memory footprints and often compromise index exhaustiveness. Locality-sensitive hashing (LSH) has emerged as a powerful alternative to address this challenge by converting large sequences into compact representations known as sketches. These sketches preserve similarity information and allow rapid comparison of sequences using similarity measures such as the Jaccard index, enabling efficient approximate matching without performing full-scale alignments.

Pros: LSH dramatically reduces memory usage by representing large sequences as small fixed-size sketches, making it highly suitable for large-scale and metagenomic datasets. It provides very fast similarity search and clustering, supports scalable comparison of millions of sequences, and is alignment-free, which significantly lowers computational complexity. LSH-based methods are particularly effective for detecting approximate similarity, taxonomic classification, and large database screening.

Cons: LSH methods provide approximate rather than exact matches, which may lead to reduced sensitivity for highly divergent or very short sequences. The accuracy of similarity estimation depends on sketch size and hash function selection, and improper parameter choices may introduce false positives or negatives. Additionally, LSH is less suitable for applications requiring precise base-level alignment, such as variant calling and fine-scale mutation detection.

1.7 MinHash based algorithm

To quantify similarity between large biological datasets, the Jaccard similarity coefficient is commonly used. For two sets A={1,2,4,5,6 and B={2,3,4,7}, the Jaccard similarity is defined as:

$JS(A,B)=|A\cap B|/|A\cup B| = |\{2,4\}|/|\{1,2,3,4,5,6,7\}| =27\approx 0.285$.

The Jaccard similarity provides a single numerical measure of similarity (or distance) between two sets.

However, direct computation becomes computationally expensive for large sets, particularly when dealing with terabyte-scale sequencing datasets. MinHash is a randomized, alignment-free technique developed to efficiently approximate the Jaccard similarity between large sets and was originally applied for detecting similar web documents (Baker and Langmead, 2019). In MinHash-based methods, biological sequences are represented as sets of shingles or k-mers. One or more hash functions are applied to these k-mers, and compact sketches are generated by selecting the minimum hash values. The similarity between two sequences is then estimated by comparing their sketches, providing an efficient approximation of the Jaccard index.

MinHash-based algorithms can be further enhanced by employing minimizers as seeds, which improves seed-based sequence searching. These approaches are particularly advantageous for mapping long reads generated by third-generation sequencing technologies such as Oxford Nanopore and Pacific Biosciences, which exhibit relatively high error rates due to frequent insertions and deletions. Recent advancements, such as weighted MinHash, allow incorporation of k-mer multiplicities into similarity estimation, thereby improving accuracy (Ioffe *et al.,* 2010). Increasing the number of permutation (hash) functions improves similarity estimation accuracy but also increases computational cost and memory usage, as additional CPU operations and storage are required for sketch generation.

Pros: MinHash is widely used in large-scale clustering, genome assembly, sequence alignment, and metagenomic classification, where clustering is based on similarity of sequence signatures. It efficiently estimates Jaccard similarity in a probabilistic manner, significantly reducing computational overhead while lowering false-negative rates in large database searches.

Cons: The performance of MinHash is sensitive to the choice of shingle (k-mer) size. Very large k-mers reduce sensitivity to near matches, whereas very small k-mers may lead to spurious matches and biologically meaningless interpretations. In addition, MinHash may still incur non-negligible computational and memory costs for very large datasets.

Suggestion: Weighted MinHash can be employed to account for k-mer multiplicities, improving similarity estimation accuracy. For large-scale parallel implementations, MapReduce-based frameworks can be used to merge MinHash sketches efficiently across distributed computing environments.

Example: Mash (Ondov *et al.,* 2016).

1.8 Locality Sensitive Hashing (LSH) based algorithm

MinHash generates a compact signature matrix in which each column represents a set (e.g., a sequence represented by its k-mer set) and each row corresponds to a hash function. These signatures enable efficient approximation of Jaccard similarity while requiring substantially less storage than full k-mer sets. However, MinHash significantly reduces memory usage, computing pairwise similarities between all signature columns remains computationally expensive for very large datasets, as it still requires comparing all possible column pairs. Furthermore, this exhaustive comparison becomes impractical when dealing with millions of sequences.

