Faster short-read mapping with strobemer seeds in syncmer space

Kristoffer Sahlin[1,✉]

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

Short-read genome alignment is a fundamental computational step used in many bioinformatic analyses. It is therefore desirable to align such data 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 a minimizer-based seeding-and-chaining strategy. Recently, syncmers and strobemers were proposed for sequence comparison. Both protocols were designed for improved conservation of matches between sequences under mutations. Syncmers is a thinning protocol proposed as an alternative to minimizers, while strobemers is a linking protocol for 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 protocol (randstrobes) to link together syncmers (*i.e.*, in syncmer-space) instead of over the original sequence. Our protocol allows us to create longer seeds while preserving mapping accuracy. A longer seed length reduces the number of candidate regions which allows faster mapping and alignment. We also contribute the insight that speed-wise, this protocol is particularly effective when syncmers are canonical. Canonical syncmers can be created for specific parameter combinations and reduce the computational burden of computing the non-canonical randstrobes in reverse complement. We implement our idea in a proof-of-concept short-read aligner strobealign that aligns short reads 3-4x faster than minimap2 and 15-23x faster than BWA and Bowtie2. Many implementation versions of, e.g., BWA, achieve high speed on specific hardware. Our contribution is algorithmic and requires no hardware architecture or system-specific instructions. Strobealign is available at https://github.com/ksahlin/StrobeAlign.

strobemers | syncmers | seed-and-extend | read mapping | short-read sequencing

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 read alignment in bioinformatics, many methods have been proposed for aligning Illumina reads. BWA (1, 2) and Bowtie/Bowtie2 (3, 4) which use the Burrows-Wheeler transform (BWT) (5) has received widespread use through their favourable memory and runtime trade-off. These aligners were for several years the two dominating genomic short-read aligners. Several alternatives approaches have been proposed, such as random permutations aligner (6), rNA (7), and SNAP zaharia2011faster, and subread (8) 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 (9, 10) or pseudo-alignment methods (11, 12) that are incredibly fast by not explicitly performing an exact alignment.

A pioneering approach was the use of minimizers as seeds (13, 14). Minimap2 was described initially 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 (15, 16) to produce seeds and then employs collinear chaining of seeds to produce candidate locations for the alignment step. Along the same trajectory Mashmap (17**?**) and winnowmap (18) designed for long-read made algorithmic contributions to minimizer-based alignment by considering and adjusting the densities and probabilities around the sampling of minimizers.

One computational hurdle is the length of the seeds, which inform the aligner of possible 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, reads can have many candidate regions that need to be evaluated before or at the alignment stage. This approach is usually referred to as seed-filter-extend. Accel-Align (19) instead proposes an idea to filter candidate mapping regions by computing a much cheaper hamming distance of an embedded representation of the read. Potentially interesting candidates have low embedded Hamming distance and are send for alignment. Accel-Align was shown to outperform minimap2 in terms of speed.

This work describes a novel seeding technique that allows us to use much longer seed lengths. 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 advancement in sequence comparison; syncmers (20) and strobemers (21). 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 protocol) between two sequences. Syncmers was proposed as an alternative to the minimizer thinning technique (i.e., subsampling), while strobemers was proposed as a linking protocol for gapped seeds, an alternative to k-mers. Syncmers and strobemers can be used in combination in what becomes a high-speed indexing protocol, roughly cor-

DRAFT

responding to the speed of computing minimizers. Our technique is based on first computing canonical open syncmers over the reference, then link the syncmers together using the randstrobe linking protocol. The syncmers are here the strobes in the strobemers. 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 of the same length as the single k-mer seed while achieving similar accuracy. Using longer seeds reduces the computational burden of evaluating candidate mapping sites. Furthermore, because we use parameters such that the syncmers become canonical, we only need to compute the syncmers once, while the randstrobes can be computed very fast in syncmer-space.

