A Comparison of *Seed-and-Extend* Techniques in Modern DNA Read Alignment Algorithms

Nauman Ahmed
Computer Engineering Lab
Delft University of Technology
2628CD Delft,The Netherlands
n.ahmed@tudelft.nl

Koen Bertels
Computer Engineering Lab
Delft University of Technology
2628CD Delft,The Netherlands
k.l.m.bertels@tudelft.nl

Zaid Al-Ars
Computer Engineering Lab
Delft University of Technology
2628CD Delft,The Netherlands
z.al-ars@tudelft.nl

***Abstract*—DNA read alignment is a major step in genome analysis. However, as DNA reads continue to become longer, new approaches need to be developed to effectively use these longer reads in the alignment process. Modern aligners commonly use a two-step approach for read alignment: 1. seeding, 2. extension. In this paper, we have investigated various seeding and extension techniques used in modern DNA read alignment algorithms to find the best seeding and extension combinations. We developed an open source generic DNA read aligner that can be used to compare the alignment accuracy and total execution time of different combinations of seeding and extension algorithms. For extension, our results show that local alignment is the best extension approach, achieving up to 3.6x more accuracy than other extension techniques, for longer reads. For seeding, if BLAST-like seed extension is used, the best seeding approach is identifying all SMEMs in the DNA read (e.g., approach used by BWA-MEM). This combination is up to 6x more accurate than other seeding techniques, for longer reads. With local alignment, we observed that the seeding technique does not impact the alignment accuracy. Furthermore, we showed that an optimized implementation of local alignment using vector instructions, enabling 4.5x speedup, makes it the fastest of all extension techniques. Overall, we show that using local alignment with non-overlapping maximal exact matching seeds is the best seeding-extension combination due to its high accuracy and higher potential for optimization/acceleration for future DNA reads.**

I. Introduction

High throughput DNA sequencing techniques have caused an enormous decrease in the cost of whole genome sequencing [1]. This decrease has ushered a new era of genome analysis for a large number of applications like genetic disease diagnosis, personalized medicine, agriculture and livestock trait selection. To extract meaningful information from the sequenced genome, it has to pass through various DNA sequence analysis stages. *DNA read alignment* or *DNA read mapping* is the core stage in this analysis. The DNA sequencing machines output the sequenced genome in the the form of millions of short DNA sequences without giving any information about their actual location in the genome. These short DNA sequences are known as *DNA reads* or simply as *reads*. In read alignment, the task is to find the actual location of these DNA reads within a *reference genome* of the species to which the sequenced genome belongs.

To align a DNA read of length m to a genome of length n, a dynamic programming algorithm (e.g., Smith-Waterman, Needleman-Wunsch or alike) will require $O(nm)$ computation steps. For a human reference genome, $n \approx 3$ billion characters (or bases) long, and therefore, a straight forward application of a dynamic programming algorithm is impractically slow. Moreover, in a typical DNA sequencing experiment, there are hundreds of millions of DNA reads that need to be aligned against the genome. Most modern DNA read aligners tackle this problem by using the *seed-and-extend* approach. The observation behind this approach is that two highly matching sequences contain short substrings that are exactly (or nearly exactly) matching. This approach, pioneered by BLAST [2], aligns a DNA read in two steps: 1. *seeding* and 2. *extension*. Figure 1 shows the seeding and extension phases in a DNA read aligner. During seeding, the aligner first finds substrings of a DNA read that are exactly matching (or nearly exactly) in the genome at one or more than one places. These substrings are known as *seeds*. During extension, the read is aligned to the region around the location of the seed. Such aligners are called seed-and-extend aligners. Many modern DNA read aligners like Novoalign [3], BWA-MEM [4], Bowtie2 [5], and Cushaw2 [6] are seed-and-extend aligners. Table I shows the different seeding and extension strategies used by these aligners (see Section III). In this work, we compare 4 seeding and 3 extension algorithms found in contemporary DNA read aligners.

TABLE I. Seeding and extension techniques used in modern DNA read aligners

Aligner	Seeding				Extension		
	all-SMEM	nov-SMEM	fix-len (0)[1]	fix-len (1)[2]	global	local	BLAST-ext
BWA-MEM	✓	✓					✓
Bowtie2			✓	✓	✓	✓	
Novoalign			✓			✓	
Cushaw2	✓					✓	

[1] 0 mismatch [2] at most 1 mismatch

This paper has the following contributions:

- We developed an open source, generic DNA read aligner that can be used with different seeding and extension techniques [7].
- We compared different combinations of seeding and extension techniques for short as well as long read lengths in terms of accuracy and speed to find the best combinations.

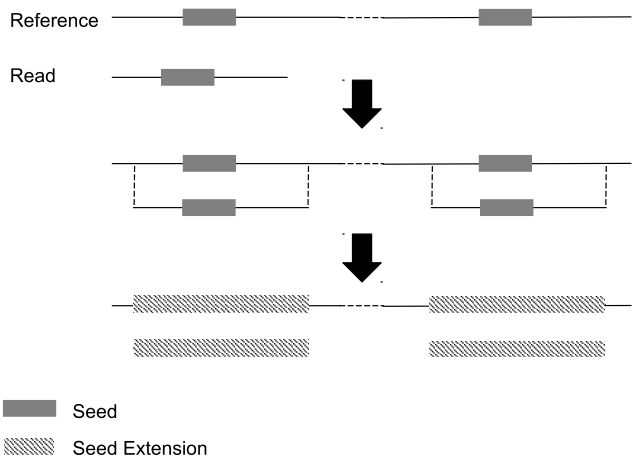

Fig. 1. Seeding followed by extension of short DNA reads against a reference genome

- We optimized the code of the local alignment extension technique to achieve the shortest runtime and one of the highest accuracy combination for longer reads

The rest of the paper is organized as follows: Section II describes the motivation for this paper. Section III and IV introduce different seeding and extension techniques used in the comparison, respectively. Section V presents the details of the DNA read aligner that we have implemented for the comparison. Results of the comparison are shown in Section VI. Finally, we conclude the paper in Section VII.