To overcome this limitation and restrict comparisons to only promising candidate pairs, Locality-Sensitive Hashing (LSH) is employed. LSH is designed such that similar objects are mapped to the same buckets with high probability, while dissimilar objects are unlikely to collide. In contrast to standard hash functions, where small input changes can produce large output variations, LSH ensures that small changes in the input result in small changes in the output, thereby preserving similarity relationships. This property allows LSH to efficiently identify candidate pairs of similar signatures without performing exhaustive pairwise comparisons.

The MinHash–LSH framework thus enables scalable similarity search by fitting signature columns into memory and drastically reducing the number of comparisons required. Only candidate pairs that fall into the same hash buckets are subsequently subjected to more detailed similarity evaluation. The workflow of the Locality-Sensitive Hashing process is illustrated in **Figs. 12A** and **12B**.

In the LSH framework, similar signature columns are substantially more likely to be grouped into the same bucket than dissimilar ones due to the banding strategy. This approach involves hashing objects multiple times, thereby increasing the probability that similar

items collide in at least one bucket while minimizing collisions among dissimilar objects. Consequently, LSH effectively restricts similarity computations to a reduced set of candidate pairs, significantly improving computational efficiency for large-scale similarity search.

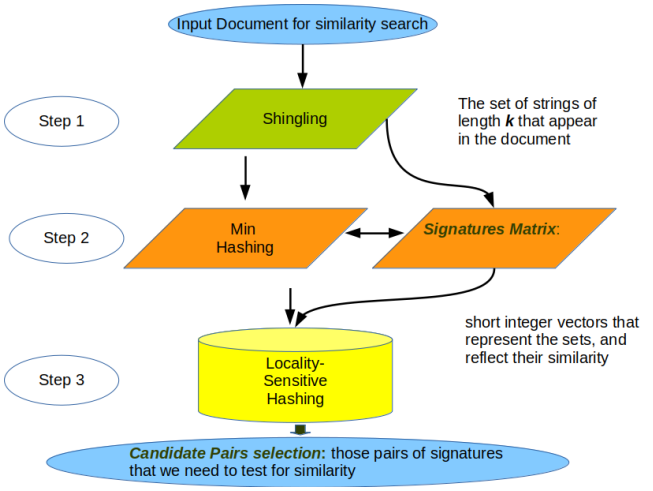

Fig. 12 A. Locality-Sensitive Hashing (LSH) flowchart

The figure illustrates the sequential steps involved in the Locality-Sensitive Hashing workflow, including signature generation, banding, hashing into buckets, and identification of candidate pairs for similarity evaluation.

Pros: LSH has widespread applications in bioinformatics, including metagenomic classification, large-scale similarity searching, and approximate read mapping. It provides high accuracy in detecting similar sequences and is considerably faster than exhaustive MinHash-based comparisons.

Cons: Despite its efficiency, LSH may require substantial memory for maintaining hash tables and bucket structures and can become computationally demanding for extremely large datasets.

Suggestion: The memory footprint of LSH can be reduced by adjusting bucket hashing strategies, such as increasing cut-off threshold values to create larger hash ranges and thereby decreasing the number of stored bucket keys. Furthermore, integrating MinHash with LSH can reduce the linear query time complexity $O(n)$ to sub-linear time, significantly improving scalability.

Example: Genetic distance calculation based on LSH (Pathirana *et al.,* 2020).

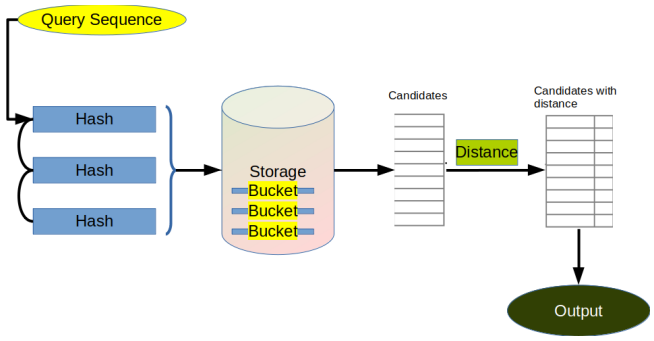

Fig. 12B. Complete workflow of Locality-Sensitive Hashing (LSH)

The figure depicts the full pipeline of the LSH process, detailing signature banding, hashing into buckets, candidate pair generation, and subsequent similarity evaluation steps.

