A Categorization of Relevant Sequence Alignment Algorithms with Respect to Data Structures

Hasna El Haji[1], Larbi Alaoui[2]
TIC-Lab, International University of Rabat
Rabat, Morocco

***Abstract*—Sequence Alignment is an active research subfield of bioinformatics. Today, sequence databases are rapidly and steadily increasing. Thus, to overcome this issue, many efficient algorithms have been developed depending on various data structures. The latter have demonstrated considerable efficacy in terms of run-time and memory consumption. In this paper, we briefly outline existing methods applied to the sequence alignment problem. Then we present a qualitative categorization of some remarkable algorithms based on their data structures. Specifically, we focus on research works published in the last two decades (i.e. the period from 2000 to 2020). We describe the employed data structures and expose some important algorithms using each. Then we show potential strengths and weaknesses among all these structures. This will guide biologists to decide which program is best suited for a given purpose, and it also intends to highlight weak points that deserve attention of bioinformaticians in future research.**

Keywords*—Sequence alignment; data structures; bioinformatics

I. Introduction

"Sequence alignment" is a relevant subfield of bioinformatics that has attracted significant interest recently. It focuses on comparing two or more sequences to find homologies and visualize the effect of evolution across a family of genes [1].

Sequences can be divided mainly into two types: genomic and protein. Genomic sequences [2] are chains of nucleotides along a DNA/RNA macromolecule. They can be represented using the alphabet of the four letters of nitrogen bases: A, C, G and T. Protein sequences [2] are chains of twenty types of amino acids along a polypeptide. They can be represented using an alphabet of 20 letters (except B, J, O, U, X and Z) corresponding to the 20 existing amino acids.

Sequence alignment plays a crucial role in biology and medicine. Indeed, it is considered as the basis of many other tasks like phylogenetic analysis, evolution modeling, and prediction of gene expression. An assortment of algorithms has been applied to deal with sequence alignment. We can classify them according to two categories: exact and heuristic algorithms. Giving exact solutions, exact algorithms [3] are considered efficient but slow. Here we can cite Dynamic Programming (DP) developed by R. Bellman (1955). The principle of the DP consists of transforming a sequence S into a sequence Q, using three operations: substitution, insertion or deletion of a character. There is a cost to every operation and the aim is to find the sequence edited with a minimal cost. Considering two sequences S and Q, alignments are mapped to a matrix with entries representing optimal costs. Each cell is calculated based on its preceding cells. Concerning heuristic algorithms [4], they are designed for large databases and they give only approximate solutions to the problem. In fact, sequences are in exponential growth, thus, aligning a sequence against a database containing millions of sequences is not practicable. The aim of heuristic algorithms is to come up with fast strategies to rapidly identify relevant fractions of the cells in the DP matrix.

According to the number of compared sequences, "sequence alignment" is classified into two categories: pairwise alignment and multiple alignment. And according to the aligned regions, "sequence alignment" admits two major categories: global alignment category that aligns the entire given sequence and local alignment category that reveals similarity areas in long sequences. Fig. 1 gives a schematization of the Sequence Alignment Problem.

In the last twenty years a broad range of alignment techniques have been proposed [5]. Citing all these won't be conceivable inside the extent of this work. Moreover, to the best of our knowledge, only a few tools described in the literature have shown efficient results. Indeed, besides giving correct biological conclusions, they have managed to reduce the execution time from several days to a few minutes.

Through this paper, we intend to classify some recent alignment tools according to the widely used data structures. We typically give more interest to those having shown considerable improvement in terms of speed and memory consumption. We will highlight their strengths in sequence alignment tools, and we will also cite some of their drawbacks to reveal their limitations.

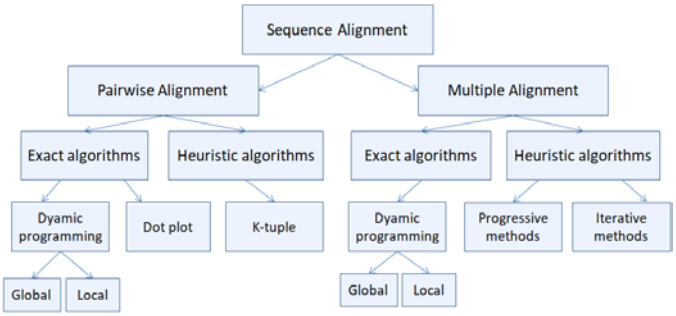

Fig. 1. A Schematization of the Sequence Alignment Problem.