II. Motivation

Aligners need to be fast and accurate, and have to rely on various heuristics to find a good balance between speed and accuracy. Seeding followed by extension is a heuristic used by many modern DNA read aligners. With the growing importance of genome analysis, many fast and accurate seed-and-extend DNA read aligners have been proposed in recent times, each having its own seed-and-extend method. The accuracy and execution time of a DNA read aligner heavily depends upon the type of seeding and extension technique used.

There are many comparisons of DNA read aligners in the literature. A more recent one is given in [8]. There is also a web based tool for comparing the accuracy of different read aligners [9]. These comparisons evaluate the complete DNA read aligner without focusing on the individual stages of a DNA read aligner. Li and Homer [10] describe different read alignment techniques used by read aligners. They also give an overview of different seeding techniques without discussing their effect on execution time and accuracy on DNA read alignment. A comparison of different kinds of fixed length seeds is given in [11]. The effect of these fixed length seeds on the mapping accuracy and execution time of the DNA read alignment is not discussed. Maximal exact matching seeds are not part of the discussion of any previous comparison. In addition, no previous research has discussed the effect of different extension algorithms on the mapping accuracy and time of the DNA read alignment. Hence, all earlier comparisons in the literature lack the measurement of the contribution of the algorithms used in the seeding and extension phase of the aligner on the accuracy and total execution time of the DNA read alignment. In this paper we will perform such analysis.

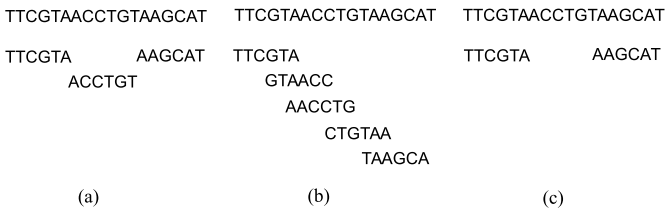

Fig. 2. Simple seeding example (a) seed length = seed interval (b) seed length $>$ seed interval (c) seed length $<$ seed interval

The decreasing cost of DNA sequencing will make computer-based analysis more viable for different application domains. Due to this reason one can foresee the emergence of more DNA read alignment algorithms in the future. This paper will serve as a guideline for developers of DNA read aligners in selecting an appropriate algorithm in the seeding and extension phases of the read alignment process.

III. Seeding Techniques

A seed is a substring of the DNA read that is exactly (or nearly exactly) matching in the genome at one or more than one places. Modern DNA read aligners use two kind of seeds: (i) fixed length seeds, and (ii) maximal exact matching seeds. As listed in Table I, Novoalign and Bowtie2 use fixed length seeds while BWA-MEM and Cushaw2 use maximal exact matching seeds.

A. Types of Seeds

1) Fixed length seeds: In this seeding scheme all the seeds have the same fixed length. They are simply overlapping or non-overlapping substrings of the read, all having the same length. Two parameters control the number of seeds generated from a DNA read. The *seed length* and *seed interval*. The seed interval is the number of the DNA read symbols between starting point of two consecutive seeds. Figure 2 shows seeds of DNA read for different relationships between seed length and seed interval. Decreasing the seed length and/or seed interval increases the number of the seeds which increases the sensitivity but at the same time increases the number of candidate seeds to be extended in the extension phase of the read alignment resulting in an increase in the computation time. It is also possible to allow mismatches in a seed. Such seeds are known as *spaced seeds*. Novoalign uses fixed length exact matching seeds. Bowtie2 allows the user to choose between fixed length exact matching seeds and fixed length seeds with at most 1 mismatch.

2) Maximal exact matches: A *maximal exact match* (MEM) is the longest exact match that cannot be further enlarged in either direction. Let P and T be the DNA read and reference string, respectively. Let $P[i, j]$ and $T[i, j]$ be defined as substring of P and T, respectively, starting from the ith

symbol and ending at jth symbol. Then a MEM of the read can be defined as a tuple $(P[q,r], T[m,n])$ such that

$$P[p] = T[t] \quad \forall p \quad q \leq p \leq r$$
$$\forall t \quad m \leq t \leq n$$

and

$$P[q-1] \neq T[m-1]$$
$$P[r+1] \neq T[n+1]$$

A more refined form of the MEM is proposed in [12] and is called as *super maximal exact match* (SMEM). A MEM which is not contained in any other MEM of the read is known as SMEM. Let there be k MEMs of a DNA read: MEM_1= $(P[q_1,r_1], T[m_1,n_1])$, MEM_2= $(P[q_2,r_2], T[m_2,n_2])$ ... MEM_k= $(P[q_k,r_k], T[m_k,n_k])$. Then MEM_i for $i = 1 \ldots k$ is an SMEM if and only if:

$$(q_i < q_j \text{ or } r_i > r_j) \text{ and } (m_i < m_j \text{ or } n_i > n_j)$$
$$\forall j \quad j = 1, 2, \ldots i-1, i+1 \ldots k-1, k$$

In this work, a seeding technique which finds all the overlapping and non-overlapping SMEMs in a read will be called as *all-SMEM*, whereas the scheme in which only the non-overlapping SMEMs are generated will be called as *nov-SMEM*. As an example, consider the following genome:

$CCAATGTCTCATGGTGTCTCAGCTCTCAGAATTCAGATC$

and a DNA read:

$CAATGTCTCAGATAA$

The all-SMEM seeds of the this read are $CAATGTCTCA$, $TGTCTCAG$, $TCAGAT$ and AA. The nov-SMEM seeds are $CAATGTCTCA$, GAT and AA. For the same seed setting (i.e., minimum required seed length) all-SMEM is more sensitive than nov-SMEM but nov-SMEM is faster.

B. Seed Computation

Seed computation refers to finding the seed sequence and computing its starting position(s) in the reference genome. As described above seeds are substrings of the read that are exactly (or nearly exactly) matching at one or more than one places in the reference genome. Computation of a seed requires a pre-built index of the reference genome. Different kinds of genome indexes can be built. Here we will only focus on those which are found in modern DNA read aligners. Contemporary DNA read aligners compute seeds by either using hash table index (e.g., as in Novoalign) or using FM-index [13] (e.g., as in BWA-MEM, Bowtie2 and Cushaw2).

1) FM-index: FM-index [13] is a memory efficient index of the reference genome. It is a representation of the suffix/prefix trie of the reference genome. Other representations also exist like suffix array [14] and enhanced suffix array [15], but FM-index has the smallest memory footprint. The FM-index consists of three arrays: (1) The count array C, (2) the BWT array B, and (3) the suffix array SA. The count array has four entries, one for each of the four DNA base symbols (i.e., A, C, T and G). An entry for symbol e stores the number of symbols in the reference DNA that are lexicographically smaller than e. The BWT array is the Burrows-Wheeler transform of the reference DNA. The suffix array holds the starting positions of the suffixes of the reference DNA. Computing a seed using FM-index is a two step process:

Step 1—Computing suffix array interval: Given a seed W of length $|W|$ and the FM-index of the reference DNA T, the suffix array interval of W can be computed using Algorithm 1. The proof of the algorithm is given in [13].

The algorithm returns the suffix array interval of W written as $[I_l, I_u]$ where:

$$I_l(W) = \min\{i : W \text{ is the prefix of } SA(i)\}$$
$$I_u(W) = \max\{i : W \text{ is the prefix of } SA(i)\}$$

From I_l to I_u are the suffix array indexes of all those suffixes of T in which W is the prefix. The algorithm returns an empty interval if W is not present in the reference DNA.

Algorithm 1: Backward Search using FM-index

Input: String W and length of reference genome $|T|$. B and C array are assumed to be known
Output: Set of suffix array intervals $[I_l, I_u]$ of W and the match length

```
Function BackwardSearch(W, |T|) begin
    Initialize [I_l, I_u] as [0, |T| − 1]
    i ← |W| − 1
    // match_len is used to compute nov-SMEM
    // match_len ← 1
    while I_l ≤ I_u and i > −1 do
        I_l ← C[W[i]] + Occ(W[i], I_l − 1)
        I_u ← C[W[i]] + Occ(W[i], I_u) − 1
        i ← i − 1
        match_len ← match_len + 1
    // Uncomment the following line to find nov-SMEM
    // return ([I_l, I_u], match_len)
    if i = −1 and I_l ≤ I_u then
        return ([I_l, I_u])
    else
        // return empty interval
        return ∅
Function Occ(a, j) begin
    x ← 0
    y ← 0
    while x ≤ j do
        if B[x] = a then
            y ← y + 1
        x ← x + 1
    return y
```

Algorithm 2: Computing the starting position for a given suffix array index

Input: Suffix array index k. Suffix array sampling rate r; B , C and SSA array are assumed to be known
Output: Starting position corresponding to k

```
Function CalcStart(k) begin
    i ← 0
    while k mod r ≠ 0 do
        k ← C[B[k]] + Occ(B[k], k − 1)
    return SSA[k]
```

Algorithm 1 is known as backward search as it starts from the last symbol of W and then builds the string in the backward direction. Each iteration of the `while` loop enlarges the string by one symbol and may be called as a *search step*. Each search step returns the suffix array interval of the enlarged string. If $I_l \leq I_u$, the enlarged string exits in T otherwise not. Algorithm 1 is used to find the suffix array interval of a fixed length seed. To find the suffix array interval of all the fixed length seeds in a DNA read, the `BACKWARDSEARCH` function is called with W set equal to the seed sequence. To find the suffix array interval of nov-SMEM, uncomment line 12. To find the first nov-SMEM, call the `BACKWARDSEARCH` with $W = P$, where P is the DNA read sequence. The function will return $match\_len$ along with the suffix array interval. If $match\_len = |P|$, we are done, otherwise call the function again to find the second nov-SMEM with $W = P_1$ where $P_1 = P[match\_len, |P|]$. Similarly, if $match\_len = |P_1|$, the algorithm completes successfully, otherwise the function is called again to find the third nov-SMEM with $W = P_2$ where $P_2 = P_1[match\_len, |P|]$, and so on. The algorithm to compute the suffix array interval of all-SMEMs is given in [12].

