COT 6417 - Algorithms on Strings and Sequences

Fall 2020

Final Report

Name: Nabila Shahnaz Khan

PID: 5067496

A comprehensive review of papers

A survey of sequence alignment algorithms for next-generation sequencing

&

Fast and accurate short read alignment with Burrows–Wheeler transform

A comprehensive review of the papers “A survey of sequence alignment algorithms for next-generation sequencing” and “Fast and accurate short read alignment with Burrows–Wheeler transform”

Nabila Shahnaz Khan
Computer Science Department, University of Central Florida

I. Introduction:

In this paper, two very well-known papers have been reviewed. The first paper was written by authors Heng Li and Nils Homer. That paper has represented an overall review of all the sequence alignment algorithms and tools. The second paper was written by authors Heng Li and Richard Durbin. In that paper, the authors have proposed a new sequence alignment algorithm, known as the Burrows-Wheeler Alignment tool (BWA).

II. Paper 1 Review: A survey of sequence alignment algorithms for next-generation sequencing

With the advancement of technologies, sequencing techniques have developed significantly, producing high amount of data in short time. Different possible applications of the data has emerged since then such as genome-wide variation, identification of protein binding sites (ChIP-seq / CLIP-seq), gene expression analysis (RNA-seq) and the assembly of new genomes or transcriptomes. But analyzing this huge volume of data is still a challenge. Without proper analysis, the data won’t be of much use. Researches are working on different algorithms and tools to analyze sequencing data. Among these, the sequence alignment approach seems to be the most commonly used and essential one to nearly all applications of sequencing technologies. In this article, the authors have systematically reviewed the current alignment algorithms and discussed their possible applications. They also tried to shed some light on the future aspects of alignment algorithms with respect to the emerging long sequence reads and the prospect of cloud computing.

A. Overview of Alignment Algorithms:

In this paper, the authors represented three different kinds of alignment algorithms based on their auxiliary data structures, known as indices. These three categories are: algorithms based on hash tables, algorithms based on suffix trees and algorithms based on merge sorting. The overall classification of the alignment algorithms presented in this paper has been shown in Figure 1. The algorithms based on hash tables can be further classified into three categories: exact seed match, spaced seed match, gapped seed match. The example of exact seed match algorithm is BLAST while tools like ELAND, SOAP, MAQ, SeqMap, RMAP uses spaced seed match and SHIRMP, RazerS, SSAHA2, BLAT uses gapped seed match. The algorithms based on suffix trees mostly use three types of data structures - suffix tree, implicit suffix tree and FM-index. MUMer, OASIS uses suffix tree, Vmatch, Segmehl uses implicit suffix tree and Bowtie, BWA, SOAP2, BWT-SW uses FM-index. Only tools developed based on merge sorting are Slider and SliderII.

