Faster short-read mapping with strobemer seeds constructed from syncmers

Kristoffer Sahlin[1,✉]

[1]Department of Mathematics, Science for Life Laboratory, Stockholm University, 106 91, Stockholm, Sweden.

Alignment of short reads to genomes is a fundamental computational step used in many bioinformatic analyses. It is therefore desirable to perform such computation as fast as possible. Most alignment algorithms consider a seed-and-extend approach. Several popular programs perform the seeding step based on the Burrows-Wheeler Transform with a low memory footprint, but they are relatively slow compared to more recent approaches that use minimizer-based seeding-and-chaining strategies. Recently, syncmers and strobemers were proposed for sequence comparison. Both methods were designed for improved conservation of matches between sequences under mutations. Syncmers is a k-mer subsampling method proposed as an alternative to minimizers, while strobemers is a method for linking gapped sequences and was proposed as an alternative to k-mers.

The main contribution in this work is a new seeding approach that combines syncmers and strobemers. We use a strobemer method (randstrobes) to link together syncmers, instead of considering k-mers over the entire sequence as strobemers were first described. Our method allows us to create longer seeds that preserve mapping accuracy and reduce the number of hits, allowing faster mapping. Our second contribution is an implementation of the seeding approach in a short-read aligner, strobealign. Strobealign supports both single-end and paired-end read alignment modes, and aligns single-end reads about 2-3 times faster than minimap2 and 12-15 times faster than BWA and Bowtie2 with similar accuracy. In paired-end mode, strobealign aligns reads of lengths 200nt and above more than 3 times as fast as minimap2 with near-identical accuracy and about 10 times faster than BWA and Bowtie2 with a 0.1-0.2% loss in accuracy. Our contribution is algorithmic and requires no hardware architecture or system-specific instructions. We believe that our seeding approach can be applied to other mapping applications such as long-read mapping and clustering. Strobealign is available at https://github.com/ksahlin/strobealign.

strobemers | syncmers | seed-and-extend | read mapping | Illumina short-reads | alignment

Correspondence: ***ksahlin@math.su.se***

Introduction

Aligning Illumina sequencing reads to a reference genome or transcriptome is the first step in many analyses pipelines. Due to the fundamental role of short-read alignment in bioinformatics, many methods have been proposed. BWA (1, 2), Bowtie (3), and Bowtie2 (4), which use the Burrows-Wheeler transform (5), has received widespread use through their favourable memory and runtime trade-off. These aligners have for several years been the two dominating genomic short-read aligners. Several alternatives approaches have been proposed, such as random permutations aligner (6), rNA (7), SNAP (8), and subread (9) that has alternative strengths showing improved accuracy or decreased runtime in specific use cases. There are also several major contributions in data structures and algorithms in splice alignment of RNA-seq data (10, 11) or pseudo-alignment methods (12, 13) that are incredibly fast by not explicitly performing an exact alignment.

A pioneering approach was the use of minimizers (14, 15) as seeds in alignment and overlap detection algorithms (16, 17). Minimap2 was initially described for long-read alignment but showed to produce alignment of short reads at similar accuracy but several times faster than the popularly used BWA and Bowtie2. Minimap2 uses minimizers to produce seeds and then employs collinear chaining of seeds to produce candidate locations for the alignment step. Along the same trajectory, Mashmap (18) and Winnowmap (19) was designed for long-reads and made algorithmic contributions to minimizer-based alignment by considering and adjusting the densities and probabilities around the sampling of minimizers.

A computational hurdle in sequence alignment is the length of the seeds, which inform the aligner of candidate mapping locations. To achieve good alignment-accuracy, alignment algorithms often need to use a shorter seed length than what is considered to give unique matches in the genome of interest. Therefore, the seeds may produce many candidate regions that need to be evaluated (and filtered) based on some score before the alignment stage. Alignment methods therefore usually described to employ a seed-filter-extend approach. Accel-Align (20) instead proposes an alternative idea to the filtering step by computing a much cheaper hamming distance of an embedded representation of the read. Interesting candidate sites should have low embedded Hamming distance and are sent for alignment. Accel-Align was shown to outperform minimap2 in terms of speed.

This work instead focus on the seeding step. We describe a novel seeding technique that allows us to use much longer seed lengths than minimizer based approaches such as minimap2. Longer seed lengths reduce the number of candidate sides for reads which allows much faster mapping and alignment. Our contribution is based on two recent advancements in the area of sequence comparison; syncmers (21) and strobemers (22). Both syncmers and strobemers were proposed to increase sequence conservation under mutated sequences. Here, sequence conservation is the total amount of sequence covered by matches (under a sampling method)

DRAFT

between two sequences. Syncmers was proposed as an alternative to the minimizer subsampling technique, while strobemers was proposed as a method for producing gapped seeds as an alternative to k-mers and spaced k-mers. Here we show that syncmers and strobemers can be used in combination in what becomes a high-speed indexing method, roughly corresponding to the speed of computing minimizers. Our technique is based on first subsampling k-mers from the reference sequences by computing canonical open syncmers (21), then, producing strobemers (22) formed from linking together syncmers occurring close-by on the reference using the randstrobe method. A consequence is that instead of using a single seed (e.g., k=21 as default in minimap2 for short-read mapping), we show that we can link together two syncmers as a strobemer seed and achieve similar accuracy to using individual minimizers. Using longer seeds reduces the computational burden of evaluating candidate mapping sites. Furthermore, we use parameters such that the syncmers are created identically from the forward and reverse complement strands. Therefore, we only need to compute the syncmers in one direction, while the randstrobes can be computed quickly over the syncmers in both directions.

In genome assembly and error correction, ideas to work in minimizer-space instead of in sequence-space have been proposed through the use of paired minimizers (23, 24) and k consecutive minimizers (k-min-mers) (25) to represent overlaps and assembly graphs. These concepts are related to what was described as minstrobes in (24). Here we instead use the randstrobe method for linking that is more efficient at creating matches than minstrobes (24).

Our main contribution in this work is the novel seeding approach by computing strobemers over syncmers. We demonstrate that this seeding is fast to compute (about 5 minutes to index hg38) and is therefore competitive to, *e.g.*, minimizers in high performance sequence mapping scenarios. Our second contribution is the implementation of our seeding technique in a short-read alignment tool, strobealign. Strobealign supports both a mapping and an alignment mode similarly to minimap2. As for computing seeds and candidate mapping sites, we show that strobealign is 2-6x faster than minimap2 in mapping mode across our simulated and biological datasets for reads of 150nt or longer. In alignment mode, we show that strobealign aligns simulated single-end Illumina reads 2-3 times faster than minimap2 and 12-15 times faster than BWA-MEM and Bowtie2 with comparable accuracy at various indel and substitution rates. In paired-end mode, strobealign aligns simulated paired-end reads of lengths 200-300nt more than 3 times as fast as minimap2 with near identical accuracy, and about 10 times faster than BWA and Bowtie2 with a 0.1-0.2% loss in accuracy. In addition, strobealign is faster and more accurate than the recently proposed Accel-Align. We also show on biological data that strobealign is significantly faster than all tools except Accel-Align, but strobealign produces alignments much more consistent with BWA-MEM and Bowtie2. Strobealign also supports multi-threading and parallelizes as well as BWA-MEM and minimap2 and better than Bowtie2 and Accel-Align. Finally, since our contribution is in the seeding step, and therefore is orthogonal to the contribution of Accel-Align (the filtering step). Potentially, the two approaches could be applied together to improve alignment speed further.

Methods

Overview. The main idea of the seeding approach presented here is to first compute open syncmers (21) from the reference sequences, then link the syncmers together using the randstrobe method (22) with two strobes. The study introducing strobemers (22) described strobemers as linking together strobes in 'sequence-space', *i.e.*, over the set of all k-mers. Since syncmers represent a subset of k-mers from the original sequence, computing randstrobes over this subset of strings is very fast; it suffice to compare a smaller set of syncmers to produce the next strobe, while still having a similar range on the upper and lower window bounds on the original sequence. By using randstrobe linking of syncmers, which is more efficient at sequence matching than simply joining adjacent minimizers (22), we can afford to use longer seeds compared to other approaches while having a broadly similar accuracy. Having a larger seed length allows for more unique matches and makes downstream candidate site selection for alignment faster. The general idea is as simple as that. In the following sections, we describe in detail the idea and our short-read alignment implementation, strobealign.

Definitions.

General. We use i to index the position in a string s and let $s[i,k]$ denote a *k-mer* substring at position i in s covering the k positions $i,\ldots,i+k-1$ in s. We will consider 0-indexed strings. We let $|\cdot|$ denote the length of strings. We will denote a match as m and use $m.q_s, m.q_e, m.r_s, m.r_e, m.o$ to denote the read start and end positions, the reference start and end positions, and the orientation of the match, respectively. Finally, we use 2-bit encoding to store nucleotides with 00, 01, 10, and 11 representing A, C, G, and T. With this bit encoding in mind, we let v refer to the integer value of a string. For example, $v(TCA) = 52$, since its binary form is 110100.

Syncmers. Syncmers was described in (21) and consists of three parameters (k,s,t) where k is the length of syncmers. We will here consider open syncmers. The open-syncmer method compares the $k-s+1$ consecutive s-mers within a k-mer and selects the k-mer as a syncmer if the smallest s-mer occurs at position $t \in [0, k-s+1]$ within the k-mer. With the smallest s-mer, we mean the hash value that the s-mer produces. Similarly to what is commonly performed in k-mer applications, we use a canonical representation of syncmers. A canonical representation means that the lexicographically smallest syncmer out of its forward and reverse-complement sequence is stored.

Strobemers. Strobemers was described in (22) and consists of several shorter k-mers, referred to as *strobes*. We use the randstrobe method (22) to join the strobes. Four parameters (n, m, w_{min}, w_{max}) are used to define a randstrobe where n

DRAFT

is the number of strobes, m is the length of the strobes, and w_{min} and w_{max} is the upper and lower downstream window on the sequence where consecutive strobes are sampled. We will here consider randstrobes with $n = 2$, *i.e.*, consists of two strobes. The randstrobe method joins together a strobe s_1 with a strobe $s_2 = s'$ if s' minimizes $\big(v(s_1) + v(s')\big)\&q$, where q is a bitmask consisting of ones at the 16 lowest-order bits and remaining zeroes and s' occurs in the interval $[w_{min}, w_{max}]$ downstream of s_1.

Modifications to strobemers. Originally, strobemers were described to be produced over the set of all k-mers in a sequence. In this work, we will compute the strobemers over syncmers. This means that s_1, s_2, and s' are syncmers, and we will let $[w_{min}, w_{max}]$ refer to the lower and upper number of syncmers downstream from s_1 where we will sample s_2 from. A second modification to the strobemers as described in (22) is that we store the strobemer hash value from two strobes s_1 and s_2 as $H(s_1, s_2) = v(s_1)/2 + v(s_2)/2$. The hash function H is symmetric ($h(v(s_1), v(s_2)) = h(v(s_2), v(s_1))$) and together with canonical syncmers it produces the same hash value if the strobemer is created from forward and reverse complement direction. It is stated in (22) that a symmetrical hash function is undesirable for mapping due to unnecessary hash collisions. However, when masking highly repetitive seeds as commonly performed in aligners (17), it turns out that a symmetrical hash function helps to avoid sub-optimal alignments, and we will now describe why.

Assume we would use an asymmetric hash function, such as $v(s_1)/2 + v(s_2)/3$ proposed in (22). Also assume that strobemer seeds (s_1, s_2) and (s_2, s_1) are both found in forward orientation the reference due to, *e.g.*, inversions. In this case, it can happen that only (s_1, s_2) is masked because of its distinct hash value to (s_2, s_1). Now, consider a read in which we extract (s_1, s_2) in forward direction and (s_2, s_1) in reverse complement direction. If the read has an optimal match to forward direction with seed (s_1, s_2) (masked on reference) we would still find the suboptimal match of (s_2, s_1) of the read in reverse complement orientation to the reference. By using a symmetric hash function we guarantee to mask the same strobemers in both directions. We observed that such cases are common on, *e.g.*, chromosome X in the human genome. While we could overcome such scenarios by indexing and masking the reference in both orientations, it introduces a lot of memory overhead when storing the index. The standard procedure in mapping is to index the reference in one direction while computing seeds from reads in both directions. We believe that the trade-off of having more collisions but mask seeds consistently across the two sequence directions is beneficial, but there is room for future investigation on handling such cases.

Indexing. We first construct syncmers from the reference sequences and then link two syncmers together using the randstrobe method as described above. A neat characteristic with syncmers is that the the same syncmers will be created from the forward and reverse complement strand if $k - s + 1$ is odd and $t = \lceil (k - s + 1)/2 \rceil$. Conveniently, it was shown that choosing this value on t produce the maximum spread, and is in consequence the optimal parameter for sequence mapping (26). We compute such canonical open syncmers (using $k = 22$, $s = 18$, $t = 3$ as default values) to produce a subsampling of roughly 20% of the k-mers sampled, which is similar to $w = 10$ in the minimizer sampling method. As for forming the strobemers, we compute the second strobe from a window of $[w_{min}, w_{max}]$ downstream syncmers to the first strobe, where we pick $w_{min} = k/(k - s + 1) + 2$ in order for the syncmers to be unlikely to overlap, and $w_{max} = k/(k - s + 1) + 10$ is chosen based on simulation. For the parameters $k = 22$, $s = 18$, $t = 3$ used here, strobealign samples the second syncmer in an expected downstream window of $[30, 70]$ nucleotides.

We store tuples (h_k, r_{id}, r_s, r_e) in a flat vector V where h is the hash value of the strobemer k (64 bit integer), r_{id} is the reference accession coded as (32 bit integer), r_s is the start of the first strobe (32 bit integer), r_e is the start of the second strobe (32 bit integer). We sort V by hash values h and construct a hash table with key h_k pointing to tuple values (p_k, c_k) where p_k is the position of strobemer k in V and c_k is the count of k in the reference. By lookup of h_k, we know which segment in V to iterate over to find query matches. This type of index representation has been used previously (17, 20), and was suggested to us by Heng Li (27). Finally, similarly to minimap2, we mask (ignore) a top fraction of repetitive strobemers in the reference. This value is a parameter to strobealign and is by default set to 0.0002, similarly to minimap2.

Finding candidate mapping sites. As for the indexing of the reference, canonical open syncmers are first computed from the read. Since the created syncmers are canonical, we can simply compute the forward and reverse complemented randstrobes by iterating over the syncmers and construct randstrobes in forward and reverse order, respectively. Computing randstrobes in forward and reverse orientation gives us a vector of tuples (o, q_s, q_e) representing orientation o (boolean value with 0 representing forward and 1 representing reverse complement), start q_s and end q_e coordinate of the strobemer on the read.

If a strobemer is found in the reference, it will have one or more coordinate-tuples (r_{id}, r_s, r_e) in the reference. We call $m = (o, q_s, q_e, r_{id}, r_s, r_e)$ a *match* between the query and the reference. We then construct *merged matches* $\mathcal{M}$ which are similar but slightly more stringent to Non-overlapping Approximate Matches (NAMs) that are defined in (22). Merged matches are produced as follows. We iterate over all matches in ascending order in the read and join two matches m and m' into a merged match if it holds that (i) $(m.r_{id} == m'.r_{id})\&(m.o == m'.o)$, (ii) $m.q_s < m'.q_s \leq m.q_e$, (iii) $m.r_s < m'.r_s \leq m.r_e$ and (iv) if one of the following holds; $(m.q_s < m'.q_s < m.q_e < m'.q_e) \wedge (m.r_s < m'.r_s < m.r_e < m'.r_e)$ or $(m.q_s < m'.q_s < m'.q_e < m.q_e) \wedge (m.r_s < m'.r_s < m'.r_e < m.r_e)$. In other words, the matches need to (i) come from the same reference and have the same direction, (ii-iii) overlap on both query and reference, and (iv) two

DRAFT

strobemer matches need to have consistent ordering of the four strobes on the reference and the query. Specifically a NAM requires only (i-iii). There are scenarios due to local repeats where, for example, $(m.q_s < m'.q_s < m.q_e < m'.q_e)$ is the order on the query but $(m.r_s < m'.r_s < m'.r_e < m.r_e)$ is the order on the reference invalidating (iv). We consider such cases as separate matches.

If m is the current considered match in the iteration over the matches in a query, we refer to all matches m'' with $m''.q_s < m.q_s \leq m''.q_e$ as *open matches* and $m''.q_e < m.q_s$ as *closed matches*. While iterating over the matches in a read, we keep a vector of currently open *merged matches*, and filter out the closed matches in this vector. In a merged match $\mathcal{M}$ we keep information on how many matches were added, and the position of the first and last strobe on both query and reference, as well as the orientation with respect to the reference genome. When reaching the end of the matches on the query, we close all merged matches. The closed matches are the final merged matches which constitutes the candidate mapping locations.

Computing MAPQ score. After merging matches, each merged match $\mathcal{M}$ consists of a number of matches $|\mathcal{M}|$ and a total span-range of the merged match on both the query $a = \mathcal{M}.q_e - \mathcal{M}.q_s$ and the reference $b = \mathcal{M}.r_e - \mathcal{M}.r_s$. We define the score of $\mathcal{M}$ as $S_{\mathcal{M}} = (\min\{a,b\} - |a-b|) \cdot |\mathcal{M}|$, which acknowledges only the minimum span over the query and reference and penalizes if there is a difference in the span lengths. We compute the MAPQ score similarly to minimap2 but substitute minimap2's chain score to our merged match score. That is, if s_1 and s_2 are the top two scores the MAPQ is computed by

$$MAPQ = 40(1 - S_2/S_1) \cdot min\{1, |\mathcal{M}|/10\} \cdot \log S_1$$

For a fair comparison of accuracy and runtime, we consider the top 20 candidates for alignment and we implement a drop-off score threshold of 0.5 (score to the highest score) which are default parameters for short reads in minimap2.

Single-end Alignment. Merged matches are produced as described earlier and constitutes the candidate mapping regions. For each candidate region (sorted order with respect to the score), we extract segments on the reference defined by coordinates $(\mathcal{M}.r_s - \mathcal{M}.q_e, \mathcal{M}.r_e + (|q| - \mathcal{M}.q_e))$, where $|q|$ is the length of the read. If $|\mathcal{M}.q_e - \mathcal{M}.q_s| = |\mathcal{M}.r_e - \mathcal{M}.r_s|$, we compute the hamming distance between the read and the extracted reference segment. Otherwise, if the distance between between merged match is different on the reference and query due to, *e.g.*, indels, we send the sequences to alignment with ksw2 (28), which is also used in Accel-Align and minimap2. We use a match score of 1, and alignment penalties of 4, 6, 1, for mismatch, gap open, and gap extension, respectively.s Additionally, if the computed hamming distance is larger than $0.05|r|$ where $|r|$ denotes the read length, we perform an additional alignment with ksw as, theoretically, there may be more than one indel within the mapped location that would lead to the same match lengths on the read and the reference.

Rescue mode. A read can have few or zero matches if all the strobemers extracted from the read were masked due to being too abundant. The cutoff of what is considered too abundant, which we denote as A, is controlled with a parameter -f (0.0002), as in minimap2. For example, $A = 41$ in hg38. In case more than 30% or strobemers was masked when finding matches, strobealign enters a rescue mode considering a higher threshold. In the rescue mode, strobealign sorts the seeds according to the abundance on the reference. Then it selects all the seeds below an abundance of R (R is a positive integer parameter with default value 2) and, if this still produce fewer than 5 seeds, it uses 1000 as a hard abundance threshold. If there are still 0 matches, the read is treated as unmapped.

Paired-end Alignment. Similarly to the single-end mapping mode, strobealign computes merged matches for both mates within the read pair and employs an identical rescue mode if too many masked strobemer seeds, as described for the single end mapping. There are however two additional components in the paired end mapping mode. Firstly, strobealign employs a joint scoring of candidate mapping location based on expected insert size (similar to BWA-MEM). Secondly, strobealign can enter a rescue mode even for a read with zero matches, based on the mate's mapping location. We describe the two components below.

For the joint scoring, strobealign first ranks the set of candidate map locations based on the total seed count for the two mates in a read pair then, through alignment, finds the best alignment locations (from alignment score) of the read pair from at most 20 of the locations with highest seed count (similar to single-end alignment mode). Let $|\mathcal{M}_i^1|$ and $|\mathcal{M}_j^2|$ be the number of matches in i-th and j-th merged match for the first and the second mate in the read-pair, respectively. If it holds for some i and j that the two merged matches are on the same chromosome in the correct relative orientation with a mapped distance $< \mu + 10\sigma$, the joint map-location count C_{ij} is simply $C_{ij} = |\mathcal{M}_i^1| + |\mathcal{M}_j^2|$. We also add the individual candidate map location counts obtained as $C_{i-} = |\mathcal{M}_i^1|$ and $C_{-j} = |\mathcal{M}_j^2|$ for the two mates individually.

For the scores in order of highest total seed count first, strobealign performs base level alignment of each mate (as described in the Single-end alignment section). The alignment score S_{ij} of such aligned pair is then computed as

$$S_{ij} = SW_i + SW_j + \log N(d_{ij}, \mu, \sigma)$$

where SW denotes the Smith-Waterman alignment score, and d_{ij} denotes the distance between the mates on the genome. The individually mapped reads (*e.g.*, if on different chromosomes) are given a score

$$S_{ij} = SW_i + SW_j - 10.$$

Since -10 corresponds to more than 4 standard deviations away, this is the score cutoff which prefers the reads to be

DRAFT

mapped individually at their respective locations with the highest SW score.

Mate rescue mode. If one of the mates does not have any merged match, we perform base level alignment of the mate within a genomic segment of $[0, \mu + 5\sigma]$ nucleotides away from the read with a match location.

Implementation details. In addition to the drop-off parameter of 0.5 and considering at most 20 candidate alignments (as in minimap2), we employ two additional optimizations. First, if we encounter a perfect match (no mismatches), we stop and report the alignment even if there are remaining candidates above the drop-off parameter. Second, suppose we have encountered an alignment with Hamming or Edit Distance of 1. In that case, we do not call ksw2 for remaining candidates since a call to ksw2, as we described in the single-end alignment section, implies that the edit distance is at least 1. Stobealign also supports multithreading using openMP. With more than one thread, strobealign will parallelize the alignment step by splitting the read batches of 1 million reads into one batch per thread.

Example parameters and implications. Here we give some numbers on what is expected by our seeding approach for Illumina reads. With a read length of length 200nt and the default parameters in this study of $k = 22, s = 18, t = 3, w_{min} = k/(k-s+1)+2$, and $w_{max} = k/(k-s+1)+10$, roughly $(1/5) * 200 = 40$ syncmers are produced for each read, and roughly 30 strobemers are created in each direction. Naturally, these values are reduced for shorter reads which impacts sensitivity. A read of 100nt will have on average 20 syncmers and only 10 strobemers. We could consider lower w_{max} to produce more strobemers for shorter reads.

Results

We evaluated strobealign (v0.0.3) to four state-of-the-art short-read aligners, BWA-MEM (v0.7.17), Bowtie2 (v2.3.5), minimap2 (v2.22), and Accel-Align (commit 01d18b2). We evaluated the accuracy, runtime, and memory usage of aligners in both single-end and paired-end alignment modes using simulated single-end and paired-end Illumina reads from genomes at different divergence levels from hg38. We denote these datasets SIM1, SIM2, and SIM3 (details on simulations below). For the aligners that also supports a mapping mode (Accel-Align, minimap2, and strobealign) we evaluated both the mapping and alignment modes. As the mapping mode does not invoke the base level alignment step, we can more closely compare the time to find candidate locations. We ran all the tools using 1 thread to investigate alignment speed per thread, as well as with 4, 8, and 16 threads to investigate how performance is affected by parallelization. Finally, we evaluated concordance between aligners on 4 million biological 2*250 paired-end Illumina reads from the mother in the Ashkenazim trio Genome-In-a-Bottle consortium (29) (file accessions D3_S1_L001_R1_001 and D3_S1_L001_R2_001). We denote this dataset as BIO.

Simulated data. We used mason2 (mason_variator) (30) to simulate VCF files with variants to hg38 at various SNP and indel rates, and denote the datasets SIM1, SIM2, and SIM3 for increasing degree of mutation and indel rates to hg38. We denote as SIM1 the genome using default mason_variator parameters (-sv-indel-rate 0.0000001 –snp-rate 0.0001 –small-indel-rate 0.000001 –max-small-indel-size 6). We let SIM2 denote the genome with variations simulated with parameters (-sv-indel-rate 0.000001 –snp-rate 0.001 –small-indel-rate 0.00001 –max-small-indel-size 20) and SIM3 be the genome with variations simulated with parameters (–sv-indel-rate 0.000005 –snp-rate 0.001 –small-indel-rate 0.0001 –max-small-indel-size 50).
For each of the genomes SIM1, SIM2, and SIM3, we used mason_simulator (30) to simulate Illumina single and paired-end reads with read lengths 100,150,200,250, and 300. For the 250 and 300 paired-end read experiments, we had to set the parameter –fragment-mean-size 700 (default is 300) as mason_simulator complained about too small fragment size in the default mode. We then mapped the reads simulated from SIM1, SIM2, and SIM3 to hg38 with the aligners.
We used short read mode (-x sr) in minimap2 and default parameters for the rest of the aligners except for bowtie2 where we had to set maximum fragment length -X 1000 (default 500) for the paired-end 250nt and 300nt experiments that were simulated with a larger fragment size.
We measured the runtime, peak memory usage, accuracy, percentage of aligned reads, and over-aligned reads for each of the tools. We measured alignment runtime after the index has been loaded into memory. We can parse and subtract the time to read or construct the index from the standard output of strobealign, minimap2, and Accel-Align. As described in (20), larger indexes take a longer time to load from disc, although the difference is negligible when aligning a large set of reads. On our data, loading the index took about 22 seconds for minimap2 and about 25 seconds for Accel-Align. For BWA-MEM and Bowtie2, we did not subtract this time as they cannot be parsed from the output, but their indices are smaller and very fast to load (about 6-9 seconds reported in (20)). We measured accuracy as the fraction of correctly aligned reads out of all aligned reads. We consider a read correctly aligned if the reported alignment coordinates overlaps with the Mason-provided ground truth reference coordinates for the read. Over-aligned reads mean that the reads are reported as unmapped in the mason ground truth file, but the aligners have aligned them.