1.9 HyperLogLog sketch-based algorithm

HyperLogLog (HLL) is a probabilistic algorithm used to estimate the cardinality, defined as the number of distinct elements in a multiset. For example, given a multiset M={4,3,6,2,2,6,4,3,5}, the cardinality is five, corresponding to the distinct elements {2,3,4,5,6}. The foundational concept of probabilistic counting was first proposed by Morris (1978) and later formalized and significantly improved by Flajolet *et al.* (2007) through the development of the HyperLogLog algorithm. Computing exact cardinality in large-scale datasets is computationally expensive and memory intensive because every element must be examined and compared. HLL addresses this challenge by constructing a compact sketch that summarizes distinct elements in a multiset, enabling fast and memory-efficient cardinality estimation.

The HyperLogLog algorithm converts hashed input data into binary form and maintains a collection of registers to store summary statistics used to compute an indicator function over the multiset. The input multiset M is partitioned into $m=2^b$ subsets based on the first b bits of the hash values, and each subset is processed independently. Flajolet demonstrated that HLL can accurately estimate cardinalities well above 10^9 with approximately 2% relative error while using as little as 1.5 kB of memory (Flajolet *et al.,* 2007). Compared with MinHash, HLL generally offers higher accuracy and faster performance for cardinality estimation, even when input sets vary widely in size, while maintaining small sketch sizes. HLL sketches can be constructed

directly from FASTA or FASTQ files, although k-mers containing sequencing errors should be filtered prior to sketching. As a probabilistic estimator, HLL uses exponentially decreasing update probabilities such that the stored counters approximate $\log_2(\log_2(n))$, providing dramatic memory savings compared with the $\log_2(n)$ bits required by exact counting or MinHash-based approaches.

Pros: HyperLogLog provides highly memory-efficient and scalable cardinality estimation, supports fast merging of sketches, and is well suited for very large genomic and metagenomic datasets.

Cons: Hash collisions or biased distributions (e.g., excessive leading zeros in hash values) may lead to inaccurate estimates. Additionally, HLL estimates only cardinality and does not retain information about the original elements, making it unsuitable for applications requiring retrieval of individual items.

Suggestion: HLL sketches can be merged to compute union cardinalities efficiently. Further improvements have been introduced in HyperLogLog++, which replaces 32-bit hashes with 64-bit hashes and incorporates enhanced bias-correction techniques. Tools such as Dashing implement HLL for fast genomic sketching. Zhu developed stable data-sketching APIs and Python packages that support MinHash, b-bit MinHash, Weighted MinHash, HyperLogLog, and MinHash-LSH for large-scale biological data analysis (Zhu *et al.,* 2024).

2. SOME EXAMPLES OF IMPORTANT SEQUENCE ALIGNMENT TOOLS

Examples of index-based read aligners include Bowtie and BWA, which employ indexing strategies based on the Burrows–Wheeler Transform (BWT) for fast and memory-efficient sequence alignment. In transcriptome analysis, several specialized spliced aligners have been developed to accurately map RNA sequencing reads that span exon–exon junctions. Prominent examples include STAR, SpliceMap, TopHat (Trapnell *et al.,* 2009), MapSplice, and GEM (Marco-Sola *et al.,* 2012). These spliced aligners are designed to align partial or full-length spliced transcripts, such as expressed sequence tags (ESTs), by exploring potential exon assemblies and detecting exon–intron boundaries that best fit the corresponding target transcripts or proteins.

BLAST : BLAST (Basic Local Alignment Search Tool) is a widely used local alignment program for comparing primary biological sequences, including nucleotide and protein sequences. It employs a heuristic strategy to rapidly identify regions of local similarity by searching for short, high-scoring segment pairs (HSPs) rather than performing exhaustive full-length sequence comparisons. This approach significantly accelerates similarity searches while maintaining high sensitivity, enabling efficient detection of homologous sequences and functional or evolutionary relationships (Altschul *et al.,* 1990).

FASTA: FASTA is a sequence alignment software package based on the Smith–Waterman algorithm for aligning DNA and protein sequences. The FASTA algorithm employs k-tuples (short subsequences) that are generally shorter than the word sizes used in BLAST. Typical k-tuple sizes range from 1-2 for protein sequences and 4-6 for nucleotide sequences. The choice of k-tuple size directly influences computational performance and sensitivity: smaller k-tuples increase sensitivity but require more computation time, whereas larger k-tuples accelerate the search but may reduce sensitivity and increase the likelihood of false positives.