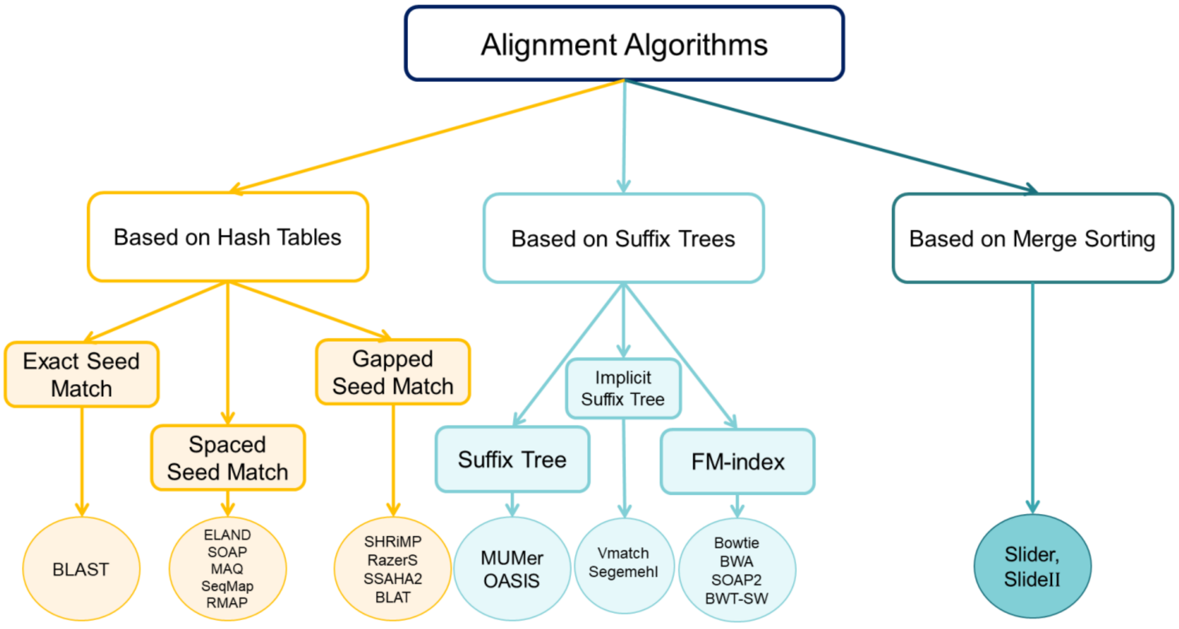

Fig. 1. Overall Classification of Alignment algorithms along with example tools

B. *Alignment Algorithms based on Hash Tables:*

Algorithms based on Hash Tables basically follow the seed-and-extend paradigm. In this approach, the position of each k-mer subsequence of the query is stored in a hash table and the k-mer subsequence is the key. Then the database sequences are scanned for exact matches with the k-mer subsequences. These exact matches are known as seeds. Finally, the seeds are joined and extended using Smith-Waterman algorithm. The final outputs are significant local alignments. Further improvement was done on the basic seed-and-extend approach by introducing concepts liked spaced seeding, q-gram filter and multiple seed hits and constraining seed extension.

- *Spaced Seed:* The concept of spaced seeding, where a seed allows internal mismatches, was first introduced when it was discovered that seeding with non-consecutive matches improves sensitivity. A template '111010010100110111', requiring 11 matches at the '1' positions is 55% more sensitive than BLAST's default template '11111111111' for two sequences of 70% similarity. Tool Eland uses six seed templates spanning the entire short read such that a two-mismatch hit is guaranteed to be identified by at least one of the templates, while SeqMap and MAQ tried to extend the method allowing k-mismatches using $\binom{2k}{k}$ templates, which is exponential in k. Another optimal tool developed based on spaced seed, known as ZOOM, was suggested by Lin et al [1] which is able to identify all two-mismatch hits for 32 bp reads using five seed templates. Apart from the above mentioned tools, the authors of this paper discussed some other alignment tools that were developed based on the concept of spaced seed searching and also talked about their methodology.

- *Q-gram Filter and Multiple Seed Hits:* This technique allows gaps within the seed. It is based on the observation that in case of the occurrence of a w-long query string with at most k differences (mismatches and gaps), the query and the w-long database substring share at least (w + 1) = (k + 1)q common substrings of length q. Methods based on spaced seeds and the q-gram filter are mainly different in that the former category initiates seed extension from one long seed match, while the latter initiates extension usually with multiple relatively short seed matches.
- *Improvement on Seed Extension:* After aligning the seeds to reference sequence, they are extended using dynamic programming which can be time consuming. A major improvement in the field of seed extension comes from the recent advance in accelerating the standard Smith–Waterman with vectorization. The basic idea is to parallelize alignment with the CPU SIMD instructions such that multiple parts of a query sequence can be processed in one CPU cycle. Another improvement is achieved by constraining dynamic programming around seeds already found in the seeding step. Thus unnecessary visits to cells far away from seed hits in iteration are greatly reduced.

C. Alignment Algorithms based on suffix/prefix Tries:

All algorithms in this category essentially reduce the inexact matching problem to the exact matching problem. Typically they involve two steps: identifying exact matches and building inexact alignments supported by exact matches. Three representations are commonly used by these algorithms: suffix tree, enhanced suffix array [2] and FM-index [3]. The advantage of using a trie over a hash table index is that alignment to multiple identical copies of a substring in the reference is only needed to be done once. However, a trie takes $O(L^2)$ space where L is the length of the reference. A suffix tree [4] improves the space complexity by reducing it to O(L) space. Later, enhanced suffix tree was proposed by Abouelhoda et al. [3] that uses even fewer bytes than suffix tree but has an identical time complexity. A further improvement on memory is achieved by Ferragina and Manzini [2] who proposed the FM-index and found that locating a child of a parent node in the prefix trie can be done in constant time using a backwards search on this data structure. As the FM-index was originally designed as a compressed data structure, the theoretical index size can be smaller than the original string if the string contains repeats. To handle inexact matches, some of these algorithms anchor the alignments with maximal unique matches, maximal repeats or exact matches and then join these exact matches with gapped alignments. Examples of tools using this methodology are MUMmer and Vmatch. Tools like Segemehl initiates the alignment with the longest prefix match of each suffix. There are other tools like OASIS, BWT-SW, Bowtie and BWA which follow a different approach for inexact matching.