In genome assembly and error correction, ideas to work in minimizer-space have been proposed through the use of paired minimizers (22, 23) and k consecutive minimizers (k-min-mers) (24) to represent overlaps and assembly graphs. These concepts roughly correspond to what was later described as minstrobes in (21). Here we instead use the randstrobe protocol for linking that is more efficient at creating matches than minstrobes (21).

While the main contribution in this work is the idea for a novel seeding approach, we also implement our idea in a proof-of-concept tool that we call strobealign. Strobealign aligns simulated single-end Illumina reads 3-4x faster than minimap2 and 15-23x faster than BWA-MEM and Bowtie2 with comparable accuracy at various indel and substitution rates. In addition, strobealign is faster and more accurate than the recently proposed Accel-Align. Furthermore, our seeding is fast. It takes about 6 minutes to index hg38. While several engineering steps remain in our implementation to make it competitive to state-of-the-art programs (such as multithreading, separate building of index, and reducing peak memory usage of indexing), we believe that the idea can reach widespread use. Particularly since the idea can be applied in other mapping scenarios such as long-read mapping and long read overlap detection. Finally, our contribution (seeding) is orthogonal to that of Accel-Align (filtering), which means that, potentially, they 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 (20) from the reference sequences, then link the syncmers together using the randstrobe linking protocol (21) with two strobes. The study introducing strobemers (21) described the protocols 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 is very fast. We can compare a smaller set of syncmers to decide on 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, we can afford to combine seeds of much longer length (*e.g.*, two strobes of length 22 as used in this study) compared to other approaches while having a broadly similar accuracy. Having a much 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. An additional benefit is that we can easily compute a candidate match score as the span of the overlapping matches on the read times the number of hits. In the next section, we describe in detail the idea and our proof-of-concept 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 use h to denote a hash function $h:\sum^* \rightarrow \mathbb{Z}$ mapping strings to random numbers. We refer to a *subsequence* of a string as a set of ordered letters that can be derived from a string by deleting some or no letters without changing the order of the remaining letters. A substring is a subsequence where all the letters are consecutive. If $s[i,k]$ is identical to a k-mer $t[i',i'+k]$ in string t, we say that the k-mers *match*, and that the match occurs at position i in s (and i' in t). Furthermore, let $f(i,k,s)$ be any function to extract a subsequence of length k with first letter at position i from s. For example, for k-mers we have $f(i,k,s)=s[i,k]$. If $f(i,k,s)$ is identical to $f(i',k,t)$, we say that the subsequences match. We will denote a match as m and use $m.q_s, m.q_e, m.r_s, m.r_e$ to denote the query (read) start and end positions and the reference start and end positions of the match, respectively.

Syncmers. Syncmers was described in (20) and consists of three parameters (k,s,t) where k is the length of syncmers. We will here consider open syncmers. The open-syncmer protocol compares the $ks+1$ consecutive s-mers within a k-mer and selects the k-mer as a syncmer if the smallest s-mer occurs at position $t \in [0, ks+1]$ within the k-mer. With the smallest s-mer, we mean the hash value that the s-mer produces.

Strobemers. Strobemers was described in (21) and consists of four parameters (n,m,w_min,w_max) where n 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 of order two (i.e., $n=2$). The randstrobe protocol links together a strobe s_1 with a strobe $s_2=s'$ if s' minimizes $h(s_1)+h(s') \mod q, \forall s' \in [w_{min}, w_{max}]$, where q is a positive integer and s' occurs in interval $[w_{min}, w_{max}]$ downstream of s_1. Originally, strobemers were described to be produced by considering all k-mers. In this work, we will compute the randstrobes over syncmers. This means that s_1, s_2, and s' above are syncmers.

