Flexible seed size enables ultra-fast and accurate read alignment

Kristoffer Sahlin[1,✉]

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

Read alignment to genomes is a fundamental computational step used in many bioinformatic analyses, and often, it is the computational bottleneck. Therefore, it is desirable to perform the alignment step as fast as possible without compromising accuracy. Most alignment algorithms consider a seed-and-extend approach, where the time-consuming seeding step identifies and decides on candidate mapping locations. Recently, several advances have been made on seeding methods for fast sequence comparison.

We combine two such methods, syncmers and strobemers, in a novel seeding approach for constructing dynamic-sized fuzzy seeds and implement the method in a short-read aligner, strobealign. We benchmark strobealign to other traditional and recently proposed aligners on simulated and biological data and show that strobealign is an order of magnitude faster than traditional aligners such as BWA and Bowtie2 while being both faster and more accurate than more recently proposed aligners. Our aligner can free up substantial time and computing resources needed for read alignment in many pipelines.

Availability: https://github.com/ksahlin/strobealign.

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

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

Introduction

Aligning Illumina sequencing reads to a reference genome is the first step in many analyses pipelines. Due to the fundamental role of short-read alignment in bioinformatics there has been considerable work in this area. 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 alternative 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. A comprehensive listing of alignment techniques is found in (10). There are also several major contributions in data structures and algorithms in splice alignment of RNA-seq data (11, 12) or pseudo-alignment methods (13, 14) that are incredibly fast by not explicitly performing an exact alignment.

A pioneering approach was the use of minimizers (15, 16) as seeds in alignment and overlap detection algorithms (17, 18). Minimap2 was initially described for long-read alignment but has shown similar alignment accuracy of short reads as popularly used BWA and Bowtie2. Minimap2 uses minimizers as seeds and then employs collinear chaining to produce candidate locations for the alignment step. Along the same trajectory, Mashmap (19) and Winnowmap (20) were 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. Alignment algorithms often need to use a shorter seed length than what gives unique matches in a genome to have good alignment accuracy. Therefore, seeds may produce many candidate regions that need to be filtered based on some score before the alignment stage. Therefore, alignment methods are usually described to employ a seed-filter-extend approach, where the seeding and filtering are at the heart of an aligner's performance. In (21), the authors propose an alternative approach 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. Their aligner, Accel-Align, outperforms other aligners in terms of speed.

While filtering is an important step for candidate selection, reducing the false matches caused by repetitive seeds is preferable. To do so, one has to increase seed size. In this work, we describe a seeding approach for creating seeds with variable sizes that are fuzzy, i.e., they can match over mutations. Our approach allows us to use much longer seed lengths than other approaches without loss in accuracy. Furthermore, longer seed lengths reduce the number of candidate sites to evaluate, allowing much faster mapping and alignment. Our contribution is based on two recent advancements in the area of sequence comparison; syncmers (22) and strobemers (23). Both syncmers and strobemers have been demonstrated to improve sequence similarity detection (23, 24). Syncmers were proposed as an alternative to the minimizer subsampling technique. In contrast, strobemers were proposed to produce gapped seeds as an alternative to k-mers and spaced k-mers.

Here we show that syncmers and strobemers can be combined 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 (22), then producing strobemers (23) formed from linking together syncmers occurring close-by on the reference using the randstrobe method. A consequence is that

DRAFT

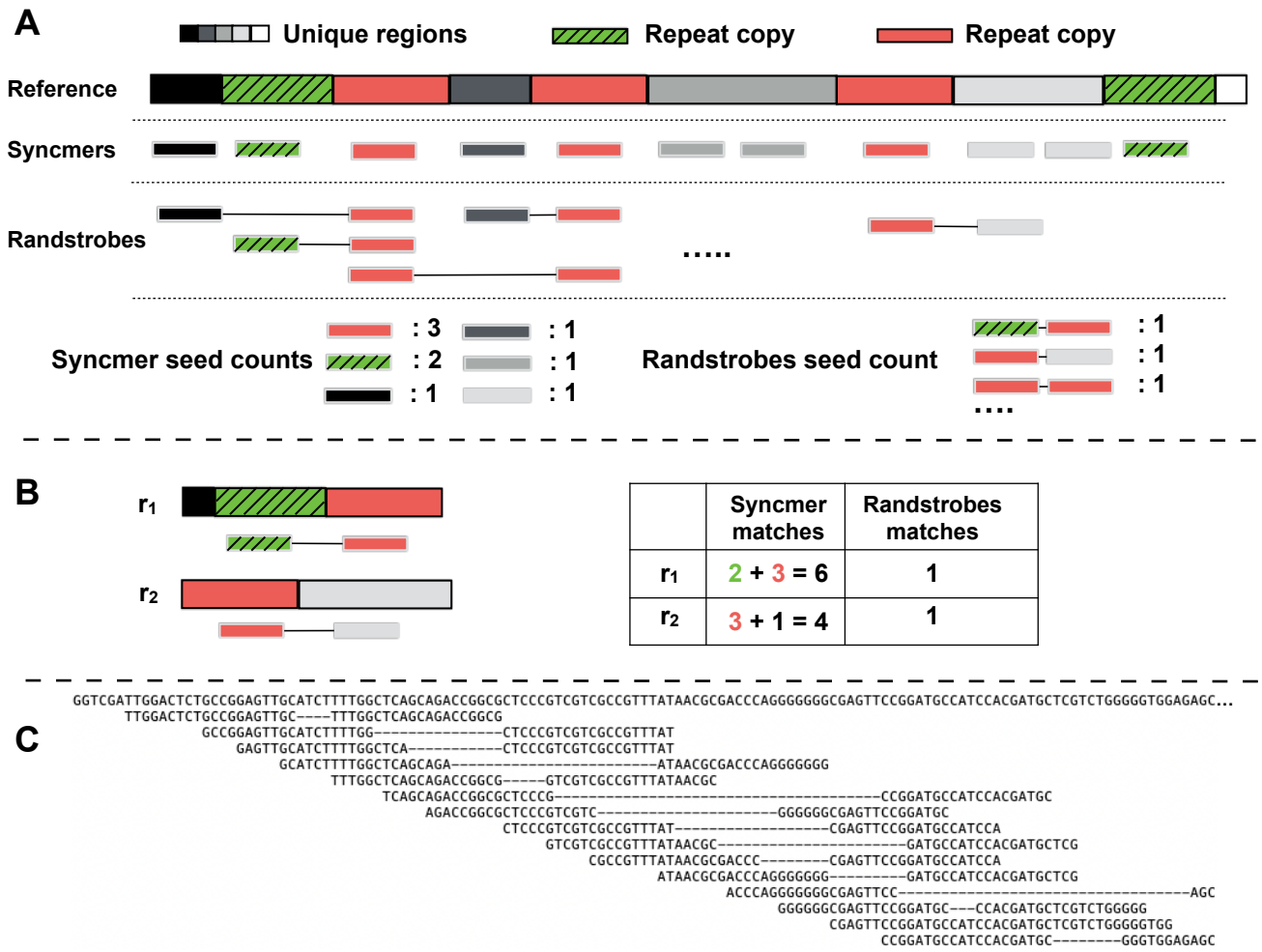

Fig. 1. Overview of seeding construction. (A) Open syncmers are constructed from the sequence, and randstrobes are constructed by linking together syncmers. The second strobe is sampled with a minimum and maximum spread between $[w_{min}, w_{max}]$ strobes downstream from the first strobe. While syncmers may occur several times due to repeats on the genome (red and green boxes), randstrobes are less repetitive. (B) Two reads, r_1 and r_2, are mapped to the reference. Finding matches using only syncmers creates several candidate mapping locations, while the randstrobes are unique in this scenario. In the illustration, the syncmers are spread out for visual purposes. Panel C shows a real sampling of randstrobes on a DNA sequence and their spread using $(2, 20, 4, 11)$ with sampling skew.

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 length 20) as a strobemer seed and achieve similar accuracy to using individual minimizers.

Our first 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 seeding method constitutes a novel algorithm class according to the comprehensive aligner classification table in (10), with our method being hashing of variable-length fuzzy seeds. Our second contribution is implementing our seeding technique in a short-read alignment tool, strobealign. We use simulated and biological data to show that strobealign is an order of magnitude faster than traditional aligners such as BWA and Bowtie2 while being faster and more accurate than more recently proposed aligners. Interestingly, we observe that subsampling methods such as strobealign and minimap2 can also be more accurate than BWA-MEM on high diversity datasets. Strobealign reaches peak performance on paired-end reads of lengths 150-300 nucleotides (nt), which is well suited for advances in short-read sequencing length and throughput, such as Illumina's Chemistry X reads are claimed to be two times longer (25). Strobealign's speed can remove the alignment bottleneck and free up substantial computing resources in many pipelines.

Results

Method overview. We present a seeding method for sequence similarity search that is based on a combination of two previously published techniques syncmers (22) and strobemers (23). The main idea of the seeding approach is to create fuzzy seeds by first computing open syncmers from the reference sequences, then linking the syncmers together using the randstrobe method (23) with two syncmers (Fig 1A). Our fuzzy seeds enable us to use larger seed lengths that are more likely to be unique (concept illustrated in Fig 1B) while still allowing mutations or read errors between the syncmers. Figure 1C illustrates the seeds extracted from a DNA se-

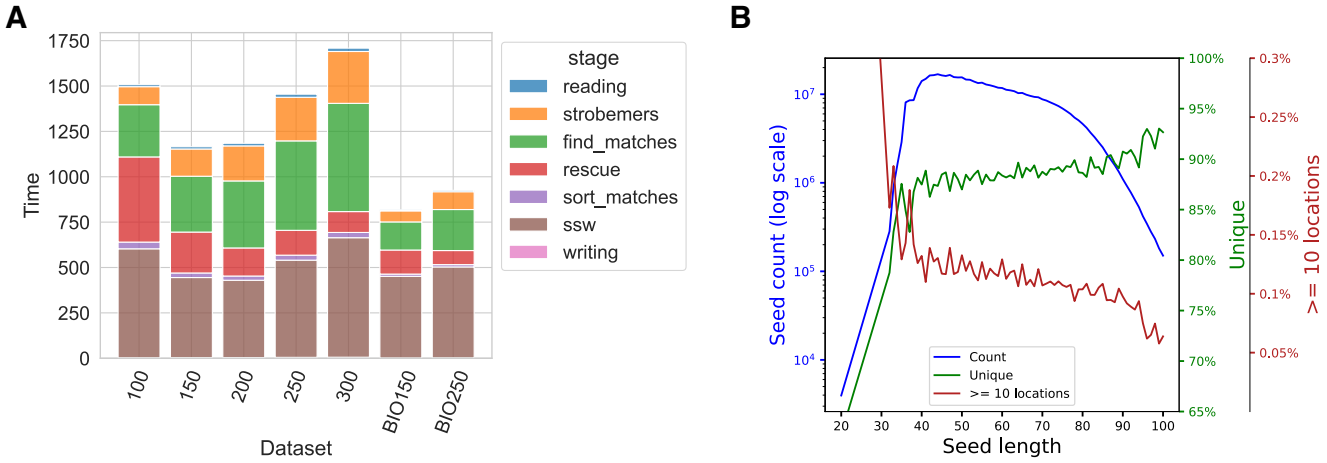

Fig. 2. Time allocation and seed uniqueness in strobealign. Panel A shows the time spent at various steps in strobealign using one thread for the SIM3 datasets of 10 million paired-end reads of different read lengths and the BIO150 and BIO250 datasets of 4 million paired-end reads of length 250. Reading and writing refer to reading the fastq file and writing the output to SAM. The time spent on these steps is 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 finding merge matches in the rescue mode, 'sort_matches' sorts the matches with respect to the candidate map score, and 'ssw' refers to the base level Smith-Waterman alignment. Panel B shows strobealign's sampled seed lengths and their uniqueness on hg38 using parameters $(2, 20, 5, 12)$, set as default for mapping reads of length 150nt. The blue line shows sampled seed lengths on a log scale, the green line shows the percentage of unique seeds at the given length, and the red line shows the percentage of seeds with ten or more locations. The red axis has been cut for visibility; the highest value is 0.8% for $k = 20$. The seed length is measured as the distance between the first strobe's first nucleotide and the last nucleotide of the second strobe.

quence.

Compared to the study introducing strobemers (23), we present two novel ideas. Firstly, (23) 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 while still having a similar linking range over the original sequence. Secondly, we also introduce a skewed linking approach of the second strobe that links nearby strobes more often (Suppl. Fig S1) which is more effective for shorter read lengths than the original approach (23) (details in the method section). We implement our seeding approach as the fundamental technique in a new short-read aligner strobealign.

Overview of experiments. We evaluated strobealign (v0.4) to six state-of-the-art and recently published short-read aligners, BWA-MEM (v0.7.17), Bowtie2 (v2.3.5), minimap2 (v2.22), Accel-Align (commit 01d18b2), SNAP (26) (v2.0.0), and URMAP (27) (v1.0.1480_i86linux64) on simulated and biological Illumina sequencing data sets. Our simulated data set consists of human genomes at four different divergence levels from hg38 denoted SIM1, SIM2, SIM3, and SIM4, where SIM4 has the highest divergence from hg38 (details on simulations in Suppl. Note A). We also evaluated the aligners by simulating reads with high SNP and indel rate from on a simulated repetitive genome (denoted REPEATS) consisting of five hundred 100kbp copies at roughly 90% similarity (details on simulations in Suppl. Note A). We evaluated the accuracy, runtime, and memory usage (for details on evaluation metrics, see Suppl. Note B). For the aligners that also support a mapping mode (Accel-Align, minimap2, and strobealign), we evaluated both the mapping and alignment modes (details on running aligners in Suppl. Note C). We ran all the tools using one thread to investigate alignment speed per thread and 4, 8, and 16 threads on some of the datasets to investigate how parallelization affects performance. We also evaluated concordance between aligners on biological 2*150 and 2*250 paired-end Illumina reads from HG004 in the Genome-In-a-Bottle consortium (28). We denote these datasets BIO150 and BIO250 (details in Suppl. Node D).

	Time (sec)	Peak mem (Gb)	#Seeds
AccelAlign	**134**	38.8	-
minimap2	167 (100^1, 67^2)	13.0	492.2M
Strobealign	274 (133^1, 141^2)	29.8	544.1M
URMAP	1,028	40.1	-
SNAP	1,759	45.9	-
BWA MEM	3,684	**4.5**	-
Bowtie2	6,008	5.5	-

Table 1. Time and memory generation of seeds and indexing using one thread. [1] Time reading genome from file and construct seeds. [2] Time sorting seeds and constructing index hash table.

Seeding results. We measured the time to produce our dynamic-sized seeds on hg38 (Table 1). Our seeding is fast. It takes 133 seconds to construct our seeds, comparable to producing minimizers in minimap2 (100 seconds). As for indexing the seeds, our indexing is faster than most of the other aligners (Table 1). Furthermore, the seeding is not a bottleneck in our alignment algorithm, taking up only a small fraction of the total alignment runtime across datasets (Fig 2A). Strobealign's flexible seed length allows for more unique matches of longer seeds as seen in Figure 2B). For example, fuzzy seeds of lengths 90-100n are about 2-4 times less likely than seeds of size 30-40nt to have more than ten mapping locations on hg38 (red axis), and 10-20 times less

DRAFT

likely than a seed size of 20. Strobealign's downstream candidate site selection fast, as can be seen from the time taken by find_matches and rescue_matches in (Fig 2A), as well as from the mapping runtime of our algorithm that we will present in the following sections.

Results on hg38 simulated data. For the paired-end read experiments, we present the accuracy, runtime, and percent aligned reads for the SIM3 dataset (Fig. 3), which gives a good indication of general performance across SIM1-4. The remaining three datasets, SIM1, SIM2, and SIM4, are found in Suppl. Fig S2-4. In general, strobealign, URMAP, and Accel-Align are the fastest aligners (Fig. 3C, Suppl. Fig S3). URMAP is the fastest aligner for the low diversity datasets SIM1 and SIM2, but its speed quickly reduces for higher diversity datasets and longer read lengths (Suppl. Fig S3). On the other hand, Strobealign is fast across datasets with read lengths 150-300nt, and has no substantial loss in speed due to high diversity. Furthermore, strobealign also has higher accuracy and percentage of aligned reads than URMAP and Accel-Align on most of the 20 experiments across diversity and read lengths (Fig. 3A, Suppl. Fig. S2), and, in many cases, the difference is substantial. Another notable aligner is SNAP, which has high accuracy and speed on SIM1. However, its performance across accuracy, speed, and percentage of aligned reads deteriorate substantially with increased diversity (Fig. 3, Suppl. Fig. S2). Overall, on the SIM1-4 experiments, strobealign has the best tradeoff and balance between accuracy, runtime, and percent aligned reads to any of the other benchmarked aligners. In this analysis, the closest competitor to strobealign is minimap2, which has similar accuracy and percentage of aligned reads. However, strobealign is about 2-3 times faster on all but the 100nt experiments. As for peak memory usage on hg38, SNAP has the highest memory usage at about 34Gb, strobealign and URMAP use around 29-30Gb, Accel-Align about 20Gb, minimap2 about 11.5Gb, BWA-MEM 5.5, and Bowtie 3.5Gb (Suppl. Fig. S5).

We also ran the alignment tools with 4, 8, and 16 threads on the different datasets from the SIM3 dataset. BWA-MEM, minimap2, SNAP, and strobealign nearly halves the alignment time when doubling the number of threads, suggesting that the tools utilized the resources well (Suppl. Fig. S7). Bowtie2 and Accel-Align also parallelized well, but not as well as the former three tools. We could not run URMAP in multithread mode as reported here (29) (see Suppl. Note C for all excluded experiments).

We largely observe the same results reported above on our (Suppl. Fig S8-11) for the single-end data experiments. Strobealign is the fastest tool on 14 out of the 20 datasets. Although strobealign has slightly worse performance on high diversity datasets in single-end mode (Suppl. Fig S8), it has substantially higher accuracy and percentage of aligned reads to the tools with similar speed (Accel-Align, SNAP, and URMAP). The analysis is reported in detail in Suppl. Note E.

Finally, as expected, the mapping modes of minimap2 and strobealign have lower accuracy than their respective alignment modes. However, strobealign's mapping mode has a more substantial accuracy reduction to its alignment mode than 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. However, the primary purpose of short read alignment is that alignment and mapping mode is not implemented in many aligners.

Results on the REPEATS dataset. We ran the aligners in paired-end mode on the REPEATS dataset, which constitutes a particularly challenging repetitive dataset with high diversity (described in Suppl. Note A). On this dataset, strobealign, minimap2, and BWA-MEM have the highest accuracy and percentage of aligned reads (Fig. 4). However, strobealign is 5-10 times faster than minimap2 and BWA-MEM on the 150-300nt datasets and more than two times faster on the 100nt dataset. Notably, we also observe that strobealign and minimap2 are more accurate than BWA-MEM and Bowtie2 on this dataset (Fig. 4A). We observed this improved accuracy also for most of the read lengths in the SIM4 dataset (Suppl. Fig. S2). This suggests that alignment algorithms based on subsampling seeds may be more accurate than traditional BWT approaches for high diversity datasets when run in their default modes. URMAP and SNAP have lower accuracy and substantially higher runtime compared to strobealign on all, but the 100nt reads experiment in this dataset (Fig. 4). We could not run Accel-Align in this analysis as it returned an error.

Results on biological data. We measured runtime, memory usage, and percentage of aligned reads for the BIO150 and BIO250 datasets identically to the simulated experiments. As for the accuracy, we do not have the ground truth genomic location of the biological reads. In this analysis, we used the slower but tried-and-tested tools BWA-MEM and Bowtie2 as gold standard as we observed that they aligned the most reads (Fig. 5A). We then scored the rest of the aligners based on concordance with the two tried-and-tested aligners. Specifically, we measured the concordance in alignment coordinates between three tools; BWA-MEM, Bowtie2, and each of the remaining aligners (Fig. 5B). We assume in this analysis that a high concordance with BWA-MEM and Bowtie2 is a proxy of high accuracy. If aggregating across read length on the biological datasets, we observe that strobealign is the fastest tool, has the most aligned reads, and is most consistent with BWA-MEM and Bowtie2 (Fig. 5). The only aligner out of competitors with comparable results is minimap2. We further looked at the detailed concordance Venn diagrams between BWA-MEM, Bowtie2, strobealign, and minimap2 (Suppl Fig F12). Minimap2 shares a substantial fraction of alignments with BWA-MEM and relatively few with Bowtie2, while strobealign has a more evenly distributed concordance diagram between Bowtie2 and BWA MEM. Some of the overlaps we see uniquely shared by two aligners 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 ambigu-

DRAFT

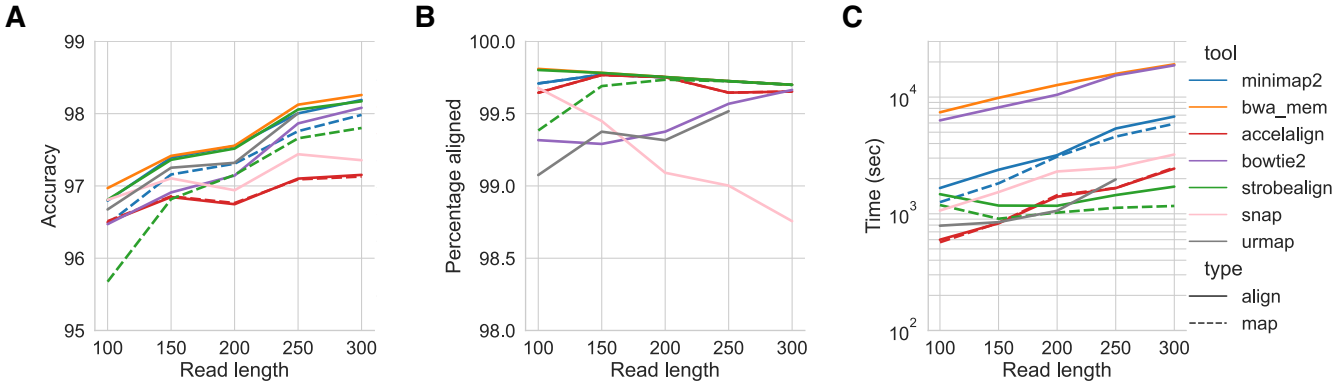

Fig. 3. Accuracy (Panel A), percent aligned reads (Panel B), and runtime (Panel C) of aligning paired-end reads to the SIM3 dataset.

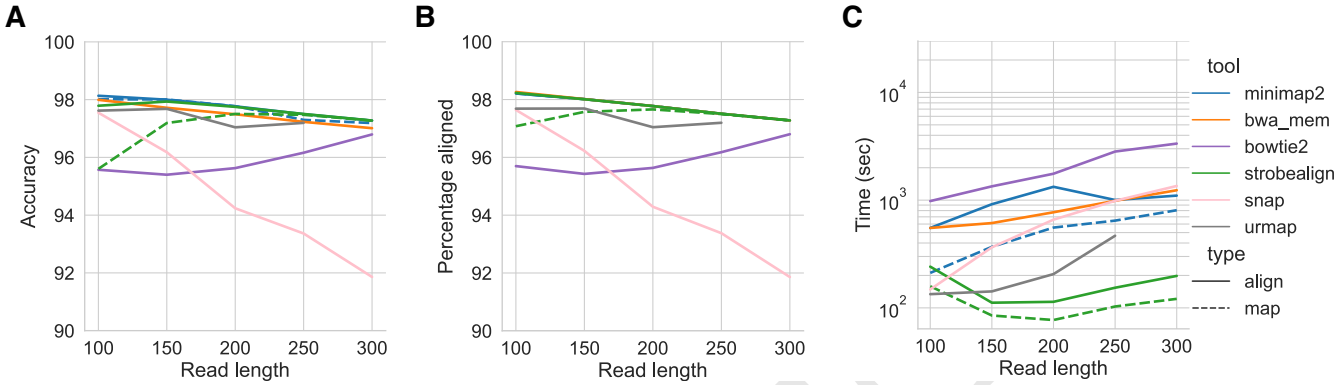

Fig. 4. Accuracy (Panel A), percent aligned reads (Panel B), and runtime (Panel C) of aligning paired-end reads to the REPEATS dataset.

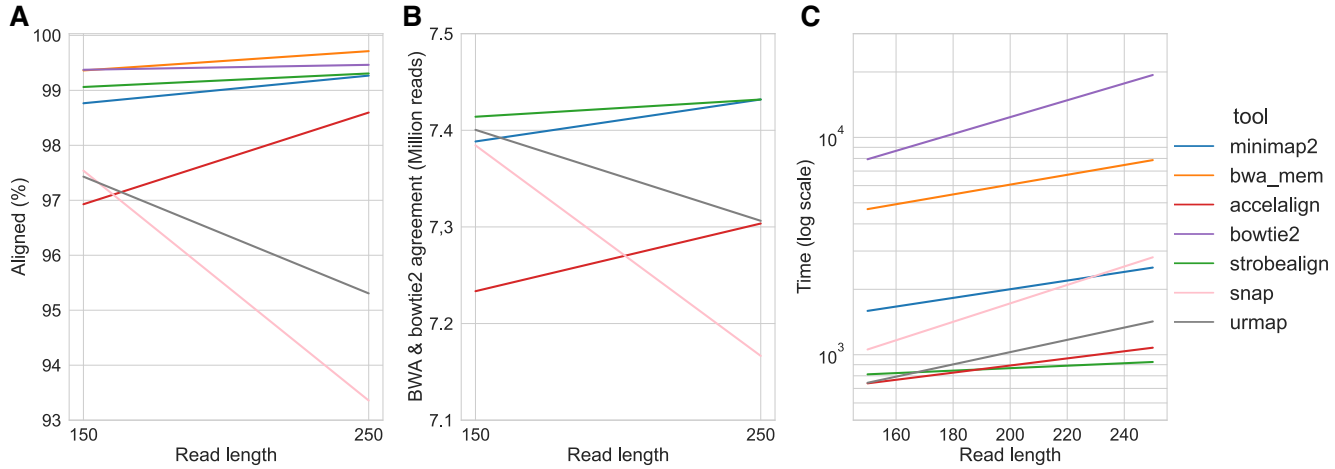

Fig. 5. Alignment results for the BIO150 and BIO250 datasets. Panel A shows the runtime, Panel B shows the percentage of aligned reads, and Panel C shows for strobealign, minimap2, Accel-Align, SNAP, and URMAP the number of reads that were aligned to the same location (overlapping alignment coordinates) out of the reads that Bowtie2 and BWA-MEM aligned to the same location (*i.e.*, three-way concordance).

ous scenarios.

Summary of evaluations. Our experiments spanned several read lengths for both biological and simulated data with different variation rates using single-end and paired-end reads. We have summarized our general observations from our experiments in Table 2. In general, we observe that strobealign is much faster than other tools in most experiments while having comparable accuracy to state-of-the-art and higher accuracy than recently proposed aligners. If ranking alternative aligners to the two established aligners BWA-MEM and Bowtie2, strobealign has the best trade-off in speed, accuracy, and number of aligned reads. The closest competitor to strobealign is minimap2. Strobealign and minimap2 have similar accuracy, but minimap2 is 2-10x slower than strobealign on all but the 2*100 paired-end datasets. Furthermore, the remaining recently proposed aligners Accel-Align, SNAP, and URMAP, either have substantially lower accuracy (Accel-Align) or do not align a substantial part of reads if variation is present (SNAP) or in repetitive locations (URMAP).

Category	Tool	%-correct ↑	%-aligned ↑	Speed ↑	Mem ↓	Comment on overall results
Traditional	BWA-MEM	High	High	Low	Low	Works well in all benchmarked scenarios. High accuracy, but slow.
	Bowie2	High	Medium	Low	Low	Works well in most benchmarked scenarios but is slow. Skips to align many reads for high diversity datasets.
Recent	Strobealign	High	High	High	High	A fast aligner with high accuracy. Peak performance for read lengths 200-300nt.
	minimap2	High	High	Medium	Medium	Works well in all benchmarked scenarios. Similar alignment-metrics to strobealign but slower for all benchmarked datasets.
	SNAP v2.0	Medium	Low	Medium to high	High	Good accuracy and speed for low diversity datasets. Poor accuracy, speed, and percent aligned reads for high diversity and on the biological datasets.
	URMAP	High	Low	High	High	Fastest aligner on low diversity datasets. Speed and accuracy reduce substantially with high diversity.
	Accel-Align	Low	Medium	High	High	A fast aligner but has the lowest accuracy across datasets.

Table 2. Overview of aligner characteristics based on results from our experiments. Brief comments on the characteristics of the aligners are included.

Discussion

We have presented a novel approach to compute seeds used for sequence mapping and implemented it as the seeding method in a short read aligner, strobealign. We demonstrate that strobealign achieves competitive accuracy and percentage of aligned reads to established popular aligners BWA-MEM and Bowtie2 (Figs. 3A, 4A, Suppl. Fig. S2) while being an order of magnitude faster (Figs. 3C, 4C, Suppl. Fig. S3). The observations made from simulated data also translate to the biological datasets (Fig. 5) where strobealign among recently proposed aligners has the fastest runtime, the most aligned reads, and the largest concordance with state-of-the-art BWA-MEM and Bowtie2. Strobealign can speed up read mapping by an order of magnitude without substantial loss in the number of aligned reads or accuracy, thus removing the bottleneck in many analysis pipelines.

The memory usage of strobealign and other recent aligners (Accel-Align, SNAP, URMAP) is relatively high compared to some memory-efficient tools such as BWA-MEM and Bowie2 (Suppl. Fig S3). However, most large sequencing datasets are aligned in multi-thread mode on computing resources with many cores and a RAM higher than 32Gb. Therefore, the memory constraint should not be of practical concern in many common bioinformatic pipelines.

Future work. A large part of the runtime for the BIO150 and BIO250 datasets is spent in base level alignments with ssw (Fig. 2). It may therefore be possible to further optimize runtime by considering a faster Smith-Waterman alignment such as the Wavefront Alignment Algorithm (30) as discussed in (21). Or find a better strategy based on the seeds to select fewer candidate sites to align to.

Our seeding method in other applications. A natural future research direction is to adapt our seeding method to other mapping scenarios. For example, applications such as long-read alignment (17, 20) or transcriptomic long-read clustering (31) may be substantially sped up when using longer, more unique seeds. Our sampling technique may also improve the computation speed of seeds for transcriptomic spliced alignment (32) as MEM finding is the current bottleneck. Another interesting direction is to explore using our seeding approach for overlap detection for genome assembly. Our seeding method can be thought of as being constructed in syncmer-space instead of over the entire sequence (sequence space). 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 (33, 34) and k consecutive minimizers (k-min-mers) (35) to represent overlaps and assembly graphs.

Conclusion

We presented a novel strategy to compute seeds based on syncmers and strobemers that can be used for candidate mapping-site detection in sequence mapping applications. We showed that our seeding is fast and we implemented our strategy in a short-read aligner strobealign. Strobealign is an order of magnitude faster than traditional aligners with comparable accuracy. Strobealign can remove the alignment bottleneck in many bioinformatic analysis pipelines and free up substantial computing resources. Furthermore, our seeding approach can potentially be used in many other applications that requires sequence mapping.

DRAFT

Methods

Definitions. 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 both the length operator applied to strings and the cardinality operator applied to sets. 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. Our fuzzy seeds produced from S are subsequences of S since they consist of two syncmers k_1 and k_2 that do not necessarily overlap on S. The syncmers are concatenated after their extraction to a string $k_{12} \doteq k_1 k_2$ that constitutes the strobemer seed.

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, each string is associated with a sequence of bits that can be interpreted as an integer. For example, $TCA = 110100 = 52$. From now on, we will assume that all strings are manipulated by manipulating their sequence of bits. We use h to denote a hash function $h:\{0,1\}^* \rightarrow \{0,1\}^*$ mapping a sequence of bits to another sequence of bits. Finally, we say that two seeds k_{12} and k'_{12} match if $h(k_{12}) = h(k'_{12})$. 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.

Open syncmers. Open syncmers is a k-mer subsampling method described in (22). They are sampled based on three parameters (k,s,t). 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. Similar 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 are described in (23) and consists of several shorter k-mers, referred to as *strobes*. We use the randstrobe method (23) described below. Four parameters (n, k, w_{min}, w_{max}) are used to define a randstrobe where n is the number of strobes, k is the length of the strobes, and w_{min} and w_{max} is the lower and upper coordinate offset to the first strobe k_1 for sampling the second strobe k_2 on a string S. We will here consider randstrobes with $n=2$, *i.e.*, consists of two strobes. Let k_1 have coordinate i on S and $\mathcal{W}$ be the set of k-mers from the substring (window) $S[i+w_{min} : i+w_{max}+k-1]$. Then, the randstrobe samples strobe k_2 according to the following sampling function (23):

$$k_2 = \underset{k' \in \mathcal{W}}{\operatorname{argmin}}\ (h(k_1) + h(k')) \,\&\, q_p. \qquad \textbf{(1)}$$

where & is the bitwise AND operator and q_p is a bitmask consisting of 1's at the p leftmost bits and remaining 0's.

Modifications to strobemers. Strobemers was first described as being produced over the set of all k-mers in a sequence (23). The first modification we make to strobemers as originally described is that we will compute the strobemers over syncmers. Therefore, from now on k_1, k_2, and k' are used to denote the subsampled set of k-mers that are open syncmers. We will let $[w_{min}, w_{max}]$ refer to the lower and upper number of syncmers downstream from k_1 where we sample k_2 from. Assume that k_1 has coordinate i, then we let $\mathcal{W}_s$ denote the set of syncmers in the substring $S[i+w_{min} : i+w_{max}+k-1]$.

A second modification to the strobemers (23) is that we use a skewed sampling function that selects nearby syncmers more frequently. The sampling skew for sampling the second syncmer k_2 is produced from

$$k_2 = \underset{k' \in \mathcal{W}}{\operatorname{argmin}}\ B(\,(h(k_1) \wedge h(k')) \,\&\, q_p). \qquad \textbf{(2)}$$

Here, $\wedge$ is the bitwise XOR operator, and B counts the number of set bits. In other words, B returns the number of set bits among the $p \in [1,64]$ leftmost bits in the 64-bit integer produced after the XOR operation of the hash values of the two strings. Function 2 maps the value space down to [0,p-1] and collisions are resolved by picking the first leftmost strobe. Therefore, a lower value on p results in more often picking nearby strobes. An example distribution is shown in Suppl fig. S1. We found that function 2 gave significantly improved performance on shorter reads (100-150nt) compared to function 1.

The third modification to strobemers is our stored hash value for the strobemer. We store for k_{12} the value $h'(k_{12}) \doteq h(k_1)/2 + h(k_2)/2$. The hash function h' is symmetric $(h'(k_{12}) = h'(k_{21}))$ 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 (23) that symmetrical hash functions are undesirable for mapping due to unnecessary hash collisions. However, when masking highly repetitive seeds as commonly performed in aligners (18), 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 $h''(k_{12}) = h(k_1)/2 + h(k_2)/3$ proposed in (23). Also assume that strobemer seeds k_{12} and k_{21} are both found in forward orientation the reference due to, *e.g.*, inversions. In this case, only k_{12} may be masked because of its distinct hash value to k_{21}. Now, consider a read in which we extract k_{12} in forward direction and k_{21} in reverse complement direction. If the read has an optimal match to forward direction with seed k_{12} (masked on reference), we would still find the suboptimal match of k_{21} 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.

Such scenarios can be overcome by indexing and masking the reference in both orientations, but it introduces a lot of

DRAFT

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 masking seeds consistently across the two sequence directions is beneficial. Still, there is room for future investigation on handling such cases.

Indexing. We first construct open syncmers from the reference sequences and then link two open syncmers together using the randstrobe method with equation 2 as sampling function. A beneficial characteristic with open 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 $t = \lceil (k-s+1)/2 \rceil$ is the optimal parameter for sequence mapping (24). 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 set w_{min} and w_{max} dependant on the read length based on an experimental evaluation. See details in Section Implementation details.

We store tuples $(h'(k_{12}), r_s, v)$ in a flat vector V where $h'(k_{12})$ is the 64-bit integer hash value of the strobemer, r_s is the coordinate start of the first strobe (32-bit integer), and v is a 32-bit integer containing reference id (rightmost 24 bits) and the offset of the second strobe (leftmost 8 bits). We sort V by hash values and construct a hash table with hash values as keys pointing to offset and the number of occurrences of the hash value in V. By lookup of $h'(k_{12})$, we know which segment in V to iterate over to find query matches. This type of index representation has been used previously (18, 21), and was suggested to us by Heng Li (36). 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. Strobealign computes canonical open syncmers from the read similarly to what is described above to index the reference. Since the created syncmers are canonical, we can compute forward and reverse complemented strobemers by iterating over the syncmers forward and reverse order, respectively. Computing strobemers in forward and reverse orientation gives us a vector of tuples (q_s, q_e, o) representing start coordinate q_s, end coordinate q_e, and orientation o (boolean value with 0 representing forward and 1 representing reverse complement), 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 $m = (r_{id}, r_s, r_e, q_s, q_e, o)$ a *match* between the query and the reference. Let d be the difference in length between the strobemer on the reference and query. If several matches are found, Strobealign iterates over the matches, stores the lowest observed d during iteration, and saves only the matches with the current lowest d. This approach is not the same as first computing the lowest d and iterating a second pass to store only matches with the lowest d. We chose the former as we tried both and observed close to no difference in accuracy while slightly increasing runtime in the latter case due to the extra iteration.

From the stored matches, strobealign constructs *merged matches* $\mathcal{M}$ which are similar but slightly more stringent to Non-overlapping Approximate Matches (NAMs) that are defined in (23). 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) $m.r_s < m'.r_s \leq m.r_e$

(iv) and 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 strobemer matches need to have a 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 of how many matches were added, the position of the first and last strobe on query and reference, and the orientation on the reference genome. After the final match on the query, we close all merged matches. The closed matches are the final merged matches that constitute 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 (18) to our merged match score $S_{\mathcal{M}}$. That is, if S_1 and S_2 are the top two scoring merged matches for a read, the MAPQ is computed by

DRAFT

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

Single-end Alignment. Merged matches are produced and scored as described above and constitute 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 merged match is different on the reference and query due to, *e.g.*, indels, we send the sequences to alignment with ssw (37). We use a match score of 1 and alignment penalties of 4, 6, and 1 for mismatch, gap open, and gap extension, respectively. Additionally, if the computed hamming distance is larger than $0.05|r|$ where $|r|$ denotes the read length, we perform an additional alignment with ssw 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 could have few or zero matches if all the strobemers extracted from the read were masked due to being too abundant. The abundance cutoff, which we denote as A, is controlled with a parameter -f (default value 0.0002), as in minimap2. For example, A is between 30 and 50 for hg38 depending on the values we use for parameters k, w_{min}, and w_{max}, described below in Implementation details. If more than 30% or strobemers were masked when finding matches, strobealign enters a rescue mode where it considers 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 produces 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. Similar to the single-end mapping mode, strobealign computes merged matches for both mates within the read pair and employs an identical rescue mode if there are 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, based on the mate's mapping location, strobealign can enter a rescue mode even for a read with zero matches. We describe the two components below.

For the joint scoring, strobealign first sorts the candidate map locations based on the total seed count for the two mates in a read pair. