SuffixAligner: A Python-based Aligner for Long Noisy Reads

Zeinab Rabea[1]*, Sara El-Metwally[2]*, Samir Elmougy[3]*, M. Z. Rashad[4]*
Computer Science Department, Faculty of Computers and Information, Mansoura University, Mansoura 35516, Egypt

Abstract*—Third-generation sequencing technologies have revolutionized genomics research by generating long reads that resolve many computational challenges such as long genomics variations and repeats. Mapping a set of sequencing reads against a reference genome is the first step of many genomic data analysis pipelines. Many mapping/alignment tools are introduced and always made different compromises between the alignment accuracy and the resource usage in terms of memory space and processor speed. SuffixAligner is a python-based aligner for long noisy reads generated from third-generation sequencing machines. SuffixAligner follows the seed extending approach and exploits the nature of the biological alphabet that has a fixed size and a predefined lexical ordering to construct a suffix array for indexing a reference genome. A suffix array is used to efficiently search the indexed reference and locate the exactly matched seeds among the reads and the reference. The matched seeds are arranged into windows/clusters and the ones with the maximum number of seeds are reported as candidates for mapping positions. Using real data sets from third-generation sequencing experiments, we evaluated SuffixAligner against lordFAST, BWA, GEM3, and Minimap2, in which the results showed that SuffixAligner mapped more reads compared to the other compared tools. The source code of SuffixAligner is available at: https://github.com/ZeinabRabea/SuffixAligner.

***Keywords*—Long reads sequencing; reads mapping; suffix array; alignment; seed extending; LF mapping**

I. Introduction

Sequencing machines have generated a flood of biological data known as reads. They have revolutionized over three generations of technologies that play a key role in the data volume, read length and accuracy, sequencing cost, and speed. Third-generation sequencing technologies, such as Pacific Biosciences (PacBio) and Oxford Nanopore, produce a high throughput of longer reads with higher error rates than Illumina sequencing machines' short reads [1]. Reads mapping/alignment is the cornerstone in any sequence analysis pipeline and implies finding the nearly matched locations of each read in the reference genome/transcriptome tolerating the mismatches due to sequencing biases and errors. Since the reads mapping/alignment is a complex and resources intensive process, efficient algorithms and data structures are introduced to complete it reasonably [2-4].

Mapping algorithms utilize two data structures: hash tables and suffix/prefix tries to handle the long reads generated from the third-generation sequencing machines [5]. The seed and extended approach are used in most sequence aligners and rely on extracting a set of matched seeds among the reference and read and index them in a hash table [6]. Then, the seeds are extended to find the optimal gapless alignment. Examples of tools that utilize hash tables for indexing a set of seeds are [7] Blast [8], NanoBLASTer [9], LAST [10], and Minimap [11, 12]. Suffix tires based algorithms divide the mapping problem into two steps: 1) finding an exact match between the database and the query read using one of the suffixes stored in suffixes tries, and 2) grouping these hits of exact matches to create the final inexact alignment [13].

Genome indexing is the first step in any mapping data analysis framework in which it aims search quickly and efficiently the reference genome for matching patterns (i.e., reads). Many efficient data structures are introduced to build an index for a reference genome, such as suffix arrays [14, 15], hash tables [16], suffix trees [17], and bloom filters [18]. There is no standard procedure for creating the optimal index file; this depends on how the information is organized and how he intends to access it. The second step is the mapping/alignment process that uses the genome indexing to query the reference and search for the reads and align it to nearly matched locations in the genome[19].

In this paper, we introduce SuffixAligner as a long-read aligner that utilizes a modified version of the suffix array construction algorithm for limited-size alphabets such as DNA/RNA and exploits the nature of their corresponding lexicographical ordering. Suffix array is used in the reference indexing stage to compute Burrows–Wheeler transforms. Reads mapping stage follows the seed and extends approach by extracting a set of matched hits called k-mers among reads and the indexed reference. These k-mers are used as seeds to find long stretches of matches extended in both directions of the sequencing read. The regions between the matched seeds among reads and the reference are aligned using dynamic programming approach.

This paper is organized as follows: Section II demonstrates a previously related work, Section III presents the methodology behind the SuffixAligner and describes its three basic stages, Section IV evaluates the SuffixAligner alignment results against four benchmarking alignment tools using two real datasets, and Section V concludes our paper and provides some future insights.

II. Literature Review

Suffix array plays a key role in indexing large-scale genomes and most aligners that dominate the bioinformatics field rely on suffix array and its enhanced versions to complete the mapping/alignment task [20]. It is a list of indexes corresponding to the sorted suffixes of a given sequence in the lexical order. Its construction algorithm is a resource-expensive process, in which there is always a trade-off between the algorithm running time and its memory usage. Most of the suffix array construction approaches for indexing a reference genome/transcriptome should consider that the sequencing bases have a limited size alphabet (i.e. four letters of DNA/RNA alphabets) and utilize the existing integer-based method for suffix array construction and manipulation [21, 22]. Examples of mapping algorithms that rely on suffix arrays for indexing and searching a reference genome are Bowtie [23], BWA [24], GEM3 [25], and lordFAST [26] (see Table I). The Burrows–Wheeler (BWT) transform is a cornerstone in many alignment/mapping tools developed for short reads and extended to long-read sequencing data. Burrows–Wheeler transform can also be generated from a suffix array and is used to build a compressed and efficient version of a reference genome [27].

TABLE I. DESCRIPTION OF BENCHMARKING ALIGNMENT TOOLS: BWA, GEM3, LORDFAST, AND MINIMAP2

Aligner	Description	Ref
BWA	BWA is based on the FM-index and utilizes the seed-and-extend approach for reads mapping. BWA has three different algorithms: • BWA-backtrack (Illumina short sequencing read) • BWA-SW (long reads ranging from 70bp to 1Mbp) • BWA-MEM(long reads ranging from 70bp to 1Mbp)	[24]
GEM3	GEM3 is based on the FM-index. Best results are produced when the read length up to 1k bases GEM3 uses a Multithread mode to speed up the running time	[25]
lordFAST	LordFAST is based on the FM-index and utilizes the seed-and-extend approach for reads mapping	[26]
Minimap2	Minimap2 collects the minimizers from the reference and indexes them into a hash table using a locally sensitive hashing technique. It utilizes the seed-and-extend approach for reads mapping. The total length of all reads should not exceed the 2 billion bases	[11, 12]

III. METHOD

SuffixAligner is designated for reads generated by third-generation sequencing machines. It aims to find the ideal location for each read in the indexed reference genome using a suffix array. SuffixAligner has three stages: 1) Finding the matched seeds between the genome and read, 2) Determining the part of the genome with a large number of seeds (identifying a matching window), 3) Using Needleman [28] algorithm to align the regions between the matched seeds (see Fig.1) .

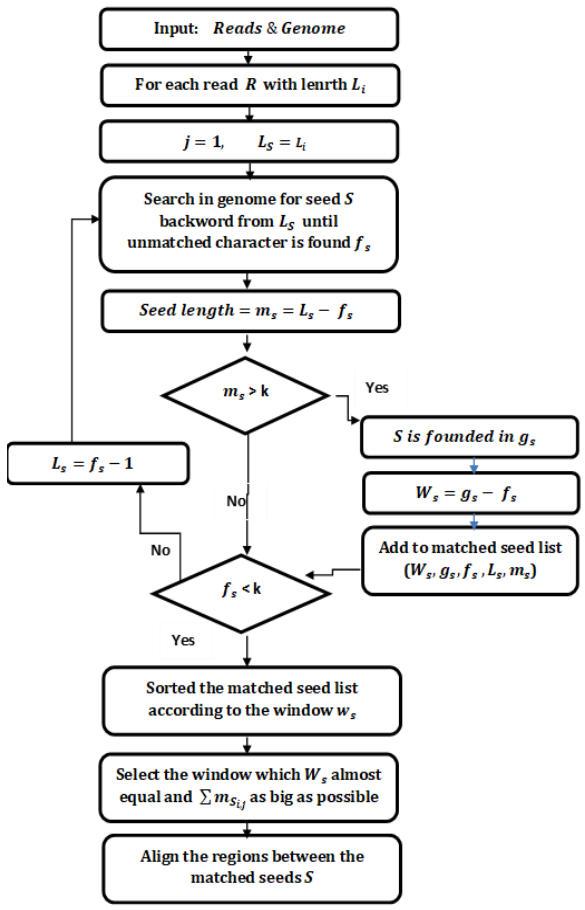
Input: *Reads* & *Genome*
For each read R with lenrth L_i
$j = 1$, $L_S = L_i$
Search in genome for seed S backword from L_S until unmatched character is found f_s
$Seed\ length = m_s = L_s - f_s$
m_s > k
Yes
S is founded in g_s
$W_s = g_s - f_s$
Add to matched seed list (W_s, g_s, f_s, L_s, m_s)
$L_s = f_s - 1$
No
No
f_s < k
Yes
Sorted the matched seed list according to the window w_s
Select the window which W_s almost equal and $\sum m_{S_{i,j}}$ as big as possible
Align the regions between the matched seeds S