Single-end evaluation. We observe that strobealign is the fastest tool (Fig. 1) for read lengths of 150nt and larger, and Accel-Align is the fastest for the 100nt reads in our simulated experiments. Strobealign is about 2-3 times faster than minimap2 and 12-15 times faster than BWA and Bowtie2 across the datasets. Furthermore, in comparison to Accel-Align, the speedup that strobealign achieves over state-of-the-art aligners does not compromise on alignment accuracy (Fig. 2). For reads of 200nt and larger, strobealign has near identical accuracy to Bowtie2 and minimap2 and about 0.2% lower accuracy than BWA-MEM. For reads of length 100nt, strobealign

DRAFT

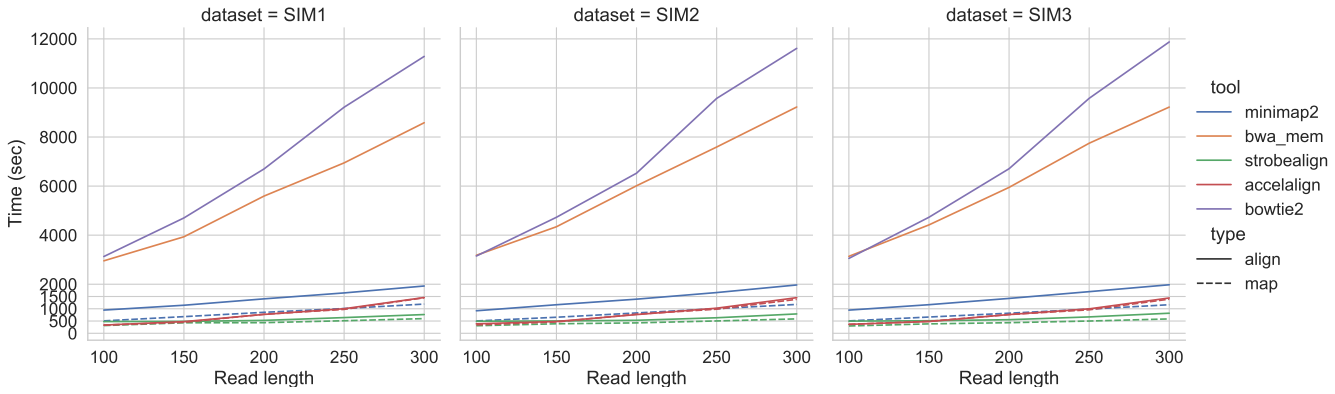

Fig. 1. Time to align 10M simulated single-end Illumina reads for the three simulated datasets (panels). The x-axis shows simulated read length and the y-axis shows the Wall clock time from after the index has been loaded into RAM.

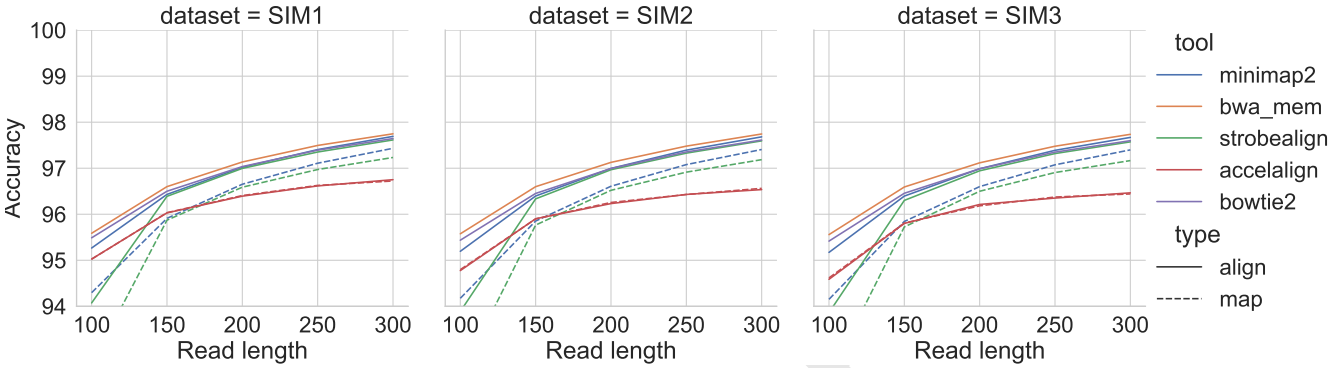

Fig. 2. Accuracy of aligned reads for the three simulated datasets (panels). The x-axis shows simulated read length and the y-axis shows the accuracy.

does not reach an accuracy competitive with state-of-the-art with the parameters we used in our experiments and spends significant time in rescuing state which in addition to accuracy also impacts the runtime.

As for the percentage of aligned reads (Fig. A.1) they are similar across tools with two exceptions. Firstly, strobealign aligns significantly fewer reads for the 100nt read datasets. Secondly, Bowtie2 aligns fewer reads for the simulated genome with the most variation (SIM3), aligning about 99.3-99.5% of the reads instead of 99.7-99.8% which is observed for the other aligners. This is because in default mode, Bowtie2 requires end-to-end alignments. The fraction of aligned reads can be improved with Bowtie2 by specifying local mode alignment (–local) at the cost of increased runtime. The number of reads reported as over-aligned in our analysis was identical for all tools in alignment mode (Fig. A.2) indicating that the tools find places to align the reads tagged as unaligned by mason_simulator.

In summary, for the single-end reads of lengths 150-200nt and above, we observe that strobealign offers a very competitive trade-off between speed and alignment accuracy and number of aligned reads to state-of-the-art aligners BWA-MEM, Bowtie2 and minimap2.

Paired-end evaluation. We observe that strobealign and Accel-Align are the fastest tools for read lengths of 150nt and larger (Fig. 3) with strobealign being more than 3 times faster than minimap2 and 10 times faster than BWA and Bowtie2 for reads of length 200-300nt. However, as for the single-end analysis, strobealign as opposed to Accel-Align achieves this speedup without compromising substantially on alignment accuracy (Fig. 4). For reads of 200nt and larger, strobealign has an accuracy similar to Bowtie2 and minimap2, and about 0.2% lower accuracy than BWA-MEM. Similarly to the single-end experiments, strobealign does not reach an accuracy competitive with state-of-the-art for reads of length 100nt with the parameters we used in our experiments.

We observe that the mapping modes of minimap2 and strobealign have, as expected, lower accuracy than their respective alignment modes. However, strobealign's mapping mode has a more substantial drop-off in accuracy to its alignment mode, compared to minimap2. This indicates that there might be room to develop a better strobemer-based scoring function that more frequently assigns the true mapping location with the highest score. As for runtime of computing candidate mapping locations with the seeds, we can compare the runtime between minimap2 and strobealign in mapping mode which excludes base level alignments with the ksw2 alignment module (Suzuki and Kasahara formulation (28)). We observe that strobealign computes mapping locations very quickly and scales well across typical short-read lengths. For example, strobealign is 5-6 times faster than minimap2 for read lengths of 300 (Fig. 3). We observed very marginal differences in both accuracy and runtime when running Accel-Align in mapping and alignment mode.

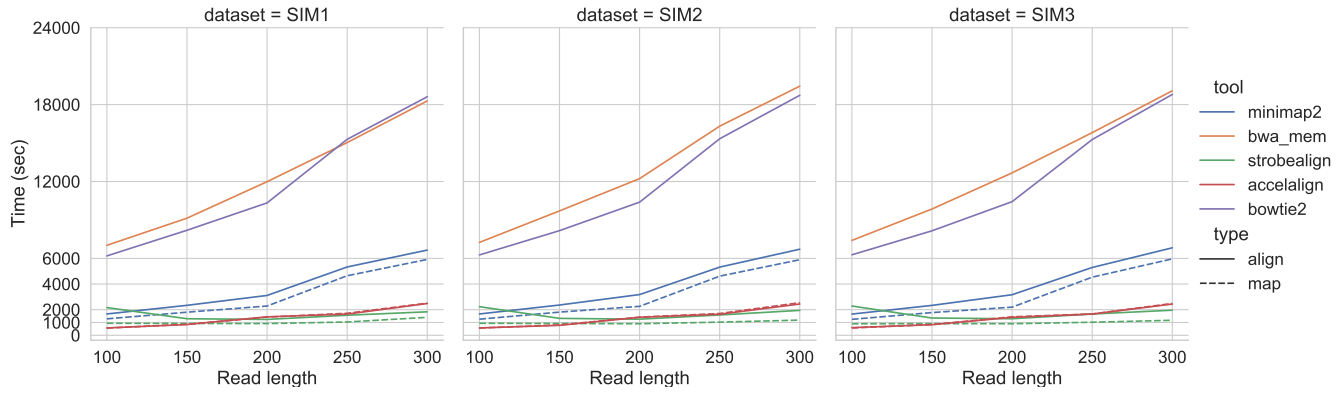

Fig. 3. Time to align 10M simulated Illumina paired-end reads for the three simulated datasets (panels). The x-axis shows simulated read length and the y-axis shows the Wall clock time from after the index has been loaded into RAM.