***Step 2**—Computing start position:* Once the suffix array intervals of the seeds are computed, the suffix array can be used to find the starting position of a seed (if present). Usually, to reduce memory a sampled suffix array SSA is used. A SSA with a sampling rate of r is the set $\{SA(k) : k \text{ is divisible by } r\}$. Algorithm 2 computes the starting position of a seed with SSA for a given suffix array index value. Each suffix array index value from I_l to I_u corresponds to one occurrence of the seed. Hence, the `CALCSTART` function is called for $k = I_l, \ldots I_u$.

The advantage of using FM-index is its memory efficiency. The complete FM-index for the the human reference genome occupies only 1.5 Gbytes of memory. The time required to compute a seed of length n is *O(n + mr)* where $m = I_u - I_l + 1$ i.e. the number of occurrences of the seed. In practice, computing seeds with FM-index consume a lot of time due to pseudo-random accesses to the large B array which is nearly 1 GB. Such a large array cannot reside in the cache. As shown in Algorithm 1 during every search step the algorithm accesses the B array. Similarly B array is also accessed in every iteration of the loop in Algorithm 2. In [16] the memory access patterns of B array are studied. The study shows that these accesses are quite random resulting in a large number of data cache and data TLB misses due to poor temporal and spatial locality of the accesses. These large D-cache and D-TLB misses cause the algorithm to almost always be waiting for the memory, substantially slowing down the algorithm.

2) Hash table index: Fast computation of fixed length seeds can be performed using a hash table. Hash table stores the starting position of n-mers of the reference genome, where n is the length of the seed. To find the starting position of a fixed length seed just index the hash table with the seed sequence. Hence, they require *O(1)* time to compute a seed. Hash tables are sensitive to the value of n and the sampling frequency s of the genome. Sampling frequency is the distance (in no. of bases) between the starting position of two consecutive n-mers of the genome. For $n > 15$ the hash table size becomes excessively large. Novoalign has a hash table index that occupies 17 GB of RAM with $n = 15$ and $s = 3$ for the human genome. Hence, a hash table index, although fast, is memory demanding. The index has to be rebuilt if the seed length is changed. Hash tables cannot be directly used to compute maximal exact matches. To find maximal exact matching seeds with hash table index, first find the start position of a substring of the read with a hash table and then enlarge it on both sides by a direct comparison between the read and the reference genome. A similar approach has been adopted by the HPG DNA read aligner [17].

IV. Extension techniques

Flanking bases of the reference genome around the seed are fetched to perform the extension step. Three types of seed extension techniques are used in modern DNA read aligners: (1) Global alignment (Needleman-Wunsch algorithm), (2) Local alignment (Smith-Waterman algorithm), and (3) BLAST-like seed extension. All these three techniques are implemented using dynamic programming with affine gap penalties. Cushaw2 performs local alignment, Bowtie2 allows the user to choose between local and global alignment, while BWA-MEM performs BLAST-like seed extension.

A. Global and local alignment

In global and local alignment, the bases around the seed in the reference genome are fetched to form a target sequence that contains the surrounding bases as well as the seed. In global alignment the goal is to find the highest scoring alignment of the full read against the target sequence. In practice the target sequence is longer than the read. Therefore, a semi-global alignment is performed in which gaps on both ends of the read are ignored. In local alignment the goal is only to achieve the highest scoring alignment and thus the resulting alignment may not contain the full read sequence. Global and local alignment algorithm are explained in detail in textbooks. Readers may refer to [18] for more in depth coverage of local and global alignment algorithms.

B. BLAST-like seed extension

BLAST-like seed extension is a fast extension technique that is performed in two steps by calling Algorithm 3 twice, which shows the BLAST-like seed extension algorithm. First step: *seq1 = read bases on the left side of the seed, seq2 = reference bases on the left side of the seed, start_score = seed score*. Second step: *seq1 = read bases on the right side of the seed, seq2 = reference bases on the right side of the seed, start_score = seed score + alignment score of first step*. The BLAST-like seed extension algorithm is similar to local alignment with the following differences:

1. Non-zero start score.
2. A standard local alignment algorithm computes all the

local alignments between two sequences. BLAST-like seed extension is faster as it only computes one local alignment that must contain the seed as a substring. The pseudo-code framed in the first box (i.e., lines 25-26) in Algorithm 3 ensures that this requirement is met. Further speedup is achieved due to the pseudo-code framed in the second box (i.e., lines 31-44) which prunes the Dynamic Programming (DP) matrix entries that cannot result in a final alignment containing the seed. Hence, BLAST-like seed extension does not compute all the the entries of the DP matrix making it faster than local and global alignment techniques.

3. The starting positions of the alignment are always known, so no traceback is required, thereby reducing run time.

C. Optimized seed extension

The dynamic programming based extension stage is compute bound. Therefore, to reduce the total DNA read alignment execution time, extension schemes can be optimized. Striped Smith-Waterman (SSW) is a SIMD optimization of local alignment [19]. The implementation of SSW is also available in the form of a C/C++ library [20]. SIMD optimized DP allows concurrent computation of many DP matrix cells. Another optimization is banded DP, which limits the number of DP matrix cell to be calculated to a narrow band along the main diagonal [21]. Banded DP works well in situations where the two sequences to be aligned are *homologous* as the case of DNA read alignment.