Fig. 1. The SuffixAligner proposed approach for mapping long noisy reads.

The reference genome should be indexed first to compute the matched seeds among reads and the reference genome. In the genome indexing stage, SuffixAligner relies on a suffix array construction algorithm proposed in [29] and exploits the nature of biological sequencing data alphabets such as DNA/RNA as it has limited size letters with a pre-defined lexicographical order. The suffix array is constructed for DNA alphabets incrementally, knowing that there are only four letters with the pre-defined sorted order {A, C, G, T}. Each portion of the suffix array will correspond to the set of indexes starting with one of the DNA alphabets and will be used as a seed to compute the next portion corresponding to the next DNA alphabet in the lexical order. The Burrows–Wheeler transform is constructed from the computed suffix array and represents a compressed version of the indexed reference genome. This paper applies the Last-to-First Mapping (LF-array), a mapping function from the last column of the Burrows–Wheeler Matrix (BWM), to the first column of the BWM. To compute LF-array easily and efficiently, BWT is extracted from the last column of BWM. Also, the first column (F_c) is an alphabetical arrangement of the elements of BWT. Instead of storing the first column of BWM, the frequency of each DNA alphabet in BWT is computed along with its start index in BWT. The LF-array is calculated by adding the rank of each DNA alphabet with its starting index (see Fig. 2). LF-array efficiently searches the reference genome for the exact matches (seeds) extracted from the sequencing reads.

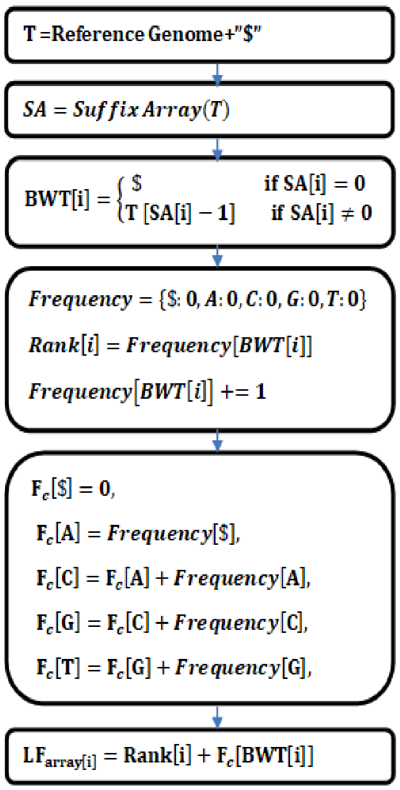
T =Reference Genome+"$"
$SA = SuffixArray(T)$
$BWT[i] = \begin{cases} \$ & \text{if } SA[i] = 0 \\ T[SA[i]-1] & \text{if } SA[i] \neq 0 \end{cases}$
$Frequency = \{\$:0, A:0, C:0, G:0, T:0\}$
$Rank[i] = Frequency[BWT[i]]$
$Frequency[BWT[i]] += 1$
$F_c[\$] = 0,$
$F_c[A] = Frequency[\$],$
$F_c[C] = F_c[A] + Frequency[A],$
$F_c[G] = F_c[C] + Frequency[C],$
$F_c[T] = F_c[G] + Frequency[G],$
$LF_{array[i]} = Rank[i] + F_c[BWT[i]]$

Fig. 2. The Last-to-first mapping (LF-array) computation from the burrows–wheeler transform.