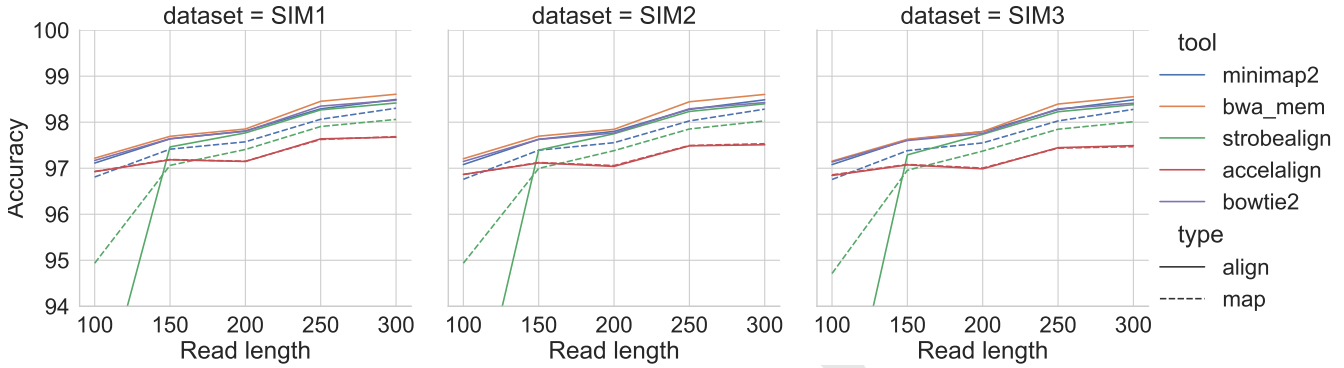

Fig. 4. Accuracy of paired-end aligned reads for the three simulated datasets (panels). The x-axis shows simulated read length and the y-axis shows the accuracy.

We also measured the time occupied by each module in strobealign's alignment paired-end alignment algorithm (Fig. 5), which includes creating strobemers, finding and constructing merged matches, rescuing reads without matches, and base level alignment with ksw2. As our total time between alignment and mapping modes in Figure 3 indicate, we see that strobealign for the 100nt paired-end reads spends substantial time in rescue mode (repetitive reads) and performs base level alignments for reads with many candidate sites of similar quality (Fig. 5). The experiments with read lengths of 200-300nt has the most balanced time across the three major steps of generating seeds, creating candidate map sites, and performing alignment. As the total runtime of strobealign is doubled or more in the alignment mode compared to the mapping mode, optimizing the base level alignment, e.g., with the Wavefront Alignment Algorithm (31) is therefore an interesting future venue as was also discussed in (20).

As for the percentage of aligned reads (Fig. A.3) they are similar across tools with the same two exceptions as for the single-end datasets. Firstly, strobealign aligns fewer reads for the 100nt read datasets, but not as much as for the single-end dataset. Secondly, similarly to the single-end analysis, Bowtie2 aligns fewer reads than the other aligners for the SIM3 datasets due to requiring end-to-end alignments. The number of reads reported as over-aligned is identical for all tools in alignment mode (Fig. A.4), as we also observed in the single-end experiment.

In summary, for the paired-end reads of lengths 150-200nt and above, we observe that strobealign offers a very competitive trade-off between speed and alignment accuracy and number of aligned reads to state-of-the-art aligners BWA-MEM, Bowtie2, and minimap2.

Multithreading evaluation. We ran the alignment tools with 4, 8, and 16 threads on the different datasets from the SIM3 genome. We observed that BWA-MEM, minimap2, and strobemap nearly halves the alignment time when doubling the number of threads suggesting that the tools utilized the resources well (Fig. A.5). Bowtie2 and Accel-Align also parallelized well, but not as well as the former three tools, particularly for the 250nt and 300nt reads. We also observe that Bowtie2 was faster than BWA-MEM in single thread mode, but is slower when multi-threaded (Fig. A.5).

Biological data. As we do not have the ground truth genomic location of the biological reads, we measured the concordance in alignment coordinates between the three fast aligners (Accel-Align, minimap2, and strobealign) to the two established short-read aligners BWA-MEM and Bowtie2. We ran all tools with default parameters, except Bowtie2 where we set the parameters –local and –X 1000 after personal communication with Ben Langmead, one of the authors of Bowtie2. For the alignments we used one thread per aligner. Similarly to the simulated data, we measured the genomic overlap of aligned reads between aligners (Fig. 6). We com-

DRAFT

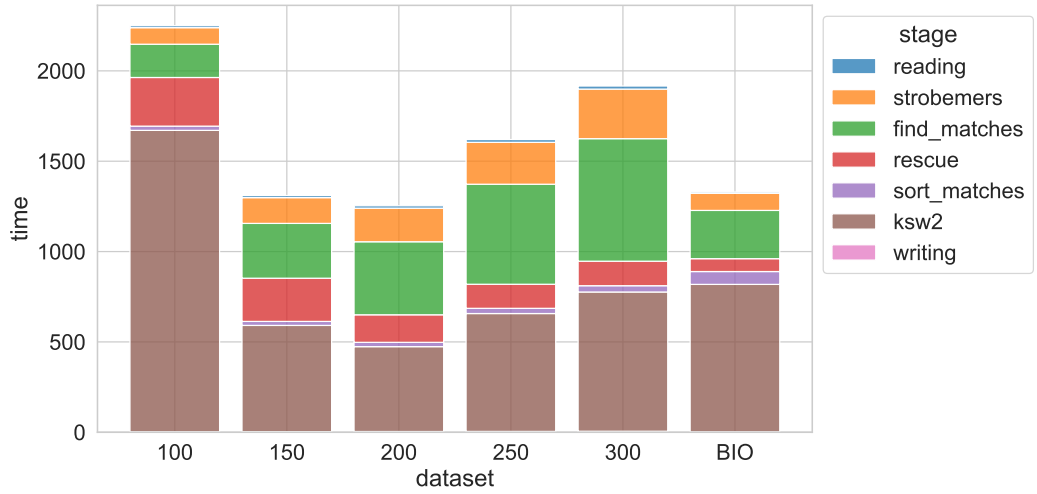

Fig. 5. The time spent at various steps in strobealign using one thread for the SIM3 datasets of 10 million paired-end reads of different read lengths as well as the BIO dataset of 4 million paired-end reads of length 250. Reading and writing refers to reading the fastq file and writing the output to SAM. The time spent on these steps are negligible when using one thread. The label 'strobemers' refers to the time to generate strobemer seeds from the reads, 'find_matches' refers to retrieving and creating merged matches from all strobemer seeds below the repetitive abundance threshold, 'rescue' refers to find merge matches in the rescue mode, 'sort_matches' sorts the matches with respect to the candidate map score, and 'ksw2' refers to the base level Smith-Waterman alignment.

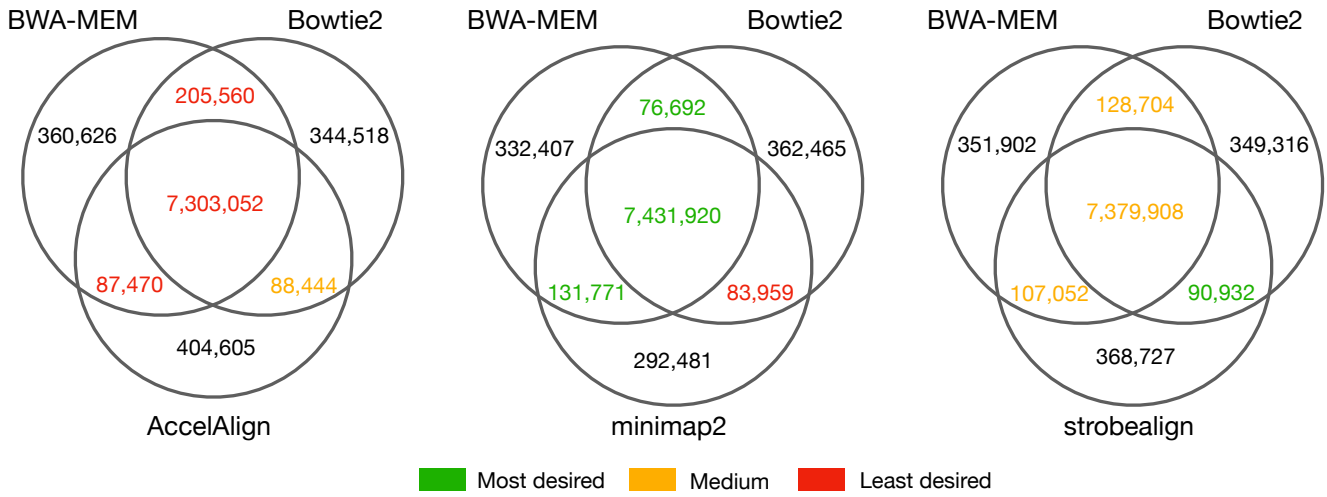

Fig. 6. The number of reads in the BIO dataset with genomic overlap between the two commonly used short-read aligners BWA-MEM and Bowtie2 and the three fast aligners Accel-Align, minimap2, and strobealign.

pute overlap between the 8 million individual reads from the 4 million paired-end reads in the BIO dataset.