Indexing. We first construct syncmers and then link them together to form strobemers of order two. A neat characteristic with syncmers is that they allow canonical representation if choosing parameter t at the center of the syncmer, i.e., having

DRAFT

an odd value of $k-s+1$ and $t = \lceil (k-s+1)/2 \rceil$. Conveniently, it was shown that this is the optimal parameter setting for t for sequence mapping (25). We compute such canonical open syncmers (using $k = 22$, $s = 18$, $t = 3$ as default values) to produce a thinning of roughly 20% of the k-mers sampled, which is similar to $w = 10$ in the minimizer protocol. We then compute randstrobes over the syncmers in both directions on the sequence. 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 very unlikely to overlap, and $w_{max} = k/(k-s+1)+10$ is chosen based on simulation. These are parameters that can be specified in our tool.

We store tuples (h_k, r_{id}, r_s, r_e) in a flat vector V where h_k is the hash value of the computed strobemer k (64 bit integer) using robin hood hashing (GitHUb repository robin-hood-hashing), r_{id} is the reference id (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 h_k and construct a hash table (robin hood 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 (14, 19). 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. Randstrobes consisting of two syncmers are produced from the query sequence (*e.g.*, a read) in forward orientation in the same way as we index the reference. For reverse orientation, since the created syncmers are canonical, we can simply compute reverse complemented randstrobes by iterating over the syncmers in reverse order. 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 $m = (q_s, q_e, r_{id}, r_s, r_e)$ a *match* between the query and the reference. We then construct *merged matches* $\mathcal{M}$ (a set of overlapping matches, roughly) in the following manner. We iterate over the matches first in forward order then in reverse in order on the query sequence and join two matches m and m' into a merged match if it holds that (i) $m.r_{id} = m'.r_{id}$, (ii) $m.q_s < m'.q_s$, (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, (i-iii) the matches need to overlap on both query and reference, and (iv) two strobemer matches need to have consistent ordering of the four strobes on the reference and the query. 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. 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 and given direction, 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.

Computing MAPQ score. After merging matches, each merged match $\mathcal{M}$ consists of a number of matches $|\mathcal{M}|$ and a total span range on both the query $\mathcal{M}.q_e - \mathcal{M}.q_s$ and the reference $\mathcal{M}.r_e - \mathcal{M}.r_s$. We define the score s of $\mathcal{M}$ as $s_{\mathcal{M}} = (\mathcal{M}.q_e - \mathcal{M}.q_s)|\mathcal{M}|$. 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. Minimap2 also uses a second pass (rescue step) for short reads without good mapping sites. We do not implement such a step.

Alignment. 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 (26), which is also used in AccelAlign and minimap2.

Other optimizations. In addition to the drop-off parameter, we employ two types of optimizations to not consider unnecessary candidates. First, if we encounter a perfect alignment, 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 ksw for remaining candidates since a call to ksw, as described in the previous subsection, implies that the edit distance is at least 1.

Example parameters and implications. Here we give some numbers on what is expected by our seeding approach

DRAFT

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.

Strobealign is proof-of-concept. Several engineering steps are remaining to make strobealign competitive with other state-of-the-art aligners for, *e.g.*, Illumina paired reads. We describe them here. Firstly, strobealign needs to be parallelized. Since strobealign is a seed-filter-extend algorithm, parallelization can be implemented in a similar manner to other alignment programs such as minimap2 and Accel-Align. Secondly, strobealign only aligns reads in single-end mode. Rescuing mates in paired mode could be added to the algorithm similar to, e.g., BWA-MEM (2), which could further improve accuracy. Thirdly, strobealign currently does not have a separate index building step. Since peak memory is reached in the indexing step, it will improve the performance to have separate index and alignment steps. Fourthly, strobealign currently only maps reads in fasta format. Nevertheless, the implementation can robustly index a genome and align fasta formatted reads in single-end mode using a single thread.

Results

We evaluated strobealign to four state-of-the-art short-read aligners, BWA-MEM (v0.7.17), Bowtie2 (v2.3.5), Minimap2 (v2.17), and Accel-Align (commit 01d18b2). As strobealign is not parallelized, we ran all the tools using one thread. We used short read mode (-x sr) in minimap2 and default parameters for the rest of the aligners.