V. GASE Generic Aligner

For the comparison of different seeding and extension techniques, we built GASE (Generic Aligner for *Seed-and-Extend*) that can be used with different seeding and extension techniques. The idea is to use GASE to measure the corresponding alignment accuracy and total execution time for different combinations of seeding and extension. GASE is a minimalistic aligners that mainly depends on the seeding and extension technique being used to identify the read alignment, with little added heuristics coded in the aligner. This results in an aligner with an alignment accuracy that is mainly determined by the seeding and extension technique being used. The different components of our read aligner are outlined below.

1) Index: The FM-index used in our aligner is same as the one generated in BWA-MEM. BWA-MEM builds the FM-Index of $T \oplus \overline{T}$, where T is the reference genome string , $\overline{T}$ is Watson-Crick reverse complement of T and $\oplus$ is the string concatenation operator. The advantages of such kind of index are: 1) Apart from backward search, shown in Algorithm 1, where the string is enlarged from right to left in the reverse direction, a *forward search* is also possible in which the string can also be enlarged from left to right in forward direction 2) A read P is only aligned against $T \oplus \overline{T}$, rather than aligning P and its Watson-Crick reverse complement $\overline{P}$ against T separately. This roughly doubles the speed of the aligner at the cost of memory.

Algorithm 3: BLAST-like seed extension

Input: The two sequences to be aligned $seq1$, $seq2$ and the start score start_score. Penalty of gap open $gapo$ and gap extension $gape$ are assumed to be known values

Output: Maximum score $score$ and its position of achievement on read sequence $read\_end$ and on the reference ref_end

1 **Function** BLASTSEEDEXTENSION($seq1, seq2, start\_score$) **begin**
2 Initialize H, E and F arrays of size of $(|seq1|+1) * (|seq2|+1)$ containing zeros
3 $H[0][0] \leftarrow start\_score$
4 $H[0][1] \leftarrow max\{start\_score - gapo - gape, 0\}$
5 **for** $j \leftarrow 2$ **to** $|seq1|$ **do**
6 $H[0][j] \leftarrow max\{H[0][j-1] - gape, 0\}$
7 $H[1][0] \leftarrow max\{start\_score - gapo - gape, 0\}$
8 **for** $j \leftarrow 2$ **to** $|seq2|$ **do**
9 $H[j][0] \leftarrow max\{H[j-1][0] - gape, 0\}$
10 $max\_score \leftarrow start\_score$
11 $read\_end \leftarrow \emptyset$
12 $ref\_end \leftarrow \emptyset$
13 $beg \leftarrow 1$
14 $end \leftarrow |seq1|$
15 **for** $i \leftarrow 1$ **to** $|seq2|$ **do**
16 $row\_max \leftarrow 0$
17 $max\_j \leftarrow \emptyset$
18 **for** $j \leftarrow beg$ **to** end **do**
19 $E[i][j] \leftarrow max\{H[i-1][j] - gapo - gape, , E[i-1][j] - gape, 0\}$
20 $F[i][j] \leftarrow max\{H[i][j-1] - gapo - gape, , F[i][j-1] - gape, 0\}$
21 $H[i][j] \leftarrow max\{H[i-1][j-1] + S(seq1[j-1], seq2[j-1]), E[i][j], F[i][j], 0\}$
22 **if** $H[i][j] > row\_max$ **then**
23 $row\_max = H[i][j]$
24 $max\_j = j$
25 **if** $row\_max = 0$ **then**
26 **return** $\{max\_score, read\_end, ref\_end\}$
27 **if** $row\_max > max\_score$ **then**
28 $max\_score = row\_max$
29 $read\_end = max\_j$
30 $ref\_end = i$
31 $j \leftarrow beg$
32 **while** $j < end$ **do**
33 **if** $H[i-1][j-1] = 0$ *and* $H[i-1][j] = 0$ *and* $E[i-1][j] = 0$ **then**
34 $j \leftarrow j + 1$
35 **else**
36 break
37 $beg \leftarrow j$
38 $j \leftarrow end$
39 **while** $j >= beg$ **do**
40 **if** $H[i-1][j-1] = 0$ *and* $H[i-1][j] = 0$ *and* $E[i-1][j] = 0$ **then**
41 $j \leftarrow j - 1$
42 **else**
43 break
44 $end \leftarrow j$
45 **return** $\{max\_score, read\_end, ref\_end\}$

2) Seeding: Seeds are computed depending upon the seeding technique under test. We have compared four seeding methodologies used in contemporary read aligners. 1) Fixed length seeds without mismatch: we varied the seed length from 15 to 50 in steps of 5 2) Fixed length seeds with at most one

mismatch allowed: we varied the seed length from 15 to 90 in steps of 5 3) all-SMEMs: we varied the minimum required seed length from 15 to 50 in steps of 5 4) nov-SMEMs: we varied the minimum required seed length from 15 to 50 in steps of 5. For all the above seeds, the seed interval is varied as 1, 5, 10, 15, ..., *seed length*. If a seed is located at more than 500 positions in the reference genome, it is not extended.

3) Chaining: The seeds which lie nearby on the reference genome are chained together. The chains are sorted in descending order on the basis of the weight of the chain. The weight of a chain is the number of reference genome bases covered by the seeds in a chain. A chain is filtered out if it is overlapping with the next higher weight chain by more than 50%.

4) Extension: The seeds in a chain are sorted on the basis of their length. The longest seeds is extended first using one of the three extension techniques being studied. The next seed in the sorted list of seeds is then extended if it is not already covered in the extension of the previous seed and so on. The process is repeated for all the chains. Three different extension techniques have been tested: global alignment, local alignment and BLAST-like seed extension. The output is written in SAM (Sequence Alignment/Map) format [22].