The search starts from the read ending position towards the starting position and stops when mismatches occur. A list of matching seeds between the read and the reference genome, where the seed length of more than 10bp is considered as a valid matched seed. The list contains the starting position f_s of each matched seed in the reference genome g_s, and the ending position l_s and the seed length is m_s which equals $m_s = l_s - f_s$ .The starting position of the expected mapping window of each seed is determined by $w_{s=}g_{s-}\ f_s$. The list of matched seeds is sorted according to the window w_s . The optimal window for mapping the read against a reference genome will be chosen according to the total number of matched seeds in the window. The window with the largest number of seeds will be selected. The regions between the matched seeds in the selected window are aligned using a dynamic programming method such as Needleman algorithm.

IV. EXPERIMENTATIONS AND RESULTS

In this paper, we evaluate the four alignment tools: BWA (bwa-0.7.17) [30], GEM3 [25], lordFAST[26], and Minimap2 [11] against our proposed SuffixAligner using real datasets generated by MinION Oxford Nanopore sequencing technology from two different bacterial strains: Flavobacterium columnare and Aeromonas veronii (see Table II).

We measured the quality of alignment by measuring the alignment rate [31]. The alignment rate is calculated by dividing the number of mapped reads by the total number of reads. Samtools [32] analyzed the SAM files and reports the total number of primary and secondary alignments produced by alignment tools. When a read maps ambiguously to multiple locations, only one of the read alignments is considered as the primary one, and the others are reported as secondary alignments. Supplementary reads are those in which parts of the reads match one location in the genome while other parts match another and often appear in long reads. All the experiments in this work were conducted on a workstation running on Ubuntu Linux with a 3.70 GHz Intel(R) Xeon(R) CPU E5-1620 v2 processor, 64 GB of RAM, and 128 GB SDD hard disk.

TABLE II. CHARACTERISTICS OF THE DATA SET USED EXPERIMENTS

Genome Name	Flavobacterium Columnare	Aeromonas Veronii
Genome Length	3221278	4559061
NCBI	txid996	txid654
Sequencing runs	SRR7449868	SRR7449790
Number Of reads	1142	3994
Number Of Bases	146.1M	644.5M
Max length in reads	62330	53937
Min length in reads	284	279

We evaluated the alignment rate of BWA, GEM3, lordFAST, Minimap2, and SuffixAligner for the real Oxford Nanopore sequencing runs SRR7449868 and SRR7449790. The characteristics of the dataset are shown in Table I. As shown in this table, SuffixAligner has the best Alignment rate. It primary mapped 94% of the total number of reads in the run SRR7449868 of the Flavobacterium columnare reference genome and primary mapped 90% of the total number of reads in the run SRR7449790 of Aeromonas veronii reference genome.

SuffixAligner searches for a one room for every read compared to a reference genome. It considers every read's best mapping position and ignores the other mapping locations. It neglects the secondary and supplementary mapping locations mapped reads (Tables III and IV).

Fig. 3 shows the total number of mapped reads with lengths less than 500bp and greater than 8000bp in the case of the sequencing run SRR7449868. For reads with lengths less than 1000bp, SuffixAligner mapped more reads than other alignment tools; the best results are produced by SuffixAligner, BWA, GEM3, and Minimap2 sequentially. lordFAST ignores reads with lengths less than 1000bp. For reads with lengths greater than 1000bp, SuffixAligner and lordFAST have approximately the same alignment rate then BWA, Minimap2, and GEM3 sequentially.

Fig. 4 shows the total number of mapped reads with lengths less than 500bp and greater than 8000bp in the case of the sequencing run SRR7449790. For reads with lengths less than 1000bp, SuffixAligner mapped more reads than other alignment tools, and the best results are produced by SuffixAligner, BWA, GEM3, and Minimap2 sequentially. lordFAST ignores the reads with lengths less than 1000bp. lordFAST has the best alignment rate with reads lengths between 1000bp and 2000bp, while the SuffixAligner has the best alignment rate with other reads lengths.