Simulated data. We used mason2 (mason_variator) (27) to simulate VCF files with variants to hg38 at various SNP and indel rates. 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 then simulate two additional variation files with a denser set of variations. 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). These three datasets represent genomes at different variation rates to hg38. For each of the genomes SIM1, SIM2, and SIM3, we used mason_simulator to simulate Illumina single-end reads with lengths 100,150,200,250, and 300. We then mapped the reads simulated from SIM1, SIM2, and SIM3 to hg38 with the aligners.

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 this information from the standard output of strobealign, minimap2, and Accel-Align, subtracting this time. As described in (19), larger indexes take a longer time to load from disc, although the difference will be 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. We do not subtract this time for BWA-MEM and Bowtie2 as they cannot be parsed from the output, but their indices are much smaller and fast to load ((19) reported about 6-9 seconds). Accuracy is the fraction of correctly aligned reads out of all aligned reads. We consider a read correctly aligned if the reported alignment overlaps with the Mason-provided ground truth reference coordinates. Over-aligned reads mean that the reads are reported as unmapped in the mason ground truth file, but the aligners align them.

We observe that strobealign is the fastest tool (Fig. 1) across all read lengths in our benchmark. Accel-Align achieves speed closest to that of strobealign. However, Accel-Align's accuracy deteriorates with the read length (Fig. 2). In contrast, strobealign produce alignments very fast while also achieving comparable accuracy (Fig. 2) and a similar number of mapped reads (Fig. 3) to the other popular aligners BWA-MEM, minimap2, and Bowtie2, except for the 100nt reads. For the 100nt reads, strobealign aligns significantly fewer reads and has lower accuracy, which is a consequence of both our approach and specific parametrization, which does not allow as many strobemers to be extracted from the shorter reads.

All the aligners produce an identical number of over-aligned reads (Fig. A.1), indicating that these reads may fit on the genome, while mason2 uses a cutoff for what is considered to be alignable.

Indexing time and memory usage. Accel-Align indexed Hg38 in 2 minutes and 15 seconds 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 6 minutes and 10 seconds with a peak memory usage of 59Gb. Indexing for the BWT based read aligners is slower. BWA indexed Hg38 in 1 hour and 1 minute with a peak memory usage of 4.5Gb. Bowtie2 indexed Hg38 in 1h 40 minutes with a peak memory usage of 5.5Gb.

Alignment memory usage. Currently, the memory usage of strobealign is high (Fig. 4). This is mainly due to implementation and can be reduced, as we will discuss here. 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