D. Other Features of Alignment Algorithms:

In their paper, the authors showed that gapped alignment increases sensitivity by a few percent in comparison to ungapped alignment, but does not reduce alignment errors. But it plays a very important role in variant discovery. This is because with ungapped alignment, a read containing an indel polymorphism may still be mapped to the correct position but with consecutive mismatches towards the underlying location of the indel. Alignment errors may be detected and fixed using mate-pair mapping. According to their analysis, paired-end alignment outperforms single-end alignment in terms of both sensitivity and specificity. Using base quality scores improves alignment accuracy because knowing the error probability of each base, the aligner may pay lower penalty for an error-prone mismatch. Long-read aligners should be able to handle alignment gaps for partially aligned read sequences as long reads have greater potential to contain long indels, structural variations and misassemblies. Color-aware alignment algorithm can be a good solution for aligning SOLiD reads. In order to handle spliced read alignment, reads are first aligned to the genome using a standard mapping program, then putative exons are identified, potential junctions are enumerated and finally unmapped reads are aligned against the sequences flanking the possible junctions. The authors have also analyzed the performance and compared a list of free popular short-read alignment software packages to give a better idea about the current alignment tools being used.

E. *Overall Review and Discussion:*

In this paper, the authors have given a precise idea about the different types of alignment algorithms, their strengths, weaknesses and available tools. They have widely discussed the methodology used by different types of tools and also discussed the variations of read alignment. They also proposed some future implications that can be really helpful. Usage of cloud computing, with data uploaded and analyzed in a shared cloud seems like a good suggestion given the current technological advancement. Simultaneous alignment against multiple genomes is another good suggestion made by them as important information might be lost when reads are aligned to a single genome. But, they didn't focus much on the application point of view. An analysis on which type of alignment algorithm might be a fair good choice for a certain type of application of sequencing could have been of utter benefit on the part of the readers.

III. Paper 2 Review: Fast and accurate short read alignment with Burrows–Wheeler transform

In this paper, authors have developed a short read alignment tool named Burrows-Wheeler Alignment tool (BWA) which efficiently aligns short sequencing reads against a large reference sequence using backward search with BWT. It allows mismatches and gaps. It outputs the distinct substrings that are less than k edit distance away from the query read. As input, it can handle Illumina sequence reads up to 100bp. Its alignment output is in SAM format. It performs two types of searches, exact searching and inexact searching.

A. *Methodology Followed:*

The prefix trie for string X is a tree where each edge is labeled with a symbol and the string concatenation of the edge symbols on the path from a leaf to the root gives a unique prefix of X. Using backward search with BWT, the authors of this paper were able to effectively mimic the top-down traversal on the prefix trie of the genome with relatively small memory footprint. In this paper, the authors have first constructed suffix-array to generate BWT. To reduce memory usage even further, here authors used the algorithm proposed in BWT-SW (Lam et al., 2008). Here, authors highlighted the observation that, if string W is a substring of X, the position of each occurrence of W in X will occur in an interval in the suffix array, as all the suffixes that have W prefix are sorted together. Based on this observation, they introduced $[\underline{R}(\mathrm{W}), \overline{R}(\mathrm{W})]$ interval, known as SA interval. $\underline{R}(\mathrm{W})$ and $\overline{R}(\mathrm{W})$ are defined as below:

$\underline{R}(\mathrm{W})$ = min{ k: W is the prefix of $X_{S(k)}$ } .... (1)
$\overline{R}(\mathrm{W})$ = max{ k: W is the prefix of $X_{S(k)}$ } .... (2)

If W is an empty string, $\underline{R}(\mathrm{W}) = 1$ and $\overline{R}(\mathrm{W}) = n - 1$; n is the length of the reference sequence. The set of positions of all occurrences of W in X is:

$\{S(k): \underline{R}(\mathrm{W}) \leq k \leq \overline{R}(\mathrm{W})\}$ .... (3)

The authors pointed out that, sequence alignment is equivalent to searching for the SA intervals of substrings of X that match the query.

B. BWA for Exact matching: backward search

Here, to perform exact matching, the authors defined two new concepts, known as C(a) and O(a, i). Here, C(a) represents the number of symbols in X[0, n−2] that are lexicographically smaller than a ∈ Σ and O(a, i) represents the number of occurrences of a in B[0,i]. Ferragina and Manzini (2000) proved that if W is a substring of X then,

$\underline{R}(\mathrm{aW})$ = C(a) + O(a, $\underline{R}(\mathrm{W}) - 1$) + 1 .... (4)
$\overline{R}(a\mathrm{W})$ = C(a) + O(a, $\overline{R}(a\mathrm{W})$) .... (5)

Now, if aW is a substring of X, then it can be said that, $\underline{\mathrm{R}}(\mathrm{aW}) \leq \overline{\mathrm{R}}(\mathrm{aW})$. Using this finding they have tested whether W is a substring of X and counted the occurrences of W in O(|W|) time by iteratively calculating $\underline{\mathrm{R}}$ and $\overline{\mathrm{R}}$ from the end of W. They referred to this procedure as backward search.