Out of the total 8 million reads, we observe that 91.3%, 92.2%, and 92.9% of reads share genomic location among all three aligners when computing overlap with Accel-Align, strobealign, and minimap2, respectively. Minimap2 shares a substantial fraction of reads uniquely with BWA-MEM, while strobealign have the highest number of uniquely shared reads with Bowtie2. While our simulated results indicate that BWA-MEM do have the highest accuracy, albeit marginal, minimap2's distribution of a high number of overlapping reads with BWA-MEM and a low number with Bowtie2 is notable. Some of the unique overlap we see between BWA-MEM and minimap2 may be on the implementation level by choosing the same random mapping location in case of ties, as aligners have different methodologies to select locations in such scenarios.

As for the percentage of aligned reads, BWA-MEM and Bowtie2 aligned 99.7% and 99.5% reads, respectively, while Accel-Align, minimap2, and strobealign aligned 98.6%, 99.2%, and 99.4% reads. The alignment runtime for this dataset was 7,883s and 19,341s for BWA-MEM and Bowtie2, respectively, while it took 1,110s, 2,448s, and 1,292s for Accel-Align, minimap2, and strobealign, respectively. This makes Accel-Align the fastest aligner on this dataset. However, we note that a majority of strobealign's runtime was spent in the ksw2 alignment step on this dataset (Fig. 5) with a total of 4,416,516 calls to ksw2 for the 8 million mates, and a substantially larger portion of time is spend in the sorting matches. This indicates that more reads have many highly scoring candidate mapping regions. When strobealign was run in the mapping mode on this dataset it took only 509s,

compared to minimap2 in mapping mode that took 1,598s. Similarly to the simulated paired-end datasets, we conclude that optimizing the base level alignment, which is orthogonal to the seeding and candidate finding, can further greatly reduce strobealign's total runtime. On this dataset we also observe that Bowtie2 is significantly slower with –local mode (19,341s) compared to default mode (6,320s). We specified the –local parameter as the default alignment mode in Bowtie2 computes only end-to-end alignments (no softclipping) which lead to only 96.4% of the reads aligned.

In summary, strobealign is almost as fast as Accel-Align but with both substantially higher concordance alignments from either Bowtie2 or BWA (94.7% to 93.5%), and higher percentage of aligned reads (99.4% to 98.6%). Compared to minimap2, strobealign has lower concordance (94.7% to 95.6%), but more aligned reads (99.4% to 99.2%) and more than three times speedup in computing candidate matches. Our results on the BIO dataset suggests as for the simulated data, that strobealign offers a competitive trade-off in runtime and accuracy.

Indexing time and memory. Accel-Align indexed hg38 in 2m15s with a peak memory usage of 39Gb, minimap2 indexed hg38 in 2 minutes and 52 seconds with a peak memory usage of 13Gb, and strobealign indexed hg38 in 4m50s with a peak memory usage of 36Gb. Indexing for the BWT based read aligners is substantially slower but has much lower peak memory. BWA indexed hg38 in 1h1m minute with a peak memory usage of 4.5Gb. Bowtie2 indexed hg38 in 1h40m with a peak memory usage of 5.5Gb.

Memory usage during alignment. Strobealign does not currently implement a separate indexing step, which makes strobealign have a peak memory usage of 36Gb across all simulated datasets (Fig. A.6) and the BIO dataset when aligning to hg38. This peak is reached during indexing and makes the peak memory relatively high compared to the other aligners that build the index in a separate step (Fig. A.6). We believe that most large short-read datasets are usually aligned in multi-threaded mode on computing nodes or computers with more than 36Gb of RAM, for which this memory constraint should not be of practical limitation.

With the parameter settings investigated in this evaluation on hg38, strobealign stores 585,809,159 strobemer tuples in the flat vector. This corresponds to roughly 12Gb (585,809,159*(8+4+4+4)) for storing the hash value, reference id, start, and end position of the strobes. The hash table for storing k-mer positions in the flat vector consists of 542,925,043 unique hash values (8 Byte) that points to an offset in the flat vector (4 Byte) and a count (4 Byte). This corresponds to roughly 8.7Gb (542,925,043*(8+4+4)) to store only the leaves in the hash table. However, there is a large overhead in storing the internal nodes in the hash map (robin hood hash map in C++), estimated to roughly two times to memory for the leaf nodes, meaning that the hash table would be using roughly 26Gb. This results in a estimated peak memory during indexing as 12 + 26 +3 = 41Gb, where the 3Gb is for holding the reference sequences, which is roughly what we observe in our experiments. Theoretically, we believe that strobealign should only require 4 bytes more per seed than minimap2 (storing two strobe positions instead of one minimizer position), as they have a similar index construction. Further, minimap2 and strobealign have similar sampling densities. Minimap2 has an average minimizer sampling density of 2/11 for short reads, while strobealign has a strobemer sampling density of 1/5.

Discussion

We have presented a novel approach to compute seeds used for sequence mapping. We believe this seeding technique can offer competitive performance in several different mapping applications through the ability to use longer seeds compared to other approaches. We demonstrate the advantage of our approach in the application of short-read alignment by implementing our idea in a short-read alignment tool, strobealign. We demonstrate that strobealign achieves similar accuracy to minimap2 on both single-end and paired-end reads (Figs. 2,4) while being 3x faster (Figs. 1,3). Strobealign is also significantly faster than BWA-MEM and bowtie2 at a slight decrease in accuracy (0.1-0.2%). Accel-Align also achieves alignment speed faster than minimap2, but it trades speed for accuracy. However, since strobealign and Accel-Align achieves the speedup at different stages in the alignment pipeline (strobealign in the seeding stage and Accel-Align in the filtering stage), they have the potential to be combined.

We observed that our alignment speed significantly dropped for the BIO dataset (Fig. 5). This was due to the large number of calls to ksw2 for this dataset, becoming a computational bottleneck (Fig. 5). We may therefore further optimize this step by either considering a faster Smith-Waterman alignment such as the Wavefront Alignment Algorithm (31) as discussed in (20), or we may perhaps find a better strategy based on the seeds to select fewer candidate sites to align to.

While the memory usage of strobealign is relatively high (36GB) compared to the other tools, the usual setting in which hundreds of millions of short reads are aligned is usually on larger computing nodes with more than 36Gb of memory. The memory constraint will therefore not be of practical concern in such scenarios.

Future work. A natural future research direction is to adapt our idea to other mapping scenarios. Applications such as minimizer-based long-read alignment (16, 19) or transcriptomic long-read clustering (32) may greatly benefit from using longer seeds. Our technique may also improve computation speed of seeds for transcriptomic spliced alignment (33) as MEM finding is the current bottleneck. Another interesting direction is to explore if our seeding approach can be used for overlap detection in genome assembly as it has been demonstrated that minimizer-linking strategies have great success in this area (25).

As for short-read alignment with strobealign, we will continue improving some aspects of the algorithm further. For example, we observed the relatively large difference between mapping and alignment accuracy in strobealign (Figs. 2,4),

DRAFT

compared to minimap2. Such a difference motivates us to better tailor the candidate map-location scoring for strobemer seeds. We could also consider using strobemers together with the embedding step presented in Accel-Align, which can further speed up computations. An additional observation is that, since our seeds can match over indels, we may directly compute the size of the indel size by calculating the difference between the start and end strobe. There are caveats with this approach such that if two or more indels are present between the strobes it will invalidate such inference.

We also aim to continue working on the strobealign implementation. We will continue to increase alignment speed various modules in strobealign through engineering. For example, in rescue mode we could reserving a fixed but larger size of vectors holding candidate sites instead of dynamically updating the vector size when needed. We can also reduce peak memory in two ways. Firstly, strobealign currently does not have a separate index building step. Since peak memory is reached in the indexing step, it will improve the alignment peak memory to separate the index and alignment steps as is implemented in other aligners. When reading in the index for alignment, we may also discard the unnecessary hash value of the strobemer (16 bytes) present at construction for each match in the flat vector. Secondly, we can use more memory-efficient hash table representations and storing the information in the index more succinctly. For example, the counts in the hash table can be stored in 2-byte integers, the second strobe position can be stored as an offset (2-byte integer), and the reference IDs with a 2-byte integer for genome-based alignment.

Conclusion

We presented a novel strategy to compute seeds that can be used for candidate mapping-site detection in sequence mapping applications. We implemented our strategy in a short-read aligner strobealign. Strobealign shows a competitive trade-off in accuracy and runtime and is 3x faster than state-of-the-art aligner minimap2 while having roughly the same accuracy. Our seeding approach can potentially be used in many other applications that requires sequence mapping.

ACKNOWLEDGEMENTS

The computations were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) at Uppsala Multidisciplinary Center for Advanced Computational Science (UPPMAX) partially funded by the Swedish Research Council through grant agreement no. 2018-05973.