VI. Experimental Results

We measured the error in alignment as:

$$\text{error} = \frac{\text{No. of incorrectly mapped reads}}{\text{No. of mapped reads}}$$

We tested all the 12 possible combinations of seeding and extension techniques. A read aligned within ± 20 base pairs of the true position is considered correct. For each combination of seeding and extension technique, the seed length and seed interval (if applicable) is varied over a range discussed in Section V-2. Only those seed settings have been considered in the comparison in which the number of mapped reads $\geq$ 99.5% of the total number of reads.

A. Input data set

5 Mega bases of the chromosome 21 of human genome (UCSC hg19) are used as a reference. 1 million single ended reads were generated using Wgsim read simulator [23]. Ten reads are aligned in parallel by running ten threads on Intel Xeon E5-2670 2.5 GHz processor. The reads have a mutation rate of 0.4% where 25% of these mutations are indels. 70% of the indels are extended (length greater than 1). This mutation rate in the simulated reads represents the upper limit in human genome variation [24]. Similarly this percentage of indels and their extension rate in the simulated reads correspond to observed values in the human genome [25]. The reads have 2% sequencing errors as well.

B. Selecting seeding parameters

Table II shows the results of measuring the mapping error and the total execution time of DNA read aligner for different seeding and extension techniques with varying read lengths. The read length is specified in base pairs (bp). As described in Section V-2 a number of parameters of the seed are varied over a wide range. Each seed setting results in a different mapping error and execution time. The values in Table II represent a tradeoff between error and time. For each setting we measured the corresponding error and time. If the difference in error between the *most accurate* seed setting and another seed setting is at most 9 incorrectly mapped reads and achieving at least 30% faster execution time than the most accurate seed setting, then the other seed setting is selected.

C. Comparison of seeding techniques

Table II shows that fixed length seeds with no mismatch are not a good choice in any case. They result in more error in the mapping and larger total execution time with all kinds of extension techniques as compared to SMEM seeds and fixed length seed with at most 1 mismatch for all read lengths. In some cases of fixed length seeds with no mismatch, we have not tested some smaller seed lengths due to orders of magnitude higher total execution time as compared to other seeding techniques, and hence useless to consider. With BLAST-like seed extension, all-SMEM is the best approach due to its higher accuracy and lower execution time. For example with 600 bp read length, it is 1.34, 6 and 2 times more accurate than nov-SMEM, fixed length (0 mismatch) and fixed length (at most 1 mismatch) seeds, respectively. Similarly for 600 bp read length, the execution time is 1.37 and 2 times less than fixed length (0 mismatch) and fixed length (at most 1 mismatch) seeds, respectively, and comparable with nov-SMEM. For local alignment all-SMEM, nov-SMEM and fixed length seeds (at most 1 mismatch) have comparable mapping error. For global alignment all-SMEM is the most accurate. With local alignment all-SMEM is faster than nov-SMEM and fixed length seeds (at most 1 mismatch). For global alignment nov-SMEM is the fastest.

D. Comparison of extension techniques

Table II shows that the local alignment is the most accurate for all kinds of seeding techniques. The local alignment becomes more accurate as compared to the global and BLAST-like seed extension techniques with increasing read lengths. For 600 bp read length it is 3.6 times and 1.6 times more accurate than BLAST-like seed extension and global alignment, respectively. The results also show that BLAST-like seed extension should only be used with all-SMEM as its accuracy drops significantly as compared to local and global alignment with other three seeding techniques (i.e. nov-SMEM, fixed length with no mismatch and fixed length with at most 1 mismatch). Global alignment is less accurate than local alignment but more accurate than BLAST-like seed extension for all kinds of seeding techniques. With regard to speed of the unoptimized techniques, BLAST-like seed extension is the fastest and it becomes faster with increasing read lengths as compared to other two extension techniques. With all-SMEM seeding it is 2.2 to 3.4 times faster than unoptimized local and global alignment for 600 bp read length. Although global alignment is more accurate than BLAST-like seed

TABLE II. Mapping error and total execution time of our DNA read aligner GASE with different combinations of seeding and extension techniques. The values before the slash (/) in the time column represent the execution time with unoptimized extension stage, whereas the values after slash are obtained with optimized extension stage.

	Read len.	all-SMEM		nov-SMEM		fix-len. (0-mismatch)		fix-len. (1-mismatch)	
		error	time (sec.)	error	time (sec.)	error	time (sec.)	error	time (sec.)
global	150	1.24e-3	93/27	1.3e-3	74/23	1.39e-3	1704/340	1.21e-3	209/120
	250	3.35e-4	173/60	3.81e-4	151/56	4.66e-4	2949/405	3.73e-4	185/96
	400	8.3e-5	370/129	9.4e-5	427/130	2.15e-4	11637/1177	1.2e-4	710/518
	600	4e-5	890/273	4.9e-5	742/255	2.7e-4	43021/3152	1.07e-4	1194/413
local	150	1.22e-3	59/26	1.3e-3	74/24	1.262e-3	1138/144	1.21e-3	183/113
	250	3.28e-4	180/61	3.68e-4	159/58	3.68e-4	554/127	3.45e-4	284/99
	400	6.8e-5	354/126	7e-5	507/128	6.9e-5	5864/520	5.6e-5	419/200
	600	2.5e-5	805/260	3e-5	1241/272	5.5e-5	3866/428	3.2e-5	824/392
BLAST-like seed extension	150	1.25e-3	28/27	1.37e-3	24/22	1.80e-3	120/119	1.34e-3	42/43
	250	4.16e-4	64/59	4.8e-4	60/58	1.05e-3	65/63	4.77e-4	262/283
	400	1.36e-4	148/134	1.97e-4	136/125	6.91e-4	193/179	2.57e-4	216/219
	600	9.4e-5	305/260	1.26e-4	276/240	5.05e-4	364/327	1.88e-4	419/390

extension, it is not suitable for aligning *split reads* (also known as chimeric reads). Split reads are generated due to large structural variations in the genome. The speed of optimized Blast-like and local alignment techniques is comparable.