CLUSTALW: CLUSTALW is a widely used multiple sequence alignment tool based on the progressive alignment strategy. It constructs a guide tree from pairwise sequence distances and then sequentially aligns sequences or groups of sequences according to their evolutionary relationships until all sequences are incorporated into a final multiple sequence alignment. CLUSTALW remains one of the most important and commonly used tools for multiple sequence alignment in bioinformatics.

BWA: BWA (Burrows–Wheeler Aligner) is a read-mapping tool based on the Burrows–Wheeler Transform (BWT) and the FM-index, designed for fast and memory-efficient alignment of sequencing reads to reference genomes. Similar to Bowtie, BWA efficiently identifies exact matches; to support inexact matching, it incorporates a backtracking algorithm that allows controlled mismatches and gaps within a defined edit distance, enabling accurate alignment of sequencing

reads containing sequencing errors or biological variations.

Bowtie 2: Bowtie 2 is a fast and memory-efficient read-mapping tool based on the Burrows–Wheeler Transform (BWT) and FM-index. Two main open-source versions are available: Bowtie, which is optimized for short, ungapped alignments, and Bowtie 2, which supports longer reads (typically >50 bp) and gapped alignment. Owing to their differing alignment strategies and heuristics, the two versions can exhibit different performance characteristics when applied to the same experimental datasets.

Dashing: Dashing is a genomic sketching and comparison tool that employs a cardinality estimation approach based on the HyperLogLog (HLL) sketch to efficiently and accurately estimate similarities among large sequencing datasets. It is implemented in C++14 and requires a modern compiler. Dashing operates in two main phases: a sketching phase, in which HLL sketches are generated from input sequences, and a distance-computation phase, which uses OpenMP for multithreaded parallel processing to achieve high performance on multicore systems. The tool also leverages hardware-level optimizations, including Advanced Vector Extensions (AVX512-BW) and SIMD (Single Instruction Multiple Data) instructions, to further accelerate data-parallel computations. Dashing is widely used for mapping sequencing reads, clustering genomes, and identifying similarity patterns that characterize species-level differences in large genomic databases. Unlike MinHash, which stores arrays of relatively large integers, HLL in Dashing stores compact arrays of small registers, resulting in lower memory usage while maintaining high accuracy. The workflow of the Dashing algorithm is illustrated in **Fig. 13**.

Advantages: Dashing outperforms traditional MinHash-based approaches in both accuracy and efficiency across a wide range of sketch sizes and input dataset scales. It can construct sketches and compute pairwise distances for more than 87,000 genomes in approximately six minutes, demonstrating its suitability for large-scale comparative genomic analyses (Cazenave, 2007).

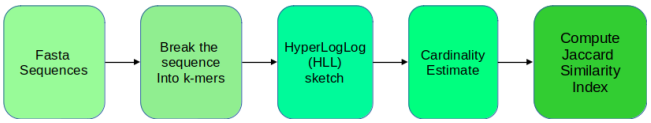

Fig. 13. Process flowchart of the Dashing tool

The figure illustrates the workflow of Dashing, including the sketching phase based on HyperLogLog, multithreaded distance computation, and similarity estimation for large-scale genomic datasets.

3. ASSESSMENT AND COMPARATIVE STUDY OF SEQUENCE ALIGNMENT ALGORITHMS

Computational time and memory consumption are two major factors considered in the assessment and comparative evaluation of sequence alignment algorithms, particularly when dealing with large-scale sequencing data. These performance metrics are influenced by algorithmic complexity, computational efficiency, and alignment accuracy. In comparative studies, algorithms that achieve faster execution with lower resource utilization are generally preferred, provided that alignment accuracy is maintained.

Classical dynamic programming–based alignment algorithms, such as Needleman–Wunsch and Smith–Waterman, have a time complexity of $O(n^2)$, making them computationally expensive and impractical for very large sequences. For multiple sequence alignment, the ClustalW algorithm exhibits a time complexity of $O(N^2n^2)$), where N represents the number of sequences and n denotes the sequence length. In contrast, heuristic-based algorithms significantly reduce computational overhead, achieving time complexity on the order of $O(N\log_2 Nn^2)$, thereby improving scalability for large datasets. Alignment-free sketching approaches, such as HyperLogLog (HLL), further reduce computational cost, with a time complexity of $O(N)$, where N is the number of distinct elements.

Accuracy constitutes another critical evaluation criterion and refers to the extent to which an algorithm produces correct and reproducible outputs for given inputs. An accurate alignment algorithm consistently yields correct alignments across repeated applications and maintains biologically meaningful results. Alignment algorithms must operate within a finite number of computational steps to ensure convergence and usability.