Bibliography

1. Heng Li and Richard Durbin. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*, 26(5):589–595, 01 2010. ISSN 1367-4803. doi: 10.1093/bioinformatics/btp698.
2. Heng Li. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem, 2013.
3. Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10(3): R25, 2009. doi: 10.1186/gb-2009-10-3-r25.
4. Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4):357–359, 03 2012. doi: 10.1038/nmeth.1923.
5. Michael Burrows and Wheeler D. J. A block-sorting lossless data compression algorithm. 1994.
6. Roy Lederman. A random-permutations-based approach to fast read alignment. *BMC Bioinformatics*, 14(5):S8, 2013. doi: 10.1186/1471-2105-14-S5-S8.
7. Francesco Vezzi, Cristian Del Fabbro, Alexandru I Tomescu, and Alberto Policriti. rna: a fast and accurate short reads numerical aligner. *Bioinformatics*, 28(1):123–124, Jan 2012. ISSN 1367-4811 (Electronic); 1367-4803 (Linking). doi: 10.1093/bioinformatics/btr617.
8. Matei Zaharia, William J. Bolosky, Kristal Curtis, Armando Fox, David Patterson, Scott Shenker, Ion Stoica, Richard M. Karp, and Taylor Sittler. Faster and more accurate sequence alignment with snap, 2011.
9. Yang Liao, Gordon K Smyth, and Wei Shi. The subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic acids research*, 41(10):e108–e108, 05 2013. doi: 10.1093/nar/gkt214.
10. Cole Trapnell, Lior Pachter, and Steven L. Salzberg. TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics*, 25(9):1105–1111, 03 2009. ISSN 1367-4803. doi: 10.1093/bioinformatics/btp120.
11. Alexander Dobin, Carrie A Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R Gingeras. Star: ultrafast universal rna-seq aligner. *Bioinformatics*, 29(1):15–21, Jan 2013. ISSN 1367-4811 (Electronic); 1367-4803 (Print); 1367-4803 (Linking). doi: 10.1093/bioinformatics/bts635.
12. Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic rna-seq quantification. *Nature Biotechnology*, 34(5):525–527, 2016. doi: 10.1038/nbt.3519.
13. Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4): 417–419, 2017. doi: 10.1038/nmeth.4197.
14. Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: Local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, page 76â85, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 158113634X. doi: 10.1145/872757.872770.
15. Michael Roberts, Wayne Hayes, Brian R. Hunt, Stephen M. Mount, and James A. Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20 (18):3363–3369, 07 2004. ISSN 1367-4803. doi: 10.1093/bioinformatics/bth408.
16. Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, Jul 2016. ISSN 1367-4811 (Electronic); 1367-4803 (Print); 1367-4803 (Linking). doi: 10.1093/bioinformatics/btw152.
17. Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18): 3094–3100, 05 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty191.
18. Chirag Jain, Sergey Koren, Alexander Dilthey, Adam M Phillippy, and Srinivas Aluru. A fast adaptive algorithm for computing whole-genome homology maps. *Bioinformatics*, 34(17): i748–i756, 09 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty597.
19. Chirag Jain, Arang Rhie, Haowen Zhang, Claudia Chu, Brian P Walenz, Sergey Koren, and Adam M Phillippy. Weighted minimizer sampling improves long read mapping. *Bioinformatics*, 36(Supplement_1):i111–i118, 07 2020. ISSN 1367-4803. doi: 10.1093/bioinformatics/btaa435.
20. Yiqing Yan, Nimisha Chaturvedi, and Raja Appuswamy. Accel-align: a fast sequence mapper and aligner based on the seed–embed–extend method. *BMC Bioinformatics*, 22(1):257, 2021. doi: 10.1186/s12859-021-04162-z.
21. Robert C. Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *bioRxiv*, 2020. doi: 10.1101/2020.09.29.319095.
22. Kristoffer Sahlin. Effective sequence similarity detection with strobemers. *Genome Research*, 31(11):2080–2094, 2021. doi: 10.1101/gr.275648.121.
23. Chen-Shan Chin and Asif Khalak. Human genome assembly in 100 minutes. *bioRxiv*, 2019. doi: 10.1101/705616.
24. Kristoffer Sahlin and Paul Medvedev. Error correction enables use of oxford nanopore technology for reference-free transcriptome analysis. *Nature Communications*, 12(1):2, 2021. doi: 10.1038/s41467-020-20340-8.
25. Bariş Ekim, Bonnie Berger, and Rayan Chikhi. Minimizer-space de bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer. *Cell Systems*, 12(10): 958–968.e6, 2021. ISSN 2405-4712. doi: https://doi.org/10.1016/j.cels.2021.08.009.
26. Jim Shaw and Yun William Yu. Theory of local k-mer selection with applications to long-read alignment. *bioRxiv*, 2021. doi: 10.1101/2021.05.22.445262.
27. Heng Li. Code examples of fast and simple k-mer counters for tutorial purposes. https://github.com/lh3/kmer-cnt/issues/4, 2020.
28. Hajime Suzuki and Masahiro Kasahara. Introducing difference recurrence relations for faster semi-global alignment of long sequences. *BMC Bioinformatics*, 19(1):45, 2018. doi: 10.1186/s12859-018-2014-8.
29. Justin M. Zook, David Catoe, Jennifer McDaniel, Lindsay Vang, Noah Spies, Arend Sidow, Ziming Weng, Yuling Liu, Christopher E. Mason, Noah Alexander, Elizabeth Henaff, Alexa B. R. McIntyre, Dhruva Chandramohan, Feng Chen, Erich Jaeger, Ali Moshrefi, Khoa Pham, William Stedman, Tiffany Liang, Michael Saghbini, Zeljko Dzakula, Alex Hastie, Han Cao, Gintaras Deikus, Eric Schadt, Robert Sebra, Ali Bashir, Rebecca M. Truty, Christopher C. Chang, Natali Gulbahce, Keyan Zhao, Srinka Ghosh, Fiona Hyland, Yutao Fu, Mark Chaisson, Chunlin Xiao, Jonathan Trow, Stephen T. Sherry, Alexander W. Zaranek, Madeleine Ball, Jason Bobe, Preston Estep, George M. Church, Patrick Marks, Sofia Kyriazopoulou-Panagiotopoulou, Grace X. Y. Zheng, Michael Schnall-Levin, Heather S. Ordonez, Patrice A. Mudivarti, Kristina Giorda, Ying Sheng, Karoline Bjarnesdatter Rypdal, and Marc Salit. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Scientific Data*, 3(1):160025, 2016. doi: 10.1038/sdata.2016.25.
30. Holtgrewe M. Mason: a read simulator for second generation sequencing data, 2010.
31. Santiago Marco-Sola, Juan Carlos Moure, Miquel Moreto, and Antonio Espinosa. Fast gap-affine pairwise alignment using the wavefront algorithm. *Bioinformatics*, 37(4):456–463, May 2021. ISSN 1367-4811 (Electronic); 1367-4803 (Print); 1367-4803 (Linking). doi: 10.1093/bioinformatics/btaa777.
32. Kristoffer Sahlin and Paul Medvedev. De novo clustering of long-read transcriptome data using a greedy, quality value-based algorithm. *Journal of Computational Biology*, 27(4): 472–484, 2020. doi: 10.1089/cmb.2019.0299. PMID: 32181688.
33. Kristoffer Sahlin and Veli Mäkinen. Accurate spliced alignment of long RNA sequencing reads. *Bioinformatics*, 07 2021. ISSN 1367-4803. doi: 10.1093/bioinformatics/btab540. btab540.

DRAFT

Supplementary Note A: Figures

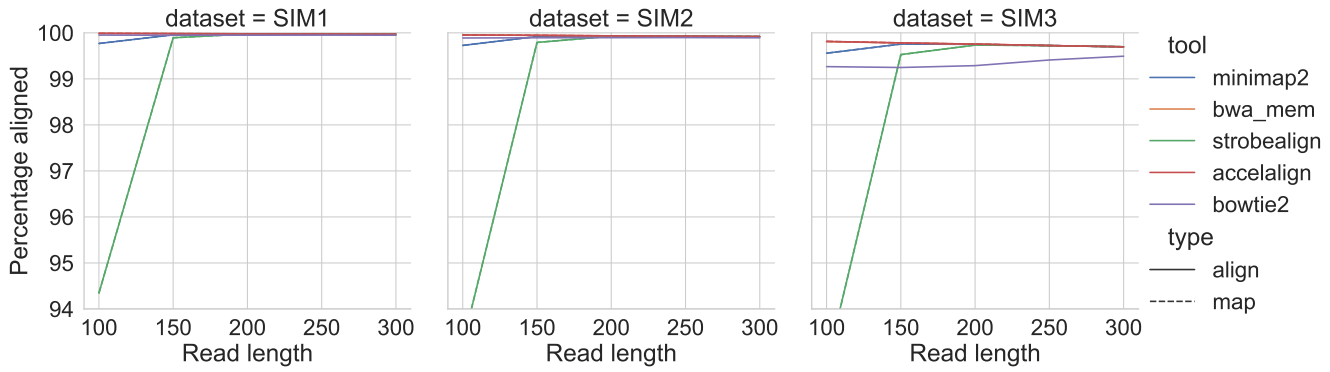

Fig. A.1. Percent aligned single-end reads for the three simulated datasets (panels). The x-axis shows simulated read length and the y-axis shows the percentage of aligned reads.