E. Selecting the best seeding-extension combination

From Table II we can conclude that:

- Blast-like seed extension is the fastest and should at least be combined with all-SMEM
- Local alignment is the most accurate and its accuracy is nearly the same with all-SMEM, nov-SMEM and fixed length (at most 1 mismatch) seeds

To come up with the best seeding-extension combination we must first decide the best seeding technique to be used with local alignment. To do this we will compare the execution times of optimized DNA read aligners. The speed optimization performed here does not affect the mapping error. These optimization speed up the extension stage of the DNA read aligner with techniques described in Section IV-C. We only focus on optimizing the extension rather than the seeding due to the memory bound nature of seed computation using FM-index, which makes it hard to optimize/accelerate. Hash table index is a fast seeding mechanism (as compared to the FM-index) for finding fixed length seeds, but it cannot be used in our case as the values reported for fixed length seeds with at most 1 mismatch in Table II are mostly for very long seeds (30 or above). Shorter fixed length seeds have much higher error as compared to the values given in Table II. Building hash table for seeds longer than 20 bp is impractical. We can select the best seeding technique to be used with local alignment by measuring the execution time of the DNA read aligner with optimized local alignment. The local alignment is SIMD optimized using SSW. It is implemented with Intel SSE2 instruction set which has 128-bit SIMD registers. A signed two-byte integer is used to store the score value. This allows the concurrent computation of 8 DP matrix cells.

Figure 3 compares the execution of local alignment with three seeding techniques (all-SMEM, nov-SMEM and fixed length seeds with at most 1 mismatch) before and after applying the SIMD optimization. The input data set is the same as the one used in Table II. The mapping error for each seeding technique remains nearly the same as reported in Table II and hence, is comparable to all techniques. Figure 3 shows that after SIMD optimization nov-SMEM-local becomes slightly faster than fixed-length(1)-local and has nearly same execution time as all-SMEM-local. The figure shows the significant reduction in execution time of local alignment with nov-SMEM. Its execution time scales down by 4.57x for 600 bp read length. For the range of DNA read lengths shown, the reduction in execution time of nov-SMEM is 2.74x up to 4.57x, for all-SMEM it is 2.32x up to 3.1x and for fixed length seeds with at most 1 mismatch it is 1.62x up to 2.86x. Therefore, the combination of nov-SMEM with local alignment achieves the highest reduction in execution time with minimal reduction in accuracy. Although, nov-SMEM generate less seeds as compared to all-SMEM and fixed length seeds, however for longer reads the amount of seeds generated by nov-SMEM is sufficient for accurate mapping of the read. The high reduction in the execution time of nov-SMEM-local for longer reads shows that with future DNA reads nov-SMEM-local has high potential for acceleration/optimization without sacrificing accuracy.

Now that we have seen that nov-SMEM is a better seeding approach for local alignment, we will now compare the execution time of SIMD optimized nov-SMEM-local and banded all-SMEM-BLAST-ext DNA read aligners (which is the fastest combination using the Blast-like extension algorithm) to come up with the best seeding-extension combination. Our banded implementation of BLAST-like seed extension is the same as done in BWA-MEM. Figure 4 compares the execution time of the SIMD optimized nov-SMEM-local and banded all-SMEM-BLAST-ext. The input data set is the same as the one used in Table II. The optimization has not increased the mapping error of both schemes and therefore, nov-SMEM-local remains more accurate than all-SMEM-BLAST-ext for all read lengths (with 600 bp read length nov-SMEM-local

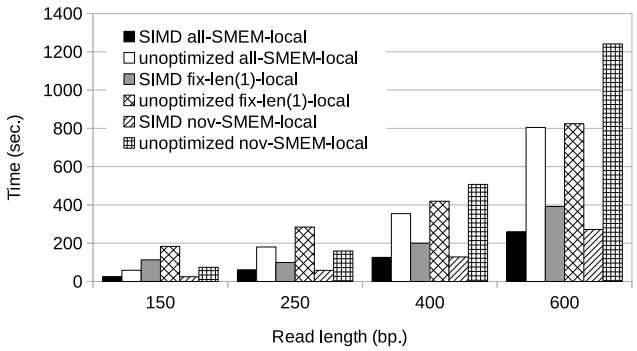

Fig. 3. Comparison of execution time of local alignment with all-SMEM, nov-SMEM and fixed length seeds (at most 1 mismatch) before and after SIMD optimization