C. BWA for Inexact matching: bounded traversal/backtracking

In order to perform inexact matching, the authors here used a recursive algorithm to search for the SA intervals of substrings of X that match the query string W with no more than z differences (mismatches or gaps). To find distinct substrings from the genome, backward search has been used. This process is bounded by the D(·) array where D(i) is the lower bound of the number of differences in W[0,i]. The better the D is estimated, the smaller the search space and the more efficient the algorithm is. A naive bound is achieved by setting D(i)=0 for all i, but the resulting algorithm would be less efficient as it is exponential in the number of differences. Using the proposed algorithm, all the intervals allowing maximum z differences can be found.

While practically implementing the algorithm, the authors had to make various modifications such as: using different penalties for mismatches, gap opens and gap extensions, using a heap-like data structure to keep partial hits, adapting iterative strategies to accelerate BWA even further and setting a limit on the maximum allowed differences in the first few tens of base pairs on a read.

D. *Strengths:*

- *Memory Efficiency:* This algorithm only needs to store BWT array and a small fraction of O and S array. S is recalculated using inverse compressed suffix array Ψ^{-1}. The memory required for storing BWT array B is n bits, O requires $n\lceil \log n \rceil/32$ bits and S requires $n\lceil \log n \rceil/32$ bits. The memory requirement is doubled if the process is bounded by D. So overall, the total memory requirement is $4n + n\lceil \log_2 n \rceil/8$ bits which is fairly less compared to other tools as they require large memory to build an index.
- *Time Efficiency:* In prefix trie, exact repeats are collapsed on one path on the prefix trie and so there's no need to align the reads against each copy of the repeat. As BWA mimics the concept of prefix trie, this fact applies for it as well and makes it a very efficient algorithm. Evaluating the tools on both simulated and real data, the authors showed that BWA is ~10 – 20× faster than MAQ, while achieving similar accuracy.
- *Gapped alignment for single-end reads:* It supports gapped alignment for single-end reads, which is increasingly important for long reads because long reads tend to contain indels.
- *Base Space Reads:* BWA supports both base space reads, example: Illumina sequencing machines, and color space reads from AB SOLiD machines.
- *Paired-end mapping:* It can handle paired-end mapping as well as single-end mapping.

E. *Weakness:*

- *Ambiguous bases:* Here, the authors have converted Non-A/C/G/T bases on reads into mismatches and Non-A/C/G/T bases on the reference genome to random nucleotides. But this might lead to a possibility of false hits at regions full of ambiguous biases.
- *Allows k mismatches:* Given a read of length m, BWA only tolerates a hit with at most k differences (mismatches or gaps), where k is chosen such that < 4% of m-long reads with 2% uniform base error rate may contain differences more than k.
- *Overestimating Mapping Quality:* While calculating a mapping quality score for alignment, BWA may overestimate mapping quality due to assuming that a true hit can always be found.
- *Handling Long Reads:* Although in theory BWA works with arbitrarily long reads, its performance is degraded on long reads especially when the sequencing error rate is high.
- *Aligning Full Read:* BWA always requires the full read to be aligned but longer reads are more likely to be interrupted by structural variations or misassemblies in the reference genome, which will eventually fail the algorithm.

F. *Overall Review and Discussion:*

The authors evaluated BWA on both simulated data and real data and compared with state-of-the-art tools like Bowtie, MAQ, SOAP2. Overall, it has been proved to be pretty efficient compared to the other tools and error rate was very low in most of the cases. Its memory consumption is also pretty reasonable and supports different types of alignments such as single-end, paired-end, base space reads, color space reads, short reads, long reads. Though its error rate is comparatively high while handling long reads, this issue can be solved by dividing the long reads into multiple short fragments. Its common

SAM output format makes it compatible for variant calling and other downstream analyses after the alignment. All these features have made BWA a highly preferred tool for the researchers over the time.

References:

[1] Lin H, Zhang Z, Zhang MQ, et al. ZOOM! Zillions of oligos mapped. *Bioinformatics 2008*; 24:2431 – 7

[2] Abouelhoda MI, Kurtz S, Ohlebusch E. Replacing suffix trees with enhanced suffix arrays. *J DiscreteAlgorithms 2004; 2: 53–86*.

[3] Ferragina P, Manzini G. Opportunistic data structures with applications. *In: Proceedings of the 41st Symposium on Foundations of Computer Science (FOCS 2000), Redondo Beach, CA, USA. 2000; 390*–8

[4] Manber U, Myers EW. Suffix arrays: a new method for on-line string searches. *SIAM J Comput 1993;22: 935–48.*