TABLE III. EVALUATION OF BWA, GEM3, LORDFAST, MINIMAP2, AND SUFFIXALIGNER ON SEQUENCING RUN SRR7449868 (TOTAL NUMBER OF BASES IS 146.1M)

Metrics / *Tools*	*Total*	*Supplementary*	*Secondary*	*Primary*	*Total mapped*		*Primary mapped*			*Primary unmapped*		
	#	*#*	*#*	*#*	*#*	*%*	*#*	*Bases(kb)*	*%*	*#*	*Bases(Kb)*	*%*
BWA	1325	**183**	0	1142	1222	92%	1039	4027.29	91%	103	87.88	9%
GEM3	1142	0	0	1142	1004	88%	1004	3999.73	88%	138	115.44	12%
lordFAST	1490	126	222	1142	820	55%	472	3723.04	41%	670	392.13	59%
Minimap2	**1518**	137	**239**	1142	**1368**	90%	992	3998.42	87%	150	116.76	13%
SuffixAligner	1142	0	0	1142	1074	**94%**	**1074**	**4072.21**	**94%**	**68**	**42.96**	**6%**

TABLE IV. EVALUATION OF BWA, GEM3, LORDFAST, MINIMAP2, AND SUFFIXALIGNER ON SEQUENCING RUN SRR7449790 (TOTAL NUMBER OF BASES IS 644.5M)

Metrics / *Tools*	*Total*	*Supplementary*	*Secondary*	*Primary*	*Total mapped*		*Primary mapped*			*Primary unmapped*		
	#	*#*	*#*	*#*	*#*	*%*	*#*	*Bases(kb)*	*%*	*#*	*Bases(Kb)*	*%*
BWA	1325	**183**	0	1142	1222	92%	1039	4027.29	91%	103	87.88	9%
GEM3	1142	0	0	1142	1004	88%	1004	3999.73	88%	138	115.44	12%
lordFAST	1490	126	222	1142	820	55%	472	3723.04	41%	670	392.13	59%
Minimap2	**1518**	137	**239**	1142	**1368**	90%	992	3998.42	87%	150	116.76	13%
SuffixAligner	1142	0	0	1142	1074	**94%**	**1074**	**4072.21**	**94%**	**68**	**42.96**	**6%**

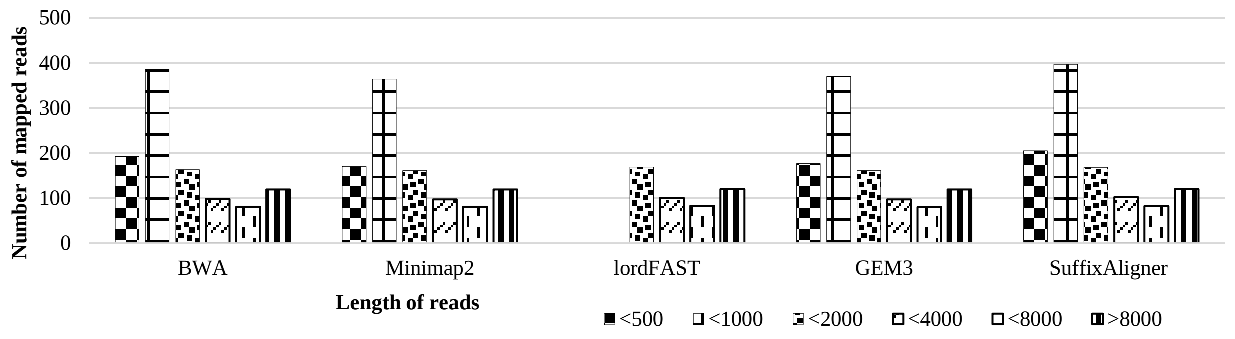

Fig. 3. The total number of mapped reads according to different read lengths in the real sequencing run (i.e., id SRR7449868) with the total number of reads equal to 1142.

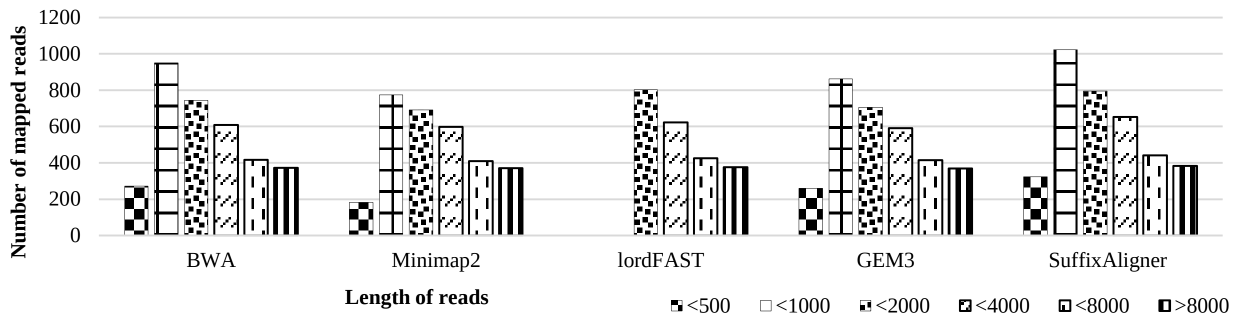

Fig. 4. The total number of mapped reads according to different read lengths in the real sequencing run (i.e., id SRR7449790) with total number of reads equal to 3994.