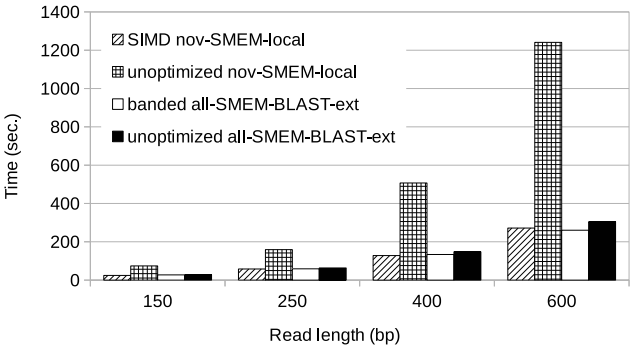

Fig. 4. Comparison of execution time of BLAST-like seed extension with all-SMEM and nov-SMEM-local before and after optimization

is 3.1x more accurate than all-SMEM-BLAST-ext). Figure 4 shows that the banded all-SMEM-BLAST-ext seed extension was not able to gain much speed as compared to the unoptimized all-SMEM-BLAST-ext of Table II, whereas SIMD nov-SMEM-local shows good speed up has nearly same execution time as banded all-SMEM-BLAST-ext. Therefore, we can conclude that SIMD optimized nov-SMEM-local seeding-extension combination have performed best in this comparison due to its high accuracy, and more potential for optimization/acceleration for future DNA reads.

VII. Conclusion

In this paper we compared different seeding and extension techniques used in modern DNA read aligners. The compared seeding techniques were maximal exact matching seeds and fixed length seeds. Three seed extension techniques were compared: global alignment, local alignment and BLAST-like seed extension. For the purpose of the comparison we built an open source generic seed-and-extend DNA read aligner called GASE and then measured the accuracy and execution time for all the possible seeding and extension techniques. Our results showed that fixed length seeds (0-mismatch) are not a good seeding choice with any type of extension algorithm, while all-SMEM is the best seeding approach with BLAST-like seed extension. Local alignment is more accurate than BLAST-like seed extension, especially for longer reads. all-SMEM, nov-SMEM and fixed length seeds (at most 1 mismatch) have comparable accuracies with local alignment. Overall, SIMD optimized nov-SMEM-local seeding-extension combination has performed best in this comparison due to its high accuracy and more potential for optimization/acceleration for future DNA reads.

References

[1] Wetterstrand KA., "DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP)," Available at: www.genome.gov/sequencingcosts, Accessed [30th September, 2015].

[2] S. F. Altschul *et al.*, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403 – 410, 1990.

[3] "NovoAlign," http://www.novocraft.com/products/novoalign/.

[4] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," *arXiv [q-bio.GN]*, May 2013. [Online]. Available: http://arxiv.org/abs/1303.3997

[5] B. Langmead and S. S., "Fast gapped-read alignment with bowtie 2," *Nature Methods*, vol. 9, pp. 357–359, 2012.

[6] Y. Liu and B. Schmidt, "Long read alignment based on maximal exact match seeds," *Bioinformatics*, vol. 28, no. 18, pp. i318–i324, 2012.

[7] "GASE generic aligner," https://github.com/nahmedraja/GASE.

[8] J. Shang *et al.*, "Evaluation and comparison of multiple aligners for next-generation sequencing data analysis," *BioMed Research International*, 2004.

[9] "Genome Comparison and Analytic Testing," http://www.bioplanet.com/gcat.

[10] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," *Briefings in Bioinformatics*, vol. 11, no. 5, pp. 473–483, 2010.

[11] I. Mandoiu and A. Zelikovsky, *Bioinformatics Algorithms: Techniques and Applications*. John Wiley and Sons, 2008, ch. 6, pp. 117–142.

[12] H. Li, "Exploring single-sample SNP and indel calling with whole-genome de novo assembly," *Bioinformatics*, vol. 28, no. 14, pp. 1838–1844, Jul 2012.

[13] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, ser. FOCS '00, 2000, pp. 390–398.

[14] U. Manber and G. Myers, "Suffix arrays: A new method for on-line string searches," *SIAM Journal on Computing*, vol. 22, no. 5, pp. 935–948, 1993.

[15] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch, "Replacing suffix trees with enhanced suffix arrays," *Journal of Discrete Algorithms*, vol. 2, no. 1, pp. 53 – 86, 2004.

[16] J. Zhang *et al.*, "Optimizing burrows-wheeler transform-based sequence alignment on multicore architectures," in *CCGrid*, Delft, Netherlands, May 2013.

[17] J. Trraga *et al.*, "Acceleration of short and long dna read mapping without loss of accuracy using suffix array," *Bioinformatics*, 2014.

[18] W. K. Sung, *Algorithms in Bioinformatics: A Practical Introduction*. CRC Press, 2009, ch. 2, pp. 29–56.

[19] M. Farrar, "Striped smithwaterman speeds database searches six times over other SIMD implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2007.

[20] M. Zhao *et al.*, "SSW library: An SIMD smith-waterman c/c++ library for use in genomic applications," *PLoS ONE*, vol. 8, 12 2013.

[21] K. Chao, W. R. Pearson, and W. Miller, "Aligning two sequences within a specified diagonal band," *Computer applications in the biosciences : CABIOS*, vol. 8, no. 5, pp. 481–487, 1992.

[22] "SAM format specification," https://samtools.github.io/hts-specs/SAMv1.pdf.

[23] "Wgsim," https://github.com/lh3/wgsim.

[24] S. Tishkoff and K. K. Kidd, "Implications of biogeography of human populations for 'race' and medicine," *Nature Genetics*, vol. 36, no. 11s, pp. S21 – S27, 2004.

[25] R. Mills *et al.*, "An initial map of insertion and deletion (indel) variation in the human genome," *Genome Research*, vol. 16, no. 5, pp. 1182–90, 2006.