DRAFT

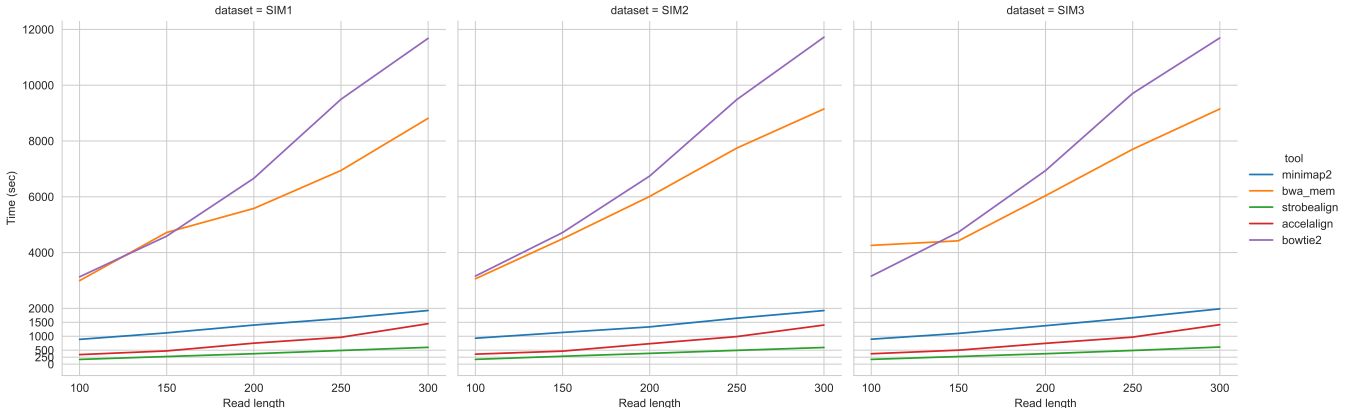

Fig. 1. Time to align 10M simulated Illumina reads for the three simulated datasets (panels). The x-axis shows simulated read length and the y-axis shows the Wall clock time from after the index has been loaded into RAM.

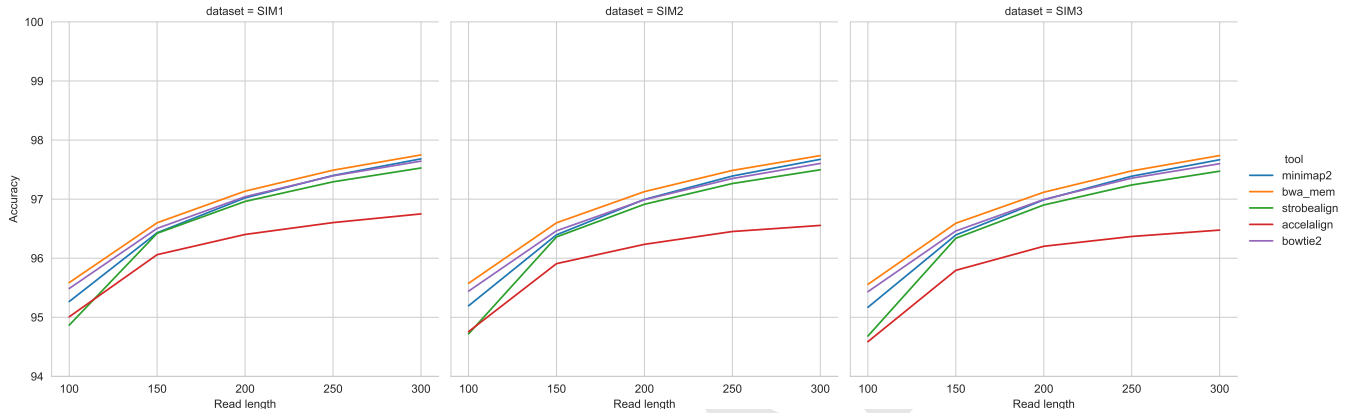

Fig. 2. Accuracy of aligned reads for the three simulated datasets (panels). The x-axis shows simulated read length and the y-axis shows the accuracy.