This work is within the framework of the research project "Big Data Analytics - Methods and Applications (BDA-MA)". Author Hasna El Haji is financially supported by a PhD grant of International University of Rabat.

The paper is structured as follows. Most frequently employed methods for sequence alignment are briefly outlined in Section II. Then Section III introduces four of the commonly used data structures in literature and proposes a categorization of the most relevant algorithms based on these structures. Section IV discusses the utility of each structure in "Sequence Alignment". Finally, Section V concludes the paper and Section VI mentions a future work.

II. Related Work

A. Global Alignment

Regarding global alignment, the "Needleman-Wunsch algorithm" [6] is referred as a global alignment method based on dynamic programming. It was implemented in 1970. It gives a score to every possible alignment and tries to find all eventual alignments having the optimal score. It is executed in 4 stages:

1) Fixing the "similarity matrix" and the gap penalty,
2) Initializing the optimality matrix F,
3) Filling in the matrix F,
4) Giving a Traceback.

Let A be a sequence of length l, B be a sequence of length k, d be the gap penalty and $\Delta_{i,j}$ be the score of Match/Mismatch. Table I gives a pseudo-code of the Needleman-Wunsch Algorithm.

TABLE I. A Pseudo-Code of Needleman-Wunsch Algorithm

1: Input: two sequences to align
2: **Initialization:**
3: for i=0..l do:
4: $F_{i,0} = d \times i$
5: for j=0..k do:
6: $F_{0,j} = d \times j$
7: **Recurrence relation:**
8: For i=1..l do:
9: For j=1..k do:
10: $F_{i,j} = max \begin{cases} F_{i-1,j-1} + \Delta_{i,j} \\ F_{i-1,j} + d \\ C_{i,j-1} + d \end{cases}$
11: Output: Optimal alignment

B. Local Alignment

For local alignment, The "Smith-Waterman algorithm" [7] is a basic tool also based on "dynamic programming" but with extra choices to begin and end at wherever. Published in 1981, it maximizes the similarity measure by matching characters or inserting/deleting gaps in 4 steps:

1) Fixing the "similarity matrix" and the gap penalty,
2) Initializing the scoring matrix C,
3) Scoring,
4) Giving a Traceback.

Let A be a sequence of length l, B be a sequence of length k, $\Delta_{ins/del}$ be the gap penalty and $\Delta_{i,j}$ be the score of Match/Mismatch. Table II gives a pseudo-code of the Smith-Waterman Algorithm.

TABLE II. A Pseudo-Code of Smith-Waterman Algorithm

1: Input: two sequences to align
2: **Initialization:**
3: $C_{i,0} = 0$
4: $C_{0,j} = 0$
5: **Recurrence relation:**
6: For i=1..l do:
7: For j=1..k do:
8: $C_{i,j} = max \begin{cases} C_{i-1,j-1} + \Delta_{i,j} \\ C_{i-1,j} + \Delta_{ins/del} \\ C_{i,j-1} + \Delta_{ins/del} \\ 0 \end{cases}$
9: Output: Optimal alignment

C. Pairwise Sequence Alignment

"Pairwise sequence alignment" [8] is applied to examine the similarities of two sequences by finding the best matching alignment of them (the highest score). In other words, for a given sequence and a reference genome, the goal is to find positions in the reference where the sequence matches the best [9]. Three approaches generate the pairwise alignment: dot-plot analysis, "dynamic programming", and k-tulpe methods.

1) Dot-plot analysis (or Dot-matrix method): It is a qualitative and simple tool that compares two sequences to give the possible alignment [10]. Indeed, here are the steps of the method:

a) Two sequences A and B are listed in a matrix,

b) We start from the first character in B, we move over the matrix maintaining the first row and putting a dot in each column where there is a similarity between A and B,

c) The process continues until all possible comparisons between A and B are done. Such main diagonal dots refer to regions of similarity and isolated dots refer to random matches.