Memory usage is also a key consideration, as some alignment and indexing strategies require substantial intermediate storage, particularly for large reference genomes. For example, FM-index-based aligners may require significant memory during index construction and querying. The space complexity of the HyperLogLog (HLL) algorithm is extremely low, approximately $O(\log_2\log_2(N_{max}))$, making it highly suitable for large-scale genomic data analysis.

4. ASSESSMENT AND COMPARATIVE STUDY OF BWT FM INDEX-BASED ALGORITHMS WITH LINEAR SEARCH METHODS

In this study, we employed a naive yet memory-efficient strategy to mitigate the high memory requirements of the Burrows–Wheeler Transform (BWT) when applied to long genomic sequences. Conventional partial pattern-matching algorithms often suffer from substantial time and space overheads, motivating the development of a novel and fast k-merbased search algorithm for large-scale bioinformatics applications. The proposed approach first applies the BWT to the input sequences and subsequently utilizes the FM-index for read mapping and pattern searching. Emphasis is placed on achieving high-speed pattern searching for big-data scenarios involving large volumes of sequencing data.

Unlike conventional aligners such as Bowtie, the proposed algorithm supports highly flexible mismatch tolerance and can efficiently perform pattern searches even with mismatch rates of 20–30%, which are typically unsupported by standard mappers. The method is particularly optimized to address memory constraints associated with long sequences, including human-scale genomes, and demonstrates significantly reduced search time compared with traditional linear search approaches. A comparative performance analysis between the proposed FM-index based approach, Bowtie, and classical dynamic programming methods is presented in Fig. 14.

For performance evaluation, a multi-FASTA dataset of 13.9 MB comprising 153,041 sequences was used. All possible 6-mer combinations (4096 query sequences) were generated and used as query patterns. All experiments were conducted on a Linux platform (scbb-Precision-T7600; Ubuntu 4.15.0-111-generic, x86_64 GNU/Linux).

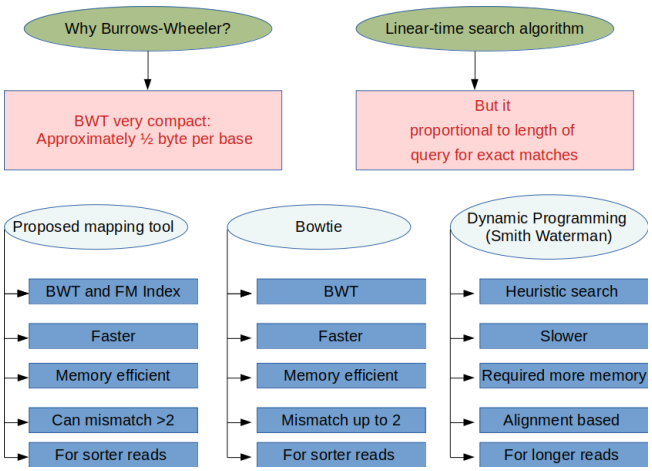

Fig. 14. Comparative study of FM-index based, Bowtie and Dynamic programming

This image depicts the comparative study of FM-index based, Bowtie and Dynamic programming based algorithms.

5. RESULT AND DISCUSSION

For mapping and alignment of sequencing data, two major methodological approaches are widely used: alignment-based methods and k-mer based (alignment-free) methods. Alignment-based approaches rely on

Table 2. Difference between K-mer based methods and Alignment-based methods

K-mer based Methods (Alignment-free)	Alignment-based methods
Based on the presence and frequency of k-mers; generally more memory-intensive due to k-mer indexing.	Computationally expensive because all possible pairwise alignments (including gaps) are evaluated.
Perform k-mer searches by indexing words and their positions from reference sequences.	Rely on optimal alignment scores computed using dynamic programming, which is computationally demanding.
Provide statistical similarity estimates but do not guarantee exact base-level alignments.	Provide exact alignments with statistical significance based on similarity scores.
Use heuristic and probabilistic approaches; low similarity scores may hinder homology inference.	Use deterministic scoring schemes for homology assessment.
More time-efficient and scalable for large datasets.	Less time-efficient, particularly for large-scale datasets.
Commonly applied in sequence similarity searches, clustering, classification, and large-scale phylogenomics.	Primarily used for detailed phylogenomic analysis and precise homology detection.