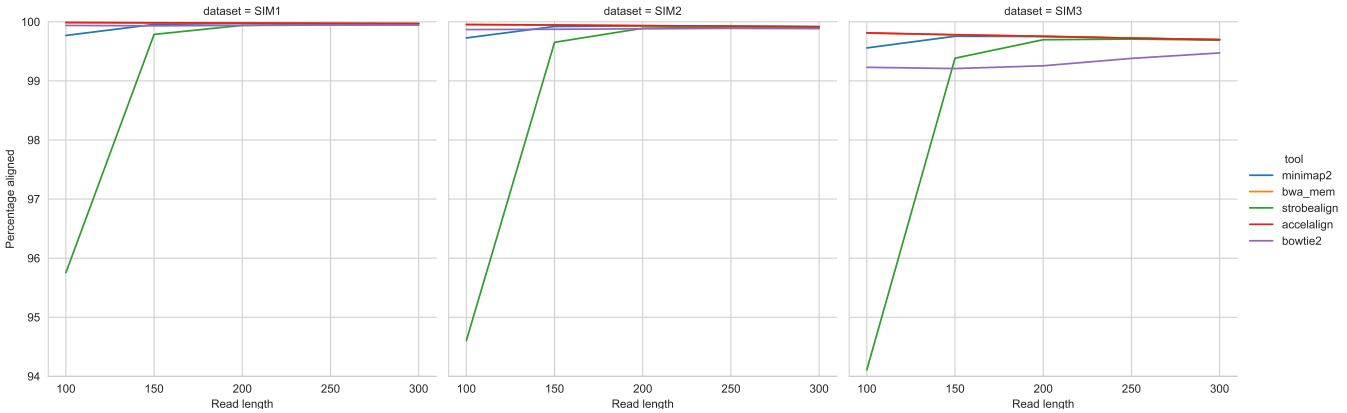

Fig. 3. Percent aligned reads for the three simulated datasets (panels). The x-axis shows simulated read length and the y-axis shows the percentage of aligned reads.

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. In addition, strobealign removes the redundant hash value from the flat vector once the hash table has been created, yielding a smaller flat vector of about 7Gb in size. This results in a total peak memory during indexing as 12 + 26 + 7 +3 = 48Gb, where the 3Gb is for holding the reference sequences. Currently, the peak memory usage we observe is higher (59Gb), and we will investigate where this comes from. Theoretically, we believe that strobealign should only require 4 bytes per seed more than minimap2 (storing two strobe positions instead of one minimizer position), as they have a similar index implementation. Further, minimap2 has an average minimizer sampling density of 2/11 for short reads, while strobealign has a strobemer sampling density of 1/5, which are similar.

Discussion

We have presented an idea for seeding and described how it could be used for candidate mapping selection. We implemented our idea into a proof-of-concept tool, strobealign. Despite the competitive area of short read alignment, we

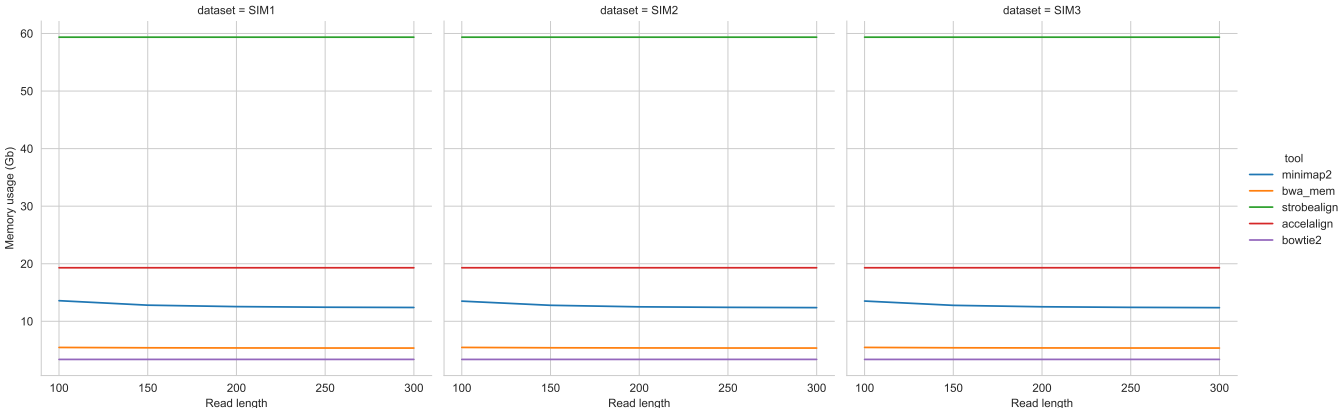

Fig. 4. Memory usage across datasets.

demonstrate that our implementation achieves similar accuracy to minimap2 while being 3-4x faster. We find this notable as minimap2 is both an algorithmic and engineering tour de force. Ultimately, our seeding strategy may be integrated into the well-established minimap2, which may also alleviate the engineering issues we currently face (peak memory usage), but this would require a larger modification to the minimap2 codebase.

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.