2) Dynamic programming: It can achieve global and local alignments. The global is most useful when the query sequences are similar and have the same length. The alignment calculation is generally done with the Needleman-Wunsch algorithm. The algorithm does not calculate the difference between two sequences but rather the similarity. Considering two sequences A and B, a two-dimensional array is filled row after row (starting from the last) and for each row, column after column (starting also from the last) respecting the recurrence relation (1):

$$F_{i,j} = max \begin{cases} F_{i-1,j-1} + \Delta_{i,j} \\ F_{i-1,j} + d \\ C_{i,j-1} + d \end{cases} \quad (1)$$

The local alignment is suitable to deal with dissimilar sequences with eventual similarities in their broader sequence context. An overall alignment would be insignificant. Calculation is generally done with the Smith-Waterman algorithm. The essential difference between Smith-Waterman and Needleman-Wunsch algorithms is that any cell of the initial comparison matrix in Smith-Waterman can be considered as a starting point for the calculation of scores and that any score that becomes less than zero stops the progression of the algorithm, then the score will be reset with the value 0. The calculation is based on the recurrence (2):

$$C_{i,j} = max \begin{cases} C_{i-1,j-1} + \Delta_{i,j} \\ C_{i-1,j} + \Delta_{ins/del} \\ C_{i,j-1} + \Delta_{ins/del} \\ 0 \end{cases} \quad (2)$$

3) K-tulpe methods (or word methods): They are heuristic methods faster than the original dynamic programming algorithms. They actually give only approximate solutions to the problem. K-tuple methods are implemented in the database search tools "FASTA" and "BLAST".

a) FASTA algorithm: An Algorithm for sequence comparison [11] based on the linked list structure: a query sequence is compared to all the strings in the database (DB). It is executed in six stages:

- Search for hot-spots (the largest common sub-sequences),
- Select the 10 best diagonal matches,
- Calculate the best scores for diagonal matches,
- Combine between good diagonal matches and indels,
- Calculate an alternative local alignment,
- Ordering of the results on the sequences of the DB

b) BLAST algorithm [12]*:* It is a tool based on a heuristic method that uses Smith-Waterman program. It looks for regions with strong similarity in alignments without spaces. It improves the speed of FASTA by looking for a smaller number of optimal hot spots. The substitution matrix is integrated from the first stage of hot spot selection.

D. Multiple Sequence Alignment

Multiple sequence alignment (MSA) [13] is a generalization of the "pairwise alignment"; it consists of comparing multiple related sequences. The aim is to deduce the presence of common ancestors between sequences.

Manually aligning more than three sequences can be difficult and time-consuming. Hence a variety of computational algorithms has been developed to accomplish this task. Most MSA algorithms use dynamic programming and heuristic methods (Progressive and Iterative).

1) Dynamic programming: DP is rarely used for more than three sequences because of its high running time and memory consumption. The same principle of DP in pairwise alignment can be applied here to multiple sequences. Unfortunately, the execution time grows considerably in an exponential way in comparison with the size of sequences, which is impractical. Nevertheless, a number of heuristic algorithms are used to accelerate computation. The most widely used heuristic methods today are the progressive and iterative techniques.

2) Progressive methods (tree methods): Invented in 1984, progressive alignment needs initial assumptions about the links between sequences to align, and uses those assumptions to build a guide tree to represent the links. The principle is as follows:

- Two most related sequences are aligned using dynamic programming methods,
- A third one is aligned to the first result,
- The process continues until a unique alignment of all the sequences persists.

The role of the "guide tree" is to choose a sequence to add to the alignment at each step.

The most popular progressive methods used at present are Clustal and T-coffee families.

a) Clustal family: Clustal (cluster analysis of the pairwise alignments) [14] are series of a widely used progressive programs; the original program was developed by Des Higgins in 1988 and was designed specifically to generate MSA on personal computers. The last standard version is ClustalΩ, it was updated in 2018 [32].

All versions of the Clustal family align sequences building progressively a multiple sequence alignment from successive pairwise alignments. This approach is executed in three steps:

- Provide a pairwise alignment,
- Construct a guide tree by a Neighbor -Joining m (developed in 1987 by Saitou and Nei),
- Use the tree to perform a multiple alignment.

b) T-coffee family: T-coffee [15] is a collection of multiple sequence alignment tools. It was originally published in 1998. T-coffee uses a new score function to evaluate the results. The method works through three steps:

- Create a library of all pairwise alignments and build a guide tree,
- Weight alignments by percentage of identical residues,
- Progressively build MSA using tree and weights.