Table 3. Various alignment and mapping algorithms and their pros and cons

Sl. No.	Algorithm	Advantages	Disadvantages
1.	Dot Matrix Method	Displays all possible matches between two sequences; readily reveals insertions, deletions, inverted repeats, and direct repeats; applicable to both nucleotide and protein sequences.	Does not provide quantitative alignment quality measures or statistical significance; true biological relevance of matches may be unclear.
2.	Dynamic Programming Algorithms (Needleman-Wunsch, Smith-Waterman)	Provide optimal alignments for a given scoring scheme; allow flexible choice of substitution matrices and gap penalties.	Computationally expensive and memory intensive; performance degrades with increasing sequence length.
3.	Progressive Alignment	Efficient for aligning large numbers of sequences; allows adaptive matrix selection and adjustable gap penalties based on evolutionary relationships.	Not globally optimal; early alignment errors propagate; performance degrades for distantly related sequences.
4.	Iterative Method	Improve multiple sequence alignment accuracy through repeated refinement until convergence.	May retain early misalignments; objective functions may be ambiguous.
5.	Heuristic Alignment Algorithms	Very fast and statistically reliable for large-scale searches.	Less sensitive than dynamic programming–based approaches.
6.	Burrows-Wheeler transform	Produces compression-friendly transformed sequences; reversible transformation enables reconstruction of the original sequence.	Sorting of long sequences is computationally expensive and not hardware efficient.
7.	FM-Index	Highly memory efficient; enables fast pattern searching without storing the original text.	Search operations may require many iterative steps, increasing query time.
8.	Min Hashing	Effective for large-scale clustering and similarity estimation using Jaccard similarity; computationally efficient and alignment-free.	Performance sensitive to k-mer size; may incur time and memory overhead; inappropriate k-mer choice can lead to misleading results.
9.	Locality-Sensitive Hashing	Enables fast approximate similarity search; widely used in homology search and metagenomics classification.	Requires substantial memory for hash tables and may be computationally demanding for very large datasets.
10.	HyperLogLog sketch-based algorithm	Provides highly memory-efficient cardinality estimation; suitable for very large datasets.	Hash distribution biases may lead to inaccurate estimates; cannot retrieve original elements.

base-by-base comparison of reads with reference sequences to generate exact or approximate alignments, whereas k-mer based methods represent sequences as collections of fixed-length substrings and estimate similarity based on shared k-mer composition. Each approach offers distinct advantages in terms of accuracy, computational efficiency, memory requirements, and scalability for large datasets. A comparative summary highlighting the key differences between k-mer based and alignment-based methods is presented in Table 2.

Based on the literature review, various sequence alignment algorithms along with their respective advantages and disadvantages are summarized in **Table 3**.

It can be concluded from the comparative study of sequence alignment algorithms that local alignment tools such as BLAST and progressive alignment tools such as ClustalW remain among the most widely used heuristic approaches for multiple sequence alignment. However, with the rapid growth of sequencing data, there is an increasing need to further improve these methods by reducing computational time complexity and enhancing their dynamic and scalable alignment capabilities for large-scale genomic analyses.

6. CONCLUSIONS

Efficient sequence alignment and read mapping are critical steps in the analysis of massive genomic datasets, particularly in next-generation sequencing (NGS) workflows, where accurate mapping to a reference genome forms the foundation for gene expression estimation, variant detection, and gene identification. A wide range of aligner tools are currently available, employing diverse alignment strategies such as heuristic and index-based approaches to meet different analytical requirements. However, genome-specific characteristics especially the presence of repetitive elements pose significant challenges to accurate alignment, particularly when mismatches are allowed.

The time complexity, performance, memory consumption, and specificity of alignment tools are strongly influenced by the underlying algorithm, genome composition, and read properties, including read length, repeat content, and the prevalence of incorrectly mapped or improperly paired reads.

Sensitivity is further affected by genome size, tandem repeats, and read length, making the accurate alignment of long reads with mismatches one of the most demanding tasks in modern bioinformatics. These challenges underscore the need for developing new alignment tools that can effectively balance sensitivity, speed, and memory efficiency.

Our evaluation of an inexact search strategy based on a naive BWT and FM-index approach demonstrates improved performance over existing tools. Owing to its unique sorted-search mechanism, the proposed method achieves faster mapping times while maintaining flexibility in handling mismatches, making it a promising solution for large-scale genomic data analysis.

COMPETING INTERESTS

The authors declare no competing financial interests.