Future work. We will continue to improve the engineering aspect of strobealign. Particularly we aim to reduce peak memory which can be achieved by using 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.

We will implement the four remaining engineering tasks to make the aligner useful in real scenarios. Another interesting direction is to explore if our seeding approach can be used for graph alignment as our gapped randstrobe seeds can span over small variations. We also consider using strobemers together with the embedding step presented in Accel-Align, which could further speed up computations. Finally, since the core idea is to increase seed length, our idea may apply to other scenarios such as long-read mapping or possibly even read-overlapping overlapping for assembly minimizer-space graphs (22, 24).

Conclusion

We presented a novel strategy to compute seeds to be used as candidate sites for mapping together with a merging step of matches that is made practically possible due to that matches can span (and overlap) substitutions and indels. We implemented our strategy in a proof-of-concept aligner and mapping tool strobealign 3-4x faster than state-of-the-art aligner minimap2 while having roughly the same accuracy. strobealign demonstrates a new approach for seeding, which can potentially be used in many other applications requiring 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).

DRAFT

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. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18): 3094–3100, 05 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty191.
15. 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.
16. 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.
17. 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.

18. 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.
19. 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.
20. 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.
21. Kristoffer Sahlin. Strobemers: an alternative to k-mers for sequence comparison. *bioRxiv*, 2021. doi: 10.1101/2021.01.28.428549.
22. Chen-Shan Chin and Asif Khalak. Human genome assembly in 100 minutes. *bioRxiv*, 2019. doi: 10.1101/705616.
23. 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.
24. Barış Ekim, Bonnie Berger, and Rayan Chikhi. Minimizer-space de bruijn graphs. *bioRxiv*, 2021. doi: 10.1101/2021.06.09.447586.
25. 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.
26. 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.
27. Holtgrewe M. Mason: a read simulator for second generation sequencing data, 2010.

DRAFT

Supplementary Note A: Figures

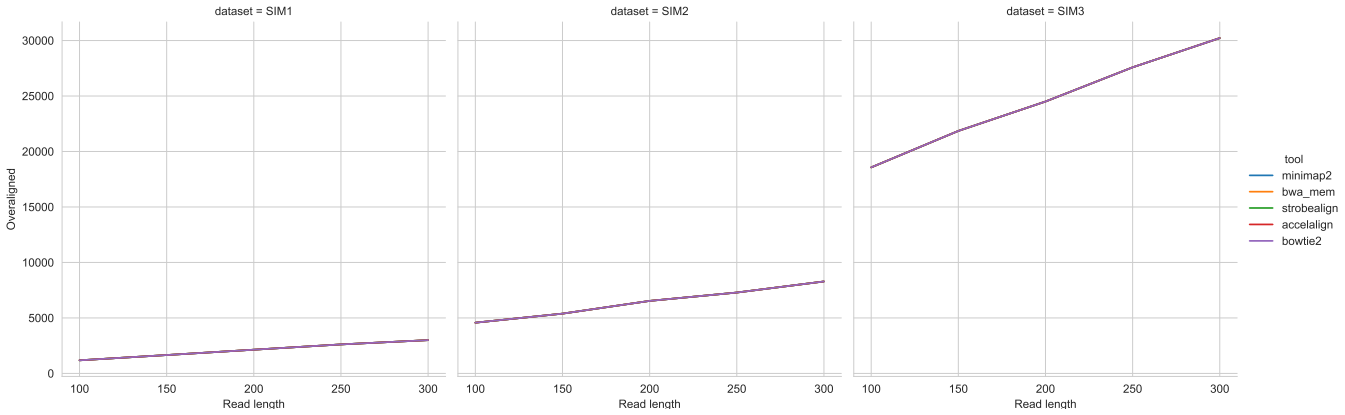

Fig. A.1. The number of aligned reads for reads classified as unaligned by the mason simulator. All the aligners have the exact same number of overaligned read which give plots with only one visible line.

DRAFT