3) Iterative methods: The major issue with progressive alignment is that errors in the initial alignments are transmitted to the whole MSA. Iterative methods [16] attempt to correct this problem by iteratively realigning subgroups of sequences; they start by making an initial global alignment of these subgroups and then revising the alignment to achieve a more efficient result. They can start from an initial MSA done with progressive alignment and then apply some modifications trying to improve it. Iteration is gainful in terms of coding,

time complexity and memory requirements. The most widely used iterative methods are: MAFFT and MUSCLE.

a) MUSCLE: A method based on the guide tree construction technique. It produces a pairwise alignment for progressive alignment and for refinement. The progressive alignment employs a profile function called log-expectation. The refinement applies a tree-dependent restricted partition technique to reduce the execution time of the algorithm [17]. The method consists of three steps:

- Draft progressive: step of multiple alignment. It produces a first guide tree and a progressive alignment using k-mer distance (k-mer is a string part of size k) and log-expectation score.
- Improved progressive: step of building a second guide tree using the Kimura distance. It re-estimates the first tree and produces a new multiple alignment.
- Refinement: step of improving alignment. It refines multiple alignment using the tree-dependent restricted partitioning (deletes edges of guide tree, and re-form the alignment of separated trees).

b) MAFFT: Developed in 2002, the first version of MAFFT [18] was based on progressive alignment and clustering with the Fast Fourier Transform. It had been later provided to deal with large number of sequences and obtain more efficient results in accuracy. It is mainly executed in three stages:

- Detection of regions of similarity with Fast Fourier Transform,
- Application of the basic dynamic programming algorithm to select important strings,
- Build alignment.

III. A Categorization of the Most Efficient Algorithms based on their Data Structures

All the cited methods are the basis of the existing sequence alignment algorithms. To obtain fast and efficient solutions in memory, fundamental methodology in many of them is to build a data structure that occupies a reasonable space of memory. Data structures are one of the most important concepts in programming. They are a way of storing and managing data. Most of the sequence alignment algorithms are built upon such basic data structures like: suffix arrays, suffix trees, hash tables, graphs and others. In this section, we discuss four relevant data structures and we propose a structure-based categorization of the most important algorithms that have marked the last two decades.

A. Suffix Arrays and Suffix Trees

Given a sequence S, a suffix array is an ordered array of all suffixes of S, and a suffix tree is a representation of all suffixes in S in the form of a tree. The latter contains a leaf for each suffix, and each edge is labeled with a string of characters so that the path from the origin to each leaf gives the corresponding suffix.

In "Sequence Alignment", a suffix tree is an index data structure for analogous sequences. It stores all suffixes of an alignment of similar strings. In this category, we will cite 3 important algorithms:

In 2017, paper [19] proposed a multiple sequence alignment algorithm that combines between a suffix tree and a center-star strategy (MASC). The latter transforms an MSA problem into a pairwise alignment, and the suffix tree matches identical regions between two pairwise sequences. The algorithm can be executed in a linear time complexity *O(mn)*, where m is the amount of sequences and n is their average length. The method is also characterized with no loss in accuracy for highly similar sequences.

Earlier, in 2013, article [20] proposed a memory gainful edition of the suffix tree method: "Suffix Array of Alignment (SAA)". It deals with pattern research appropriately like the "Generalized Suffix Array (GSA)". The paper also presents a practical approach for building the SAA. Experiments have shown that the SAA is a relevant data structure for relatively identical strings. It only takes around one seventh of the memory provided by the GSA to process 11 strings.

Suffix trees aim to reduce memory space, they provide a linear space complexity and they allow a linear-time searching [21]. However they could require more than 20 bytes per character, rather, suffix arrays are more useful generally. And here we should cite reference [22] that shows in depth the weakness of suffix trees and their negative effect on the algorithm efficiency. The paper also proved that algorithms based on suffix tree could be replaced with equivalent algorithms based on suffix array, which is memory gainful.