CONFLICT OF INTEREST

The authors declare that there is no conflict of interest.

REFERENCES

Ait Lahcen, L., Yamak, Z., and Morgenstern, B. (2013). DIALIGN at GOBICS Multiple sequence alignment using various sources of external information. *Nucleic Acids Research*, **41(W1)**, W3–W7.

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, **215(3)**, 403-410.

Baker, D.N., and Langmead, B. (2019). Dashing: Fast and accurate genomic distances with HyperLogLog. *Genome Biology*, **20**, 265.

Brinda, K. (2016). Novel computational techniques for mapping and classifying next-generation sequencing data. (Doctoral dissertation). University Paris-Est.

Broder, A.Z. (1997). On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997* (*Cat. No. 97TB100171*) (pp. 21-29). IEEE.

Burrows, M., and Wheeler, D.J. (1994). A block-sorting lossless data compression algorithm (*SRC Research Report No. 124*). Palo Alto, CA.

Canzar, S., and Salzberg, S.L. (2017). Short read mapping: An algorithmic tour. *Proceedings of the IEEE*, **105**(3), 436-458.

Cazenave, T. (2007). Overestimation for multiple sequence alignment. In *Proceedings of the IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology* (pp. 159-164).

Durbin, R., Eddy, S.R., Krogh, A., and Mitchison, G. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge University Press.

Edgar, R.C. (2004). MUSCLE: Multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, **32(5)**, 1792-1797.

Ferragina, P., and Manzini, G. (2000). Opportunistic data structures with applications. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science* (pp. 390-398). IEEE Computer Society.

Feng, D.F., and Doolittle, R.F. (1987). Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, **25**, 351-360.

Flajolet, P., Fusy, E., Gandouet, O., and Meunier, F. (2007). HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm. In *Analysis of Algorithms (pp. 127-146)*.

Gibbs, A. J., and McIntyre, G.A. (1970). The diagram, a method for comparing sequences. *European Journal of Biochemistry*, **16**, 1-11.

Gotoh, O. (1982). An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, **162**, 705-708.

Indyk, P., and Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (pp. 604-613)*. ACM.

Ioffe, M.V., Nishnianidze, D.N., and Valinevich, P.A. (2010). A new exactly solvable two-dimensional quantum model not amenable to separation of variables. *Journal of Physics A: Mathematical and Theoretical*, **43**, 485303.

Kucherov, G. (2019). Evolution of biosequence search algorithms: A brief survey. *Bioinformatics*, **35(19)**, 3547-3552.

Kucherov, G., Salikhov, K., and Tsur, D. (2016). Approximate string matching using a bidirectional index. *Theoretical Computer Science*, **638**, 145-158.

Lam, T. W., Li, R., Tam, A., Wong, S., Wu, E., and Yiu, S.M. (2009). High throughput short read alignment via bi-directional BWT. In *Proceedings of the IEEE International Conference on Bioinformatics and Biomedical Engineering* (pp. 31-36).

Langmead, B., and Nellore, A. (2018). Cloud computing for genomic data analysis and collaboration. *Nature Reviews Genetics*, **19(4)**, 208-219.

Langmead, B., and Salzberg, S.L. (2012). Fast gapped-read alignment with Bowtie 2. *Nature Methods*, **9(4)**, 357-359.

Lee, C.Y., Chiu, Y.C., Wang, L.B., Kuo, Y.L., Chuang, E.Y., Lai, L.C., and Tsai, M.H. (2013). Common applications of next-generation sequencing technologies in genomic research. *Translational Cancer Research*, **2(1)**, 33-45.

Li, H. (2013). Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv*, 1303.3997.

Li, H., and Durbin, R. (2009). Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, **25(14)**, 1754-1760.

Lindner, R., and Friedel, C. C. (2012). A comprehensive evaluation of alignment algorithms in the context of RNA-seq. *PLoS ONE*, **7(12)**, e52403.

Magi, A., Benelli, M., Gozzini, A., Girolami, F., Torricelli, F., and Brandi, M.L. (2010). Bioinformatics for next generation sequencing data. *Genes*, **1(2)**, 294-307.

Makinen, V., Belazzougui, D., Cunial, F., and Tomescu, A.I. (2015). *Genome-scale algorithm design*. Cambridge University Press.

Marco-Sola, S., Sammeth, M., Guigo, R., and Ribeca, P. (2012). The GEM mapper: Fast, accurate and versatile alignment by filtration. *Nature Methods*, **9**, 1185-1188.