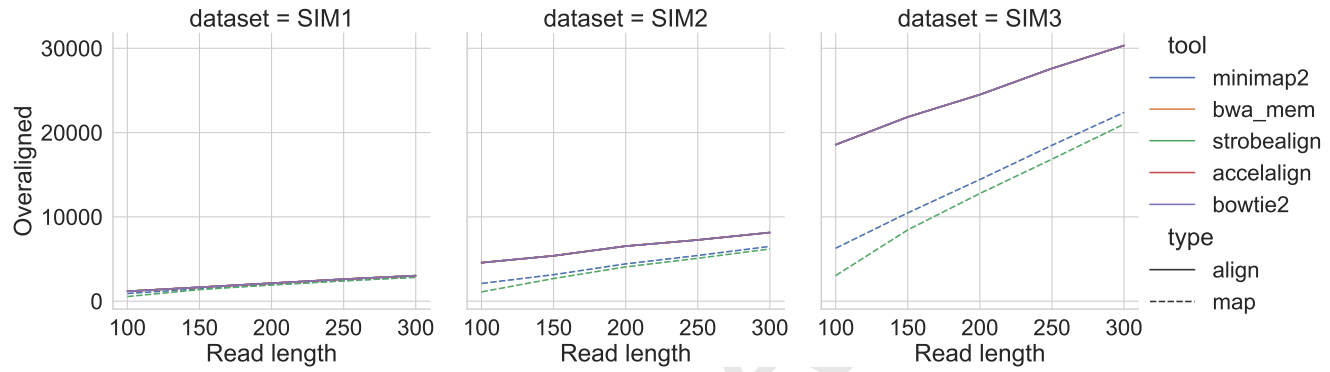

Fig. A.2. The number of aligned single-end reads for reads classified as unaligned by the mason simulator. All the aligners have the exact same number of over-aligned read which give plots with only one visible line.

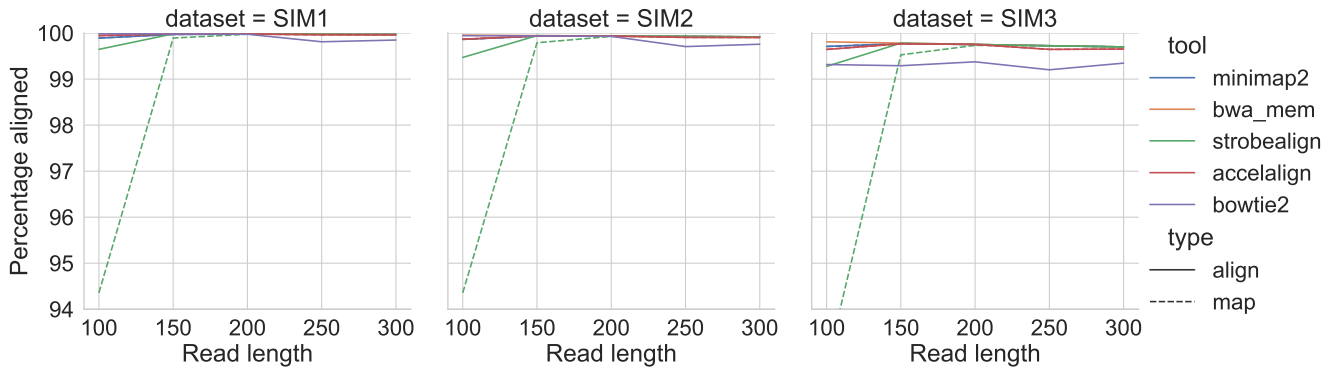

Fig. A.3. Percent aligned paired-end reads for the three simulated datasets (panels). The x-axis shows simulated read length and the y-axis shows the percentage of aligned reads.

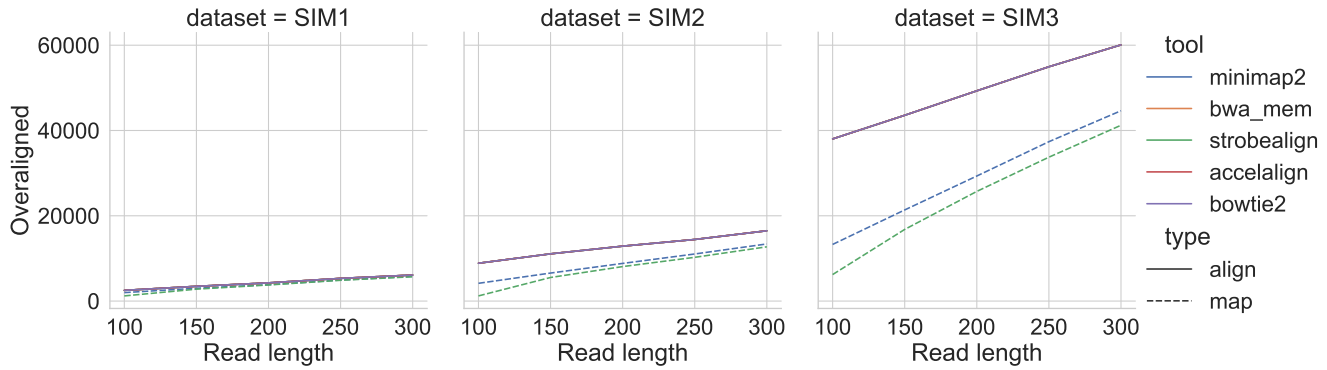

Fig. A.4. The number of aligned paired-end reads for reads classified as unaligned by the mason simulator. All the aligners have the exact same number of over-aligned read which give plots with only one visible line.

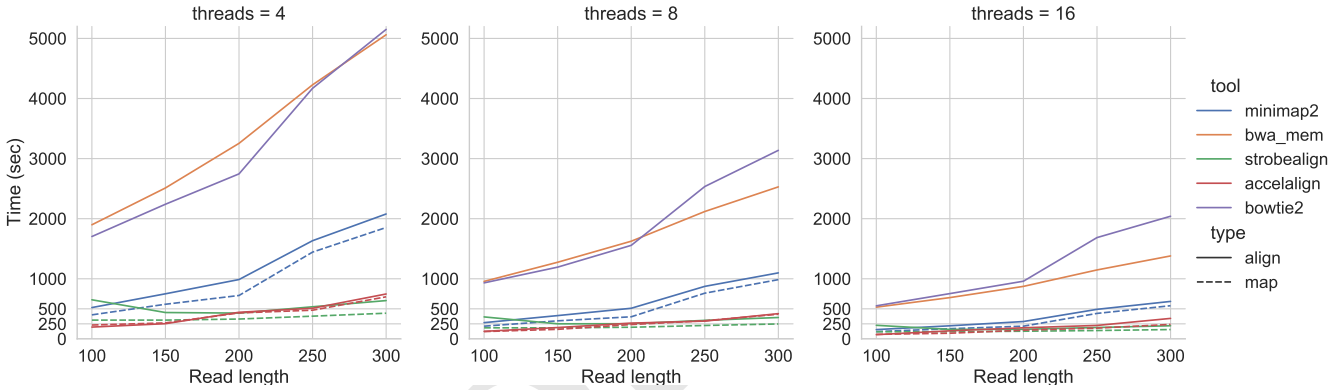

Fig. A.5. Runtime in multithread mode for the paired-end SIM3 datasets.

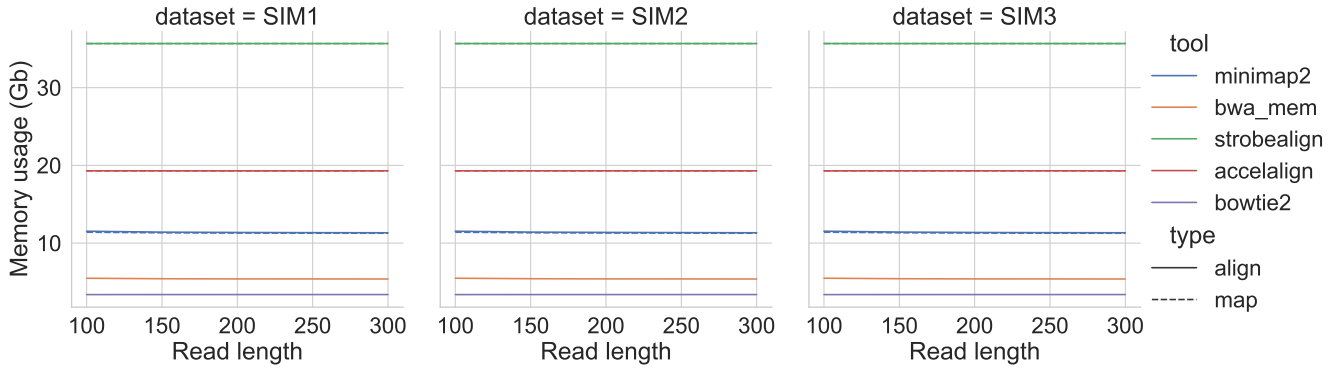

Fig. A.6. Memory usage across datasets.