SuffixAligner maps more reads than lordFAST, BWA, GEM3, and Minimap2 and has the best alignment rate. SuffixAligner and lordFAST have approximately the same alignment rate with the longest read lengths. Also, SuffixAligner maps short reads and reads with lengths greater than 8000bp.

V. CONCLUSIONS

In this paper, SuffixAligner is proposed for reads produced by third-generation sequencing machines such as PacBio and Oxford Nanopore. It relies on a specific type of suffix array constructed for the limited-size alphabets, such as the four-size alphabets of genomic data. It has three basic stages: indexing a reference genome using a suffix array, mapping a set of sequencing reads using the seed and extend approach, and aligning the regions between the matched seeds using a dynamic programming approach. It mapped more reads compared to the other alignment tools and has the best alignment rate accordingly.

SuffixAligner has a long execution running time since it aligns one read at a time. So, as a future work, distributing the computation of mapping reads among different machines to increase the mapping speed and to reduce the memory usage on a single computing node. Also, SuffixAligner may be used to improve the alignment results by mapping only unmapped reads produced by another aligner.

REFERENCES

[1] C. Delahaye and J. J. P. o. Nicolas, "Sequencing DNA with nanopores: Troubles and biases," vol. 16, no. 10, p. e0257521, 2021.

[2] C. Marchet, C. Boucher, S. J. Puglisi, P. Medvedev, M. Salson, and R. J. G. R. Chikhi, "Data structures based on k-mers for querying large collections of sequencing data sets," vol. 31, no. 1, pp. 1-12, 2021.

[3] T. Hu, N. Chitnis, D. Monos, and A. J. H. I. Dinh, "Next-generation sequencing technologies: An overview," vol. 82, no. 11, pp. 801-811, 2021.

[4] V. Bansal and C. Boucher, "Sequencing Technologies and Analyses: Where Have We Been and Where Are We Going?," (in eng), iScience, vol. 18, pp. 37-41, Aug 30 2019.

[5] C. Mingard, J. Wu, M. McKeague, and S. J. J. C. S. R. Sturla, "Next-generation DNA damage sequencing," vol. 49, no. 20, pp. 7354-7377, 2020.

[6] A. Sarkar, S. Banerjee, and S. J. I. T. o. V. L. S. I. S. Ghosh, "An Energy-Efficient Pipelined-Multiprocessor Architecture for Biological Sequence Alignment," vol. 28, no. 12, pp. 2598-2611, 2020.

[7] N. Pavlovikj, E. N. Moriyama, and J. S. Deogun, "Comparative analysis of alignment tools for nanopore reads," in 2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2017, pp. 169-174: IEEE.

[8] M. Johnson, I. Zaretskaya, Y. Raytselis, Y. Merezhuk, S. McGinnis, and T. L. Madden, "NCBI BLAST: a better web interface," Nucleic Acids Research, vol. 36, no. suppl_2, pp. W5-W9, 2008.

[9] M. R. Amin, S. Skiena, and M. C. Schatz, "NanoBLASTer: Fast alignment and characterization of Oxford Nanopore single molecule sequencing reads," in 2016 IEEE 6th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS), 2016, pp. 1-6: IEEE.

[10] S. M. Kiełbasa, R. Wan, K. Sato, P. Horton, and M. C. J. G. r. Frith, "Adaptive seeds tame genomic sequence comparison," vol. 21, no. 3, pp. 487-493, 2011.

[11] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," Bioinformatics, vol. 34, no. 18, pp. 3094-3100, 2018.

[12] H. Li, "Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences," Bioinformatics, vol. 32, no. 14, pp. 2103-2110, 2016.