Morris, R. (1978). Counting large numbers of events in small registers. *Communications of the ACM*, **21(10)**, 840-842.

Morgenstern, B., Zhu, B., Horwege, S., and Leimeister, C.A. (2015). Estimating evolutionary distances between genomic sequences from spaced-word matches. *Algorithms for Molecular Biology*, **10**, 5.

Muir, P., Li, S., Lou, S., Wang, D., Spakowicz, D.J., Salichos, L., Zhang, J., Weinstock, G.M., Isaacs, F., Rozowsky, J., and Gerstein, M. (2016). The real cost of sequencing: Scaling computation to keep pace with data generation. *Genome Biology*, **17**, 53.

Neilsen, R., Paul, J. S., Albrechtsen, A., and Song, Y.S. (2011). Genotype and SNP calling from next-generation sequencing data. *Nature Reviews Genetics*, **12**, 443-451.

Needleman, S.B., and Wunsch, C.D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, **48**, 443-453.

Nowrousian, M. (2010). Next-generation sequencing techniques for eukaryotic microorganisms: Sequencing-based solutions to biological problems. *Eukaryotic Cell*, **9(9)**, 1300-1310.

Ondov, B.D., Treangen, TJ., Melsted, P., Mallonee, A. B., Bergman, N. H., Koren, S., and Phillippy, A.M. (2016). Mash: Fast genome and metagenome distance estimation using MinHash. *Genome Biology*, **17(1)**, 132.

Pathirana, T., Bandara, S., Gamage, G., Gimhana, N., Wickramarachchi, A., Mallawaarachchi, V., and Perera, I. (2020). Genetic distance calculation based on locality sensitive hashing. *bioRxiv*, 2020-04. https://doi.org/10.1101/2020.04.06.027250

Roberts, M., Hayes, W., Hunt, B.R., Mount, S.M., and Yorke, J.A. (2004). Reducing storage requirements for biological sequence comparison. *Bioinformatics*, **20(18)**, 3363-3369.

Schleimer, S., Wilkerson, D. S., and Aiken, A. (2003). Winnowing: Local algorithms for document fingerprinting. In P*roceedings of the ACM SIGMOD International Conference on Management of Data (pp.* 76-85*)*. ACM.

Smith, T.F., and Waterman, M.S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, **147**, 195-197.

Stephens, Z.D., Lee, S.Y., Faghri, F., Campbell, R.H., Zhai, C., Efron, M.J., Iyer, R., Schatz, M.C., Sinha, S., and Robinson, G.E. (2015). Big data: Astronomical or genomical*? PLoS Biology*, **13(7)**, e1002195.

Tørresen, O.K., Star, B., Mier, P., Andrade-Navarro, M.A., Bateman, A., Jarnot, P., Gruca, A., Grynberg, M., Kajava, A.V., Promponas, V.J., Anisimova, M., Jakobsen, K.S., and Linke, D. (2019). Tandem repeats lead to sequence assembly errors and impose multi-level challenges for genome and protein databases. *Nucleic Acids Research*, **47(21)**, 10994-11006.

Trapnell, C., Pachter, L., and Salzberg, S.L. (2009). TopHat: Discovering splice junctions with RNA-Seq. *Bioinformatics*, **25(9)**, 1105-1111.

Treangen, T.J., and Salzberg, S.L. (2012). Repetitive DNA and next-generation sequencing: Computational challenges and solutions. *Nature Reviews Genetics*, **13**, 36-46.

Wang, Y., Parthasarathy, S., and Tatikonda, S. (2011). Locality sensitive outlier detection: A ranking driven approach. In *Proceedings of the IEEE International Conference on Data Engineering*.

Wood, D.E., and Salzberg, S.L. (2014). Kraken: Ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, **15**, R46.

Zhu, E., Markovtsev, V., Astafiev, A., Khan, A., Ha, C., Łukasiewicz, W., Foster, A., Sinusoidal, S., Thakur, S., Ortolani, S., Titusz, Letal, V., Bentley, Z., fpug, hguhlich, long2ice, oisincar, Assa, R., Ibraimoski, S., Kumar, R., TianHuan, Q., Rosenthal, M.J., Joshi, K., Mann, K., JonR, and Halliwell, J. (2024). ekzhu/datasketch: v1. 5.9. Zenodo.https://doi.org/10.5281/zenodo.11462182