Suffix trees are used for Read Alignment and Whole Genome Alignment while suffix arrays are more adequate to Prefix-suffix Overlaps Computation and Sequence Clustering.

B. Hash-Tables

A hash table is a kind of associative array storing pairs of keys and values. It represents a genome sequence as multiple lists of genomic positions. The concept of hash tables belongs to the heuristic method "BLAST". Indeed, all hash table tools adopt typically one principle.

Author in [23] has developed the first BLAST implementation algorithm such that the stage of "word mapping" is promoted by a hash table. The algorithm was later improved by other researchers involving the notion of parallelization and different other techniques to accelerate alignments. Author in [24] proposed in 2012 an accelerated short read aligner to approximate the original dynamic programming algorithm. It uses a hash table and applies the Needleman-Wunsch algorithm as an extension. The advantage of the hash table here is to reduce the amount of work by avoiding the generation of too many candidate regions.

Author in [25] proposed BFAST, a two-level indexing method. It applies the hash in indexing to decrease the research time. It was introduced to handle the alignment of short human genomic sequences; this makes it an efficient aligning method that is recommended to deal with each number of sequences and every reference genome.

The methods based on hash tables usually provide high sensitivity, but as a limitation, they occupy a lot of memory because the size of the array grows exponentially.

C. Tries

Tries are a kind of trees storing the entire suffixes of a sequence and enabling a quick matching of sequences. They derive from the middle letters of "retrieval". They are efficient in the storage of multiple sequences, and useful in accelerating the process of sequence searching [26].

We have to cite here the reference [27] that proposed in 2018 a fast trie based method for multiple alignment. It bypassed the classic enumeration of successive comparisons with all strings. It also provides an original algorithm combining a trie and an exact algorithm to find the edit distance between strings.

Author in [28] marked 2015 by developing two multiple sequence alignment tools. The first one employed tries to accelerate the alignment of highly similar sequences. The second worked in parallel with Hadoop to deal with big data. Tries worked as a dictionary that stores substrings and indexes. To collect each substring in a long sequence, tries reduce the running time by avoiding the individual substring research.

We also cite [12] that presented BLAT, a tool applying tries to find the regions matching with the query sequence. It demonstrated a considerable progress in accuracy and running time compared to the popular existing tools in the early 2000s.

The practical benefit of tries remains in time reduction. In fact, aligning a sequence to the same duplicates of a string is done only once. This is mainly due to the fact that duplicates fall on the same line in the tree. Though, a hash table should perform an alignment for each duplicate. Thus, tries stay considerably faster than hash tables in time execution.

However, given a reference sequence of length m, a trie can take $O(m^2)$ memory space, which makes it impractical to build a trie for long sequences.

D. Graphs

Recent researches have shown superior accuracy and speed in aligning sequences by using a "Variation Graph" instead of a reference genome. A "Variation Graph" is a directed graph where each edge spells a sequence. In fact, aligning sequences to graph is trying to determine the optimal corresponding path in the graph for every sequence.

In addition to variation graphs, a variety of graph data structures have been studied in the last decades, such as "De Bruijn" graphs, "ABruijn" graphs, String graphs, Partial Order graphs, "Wheeler graphs", etc.

Each of these structures had demonstrated a considerable progress in solving the sequence alignment problem (more details are given in [29]). However, they had registered some weaknesses: high execution time, overlook of arbitrary graphs, imprecise results. In what follows, we will expose three recent methods, based on variation graph, that have influenced more considerably the alignment accuracy.

Authr in [30] suggests PaSGAL, an algorithm to solve sequence-to-graph alignment using parallelism. It is considered as the first parallel algorithm provided for this propose. It generates improved results compared to previous tools on execution time term. Indeed, it allows a decrease from long durations to few hours.

The main idea of the algorithm is to accelerate the "Dynamic Programming" approach on 3 phases: the 1st and the 2nd phases compute the starting and ending cells of the alignment matrix, and the final phase performs a traceback. The latter aims to calculate the "base-to-base" alignment scores required for downstream biological analysis. The algorithm is highly recommended for pan-genomics and antibiotic resistance profiling.

In 2019, the authors of [31] studied two problems and proposed two solutions: a generalization of the "Shift-And algorithm" (designed for exact string matching) to graphs, and a generalization of "Myers' bit vector alignment algorithm" to graphs. Both solutions are based on Needleman-Wunsch algorithm. The paper used a "bit-level parallelization" to estimate the distance between the query sequences and the graph. The method is supposed to fit with the mammalian genome.

The first author of [31] has recently improved his works, and came up with GraphAligner, an efficient sequence-to-graph alignment tool. Compared to existing graph methods, it is 12 times more effective in time complexity. It also takes in consideration long reads error correction and outperforms the current tools 3 times in error rate.

The major advantage of graphs in general is that each stage of the alignment can store and use results from previous alignments. More specifically, Variation graphs become today a reference for analyzing genetic variants.

IV. Discussion

We highlighted the weaknesses and strengths of the most relevant sequence alignment tools and precise their utility in Biology. We aim to facilitate the choice of appropriate tools for each biologist depending on his research intention.

Suffix trees are applicable to Read Alignment and Whole Genome Alignment. Suffix arrays might be also applicable to Read Alignment, but they are more useful in Prefix-suffix Overlaps Computation and Sequence Clustering. Compared to the discussed data structures, hash tables remain more performing in query time execution but they are memory consumers. As application in Bioinformatics, they are more suitable for storing sets of k-mers. Finally, Graphs are relevant in Read Mapping and they can fully represent population-wide variations.

V. Conclusion

This work is a small sample of two decades of scientific production in the field of sequence alignment. In fact, many studies have demonstrated considerable progress in storage and acceleration of the aligning process. The key idea of all the exposed methods is to create data structures to store calculated

scores in the smallest possible space during alignment. That will provide a gain in both memory and runtime.

We started with a classification of the basic methods of sequence alignment. Then we introduced four of the commonly used data structures in literature (suffix arrays/trees, tries, hash-tables and graphs) and we proposed a categorization of the most relevant algorithms based on these. There are such other structures like Burrows-Wheeler transform (BWT), bloom filters, FM-index (combination between the properties of suffix array and the BWT), etc., but we tried to give more attention to the four cited data structures by means of their efficiency in giving the best alignment.

VI. Future Work

From the performance point of view, none of the cited algorithms is yet considered as ideal. A best solution should combine accuracy, speed and small memory space. Furthermore, sequence databases are rapidly and continuously growing, thus the development of high performing methods is still under research.

We believe that graphs, in full development, can be refined further. As a future work, we will give more interest to the sequence-to-graph alignment.

References

[1] B. Chowdhury and G. Garai, "A review on multiple sequence alignment from the perspective of genetic algorithm," Genomics, pp. 419–431, 2017.

[2] J. C. WHISSTOCK and A. M. LESK, "Prediction of protein function from protein sequence and structure," Q. Rev. Biophys., pp. 307–340, 2003.

[3] A. PHILLIPS, D. JANIES, and W. WHEELER, "Multiple sequence alignment in phylogenetic analysis," Mol. Phylogenet. Evol., pp. 317–330, 2000.

[4] İ. Ö. Bucak, V. Uslan, and S. Member, "An analysis of Sequence Alignment : Heuristic Algorithms," pp. 1824–1827, 2010.

[5] H. Ng, S. Liu, and W. Luk, "Reconfigurable Acceleration of Genetic Sequence Alignment : A Survey of Two Decades of Efforts," in 27th International Conference on Field Programmable Logic and Applications (FPL), 2017.

[6] "S.B. Needleman and C.D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins, "J. Mol. Biol., pp. 443-453, 1970.

[7] A. Khajeh-Saeed, S. Poole, and J. Blair Perot, "Acceleration of the Smith-Waterman algorithm using single and multiple graphics processors," J. Comput. Phys., pp. 4247–4258, 2010.

[8] W. Haque, A. A. Aravind, and B. Reddy, "Pairwise sequence alignment algorithms: a survey," in ISTA '09: Proceedings of the 2009 conference on Information Science, Technology and Applications, pp. 96–103, 2009.

[9] K. Salikhov, "Efficient algorithms and data structures for indexing DNA sequence data," Bioinformatics [q-bio.QM]. Université Paris-Est; Université Lomonossov (Moscou), 2017. English. ffNNT : 2017PESC1232ff. fftel-01762479f.

[10] N. GALTIER, M. GOUY, and C. GAUTIER, "SEAVIEW and PHYLO_WIN: two graphic tools for sequence alignment and molecular phylogeny," Bioinformatics, pp. 543–548, 1996.

[11] A. L. Delcher, "Fast algorithms for large-scale genome alignment and comparison," Nucleic Acids Res., pp. 2478–2483, 2002.

[12] W. J. Kent, "BLAT — The BLAST -Like Alignment Tool," Genome Res., pp. 656–664, 2002.

[13] D. W. Mount, "Using Iterative Methods for Global Multiple Sequence Alignment," Cold Spring Harb Protoc, pp. 1–6, 2009.

[14] J. Daugelaite, A. O' Driscoll, and R. D. Sleator, "An Overview of Multiple Sequence Alignments and Cloud Computing in Bioinformatics," ISRN Biomath., pp. 1–14, 2013.

[15] C. Notredame, "Recent progress in multiple sequence alignment: A survey," Pharmacogenomics, pp. 131–144, 2002.

[16] T. Rausch, A. K. Emde, D. Weese, A. Döring, C. Notredame, and K. Reinert, "Segment-based multiple sequence alignment," Bioinformatics, pp. 187–192, 2008.

[17] R. C. Edgar, "MUSCLE: Multiple sequence alignment with high accuracy and high throughput," Nucleic Acids Res., pp. 1792–1797, 2004.

[18] K. Katoh, K. Misawa, K. Kuma, and M. T, "MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform," Nucleic Acids Res., pp. 3059 -3066, 2002.

[19] W. Su, X. Liao, Y. Lu, Q. Zou, and S. Peng, "Multiple Sequence Alignment Based on a Suffix Tree and Center-Star Strategy: A Linear Method for Multiple Nucleotide Sequence Alignment on Spark Parallel Framework," J. Comput. Biol., 2017.

[20] J. C. Na et al., "Suffix array of alignment: A practical index for similar data," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), pp. 243–254, 2013.

[21] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," Briefings In Bioinformatics, pp. 473-483, 2010.

[22] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch, "Replacing suffix trees with enhanced suffix arrays," J. Discret. Algorithms, pp. 53–86, 2004.

[23] P. Krishnamurthy et al., "Biosequence Similarity Search on the Mercury System," J. VLSI Signal Process. Syst., pp. 101–121, 2007.

[24] Y. Chen, B. Schmidt, and D. L. Maksell, "An FPGA Aligner for Short Read Mapping," in 22nd International Conference on Field Programmable Logic and Applications, pp. 511–514, 2012.

[25] N. Homer, B. Merriman, and S. F. Nelson, "BFAST: an alignment tool for large scale genome resequencing," PLoS One, 2009.

[26] J. Wang et al., "Interactive and fuzzy search : a dynamic way to explore MEDLINE," Bioinformatics, pp. 2321–2327, 2010.

[27] P. A. Yakovlev, "Fast Trie-Based Method for Multiple Pairwise Sequence Alignment," Doklady Akademii Nauk, pp. 64–67, 2019.

[28] Q. Zou, Q. Hu, M. Guo, and G. Wang, "HAlign: Fast multiple similar DNA/RNA sequence alignment based on the centre star strategy," Bioinformatics, pp. 2475–2481, 2015.

[29] B. Kehr, K. Trappe, M. Holtgrewe, and K. Reinert, "Genome alignment with graph data structures : a comparison," BMC Bioinformatics, 2014.

[30] C. Jain, A. Dilthey, S. Misra, H. Zhang, and S. Aluru, "Accelerating sequence alignment to graphs," Proc. - 2019 IEEE 33rd Int. Parallel Distrib. Process. Symp. IPDPS 2019, pp. 451–461, 2019.

[31] M. Rautiainen, V. Mäkinen, and T. Marschall, "Bit-parallel sequence-to-graph alignment," Bioinformatics, pp. 3599–3607, 2019.

[32] F. Sievers and D. G. Higgins, "Clustal Omega for making accuratealignments of many protein sequences", Protein Science, pp. 135-145, 2018.