[13] A. J. T. J. o. C. Chayapathi and M. Education, "Survey and Comparison of String Matching Algorithms," vol. 12, no. 12, pp. 1471-1491, 2021.

[14] A. Amir and I. J. a. p. a. Boneh, "Update query time trade-off for dynamic suffix arrays," 2020.

[15] F. A. Louza, S. Gog, G. P. Telles, F. A. Louza, S. Gog, and G. P. J. C. o. F. D. S. f. S. Telles, "Induced suffix sorting," pp. 23-40, 2020.

[16] T. Maier, P. Sanders, and R. J. A. T. o. P. C. Dementiev, "Concurrent hash tables: Fast and general (?)!," vol. 5, no. 4, pp. 1-32, 2019.

[17] M. M. A. Aziz, P. Thulasiraman, and N. Mohammed, "Parallel generalized suffix tree construction for genomic data," in Algorithms for Computational Biology: 7th International Conference, AlCoB 2020, Missoula, MT, USA, April 13–15, 2020, Proceedings 7, 2020, pp. 3-15: Springer.

[18] M. Najam, R. U. Rasool, H. F. Ahmad, U. Ashraf, and A. W. J. B. r. i. Malik, "Pattern matching for DNA sequencing data using multiple bloom filters," vol. 2019, 2019.

[19] Y. Wu, B. Lao, X. Ma, and G. Nong, "An improved algorithm for building suffix array in external memory," in Parallel Architectures, Algorithms and Programming: 10th International Symposium, PAAP 2019, Guangzhou, China, December 12–14, 2019, Revised Selected Papers 10, 2020, pp. 320-330: Springer.

[20] B. Lao, Y. Wu, G. Nong, and W. H. J. I. T. o. C. Chan, "Building and checking suffix array simultaneously by induced sorting method," vol. 71, no. 4, pp. 756-765, 2021.

[21] A. Das and R. Baruri, "All Pairs Suffix-Prefix Matches using Enhanced Suffix Array," in 2020 International Conference on Smart Electronics and Communication (ICOSEC), 2020, pp. 815-822: IEEE.

[22] F. A. Louza, S. Gog, and G. P. Telles, "Optimal suffix sorting and LCP array construction for constant alphabets," Information Processing Letters, vol. 118, pp. 30-34, 2017.

[23] M. Naghibzadeh, S. Babaei, B. Behkmal, M. J. I. J. o. I. Hatami, and C. T. Research, "The Efficient Alignment of Long DNA Sequences Using Divide and Conquer Approach," vol. 14, no. 3, pp. 48-56, 2022.

[24] S. Liu, Y. Wang, W. Tong, and S. J. B. Wei, "A fast and memory efficient MLCS algorithm by character merging for DNA sequences alignment," vol. 36, no. 4, pp. 1066-1073, 2020.

[25] S. Marco-Sola, M. Sammeth, R. Guigó, and P. J. N. m. Ribeca, "The GEM mapper: fast, accurate and versatile alignment by filtration," vol. 9, no. 12, pp. 1185-1188, 2012.

[26] E. Haghshenas, S. C. Sahinalp, and F. J. B. Hach, "lordFAST: sensitive and fast alignment search tool for long noisy read sequencing data," vol. 35, no. 1, pp. 20-27, 2019.

[27] L. Egidi, F. A. Louza, G. Manzini, and G. P. Telles, "External memory BWT and LCP computation for sequence collections with applications," Algorithms Mol Biol, vol. 14, p. 6, 2019.

[28] S. J. M. B. Needleman, "Wunsch C (1970) J," vol. 48, pp. 444-453.

[29] Z. Rabea, S. El-Metwally, S. Elmougy, and M. Zakaria, "A fast algorithm for constructing suffix arrays for DNA alphabets," Journal of King Saud University - Computer and Information Sciences, 2022/05/06/ 2022.

[30] H. Li and R. J. B. Durbin, "Fast and accurate long-read alignment with Burrows–Wheeler transform," vol. 26, no. 5, pp. 589-595, 2010.

[31] M. R. Stratton, P. J. Campbell, and P. A. Futreal, "The cancer genome," Nature, vol. 458, no. 7239, pp. 719-724, 2009.

[32] P. Danecek et al., "Twelve years of SAMtools and BCFtools," vol. 10, no. 2, p. giab008, 2021.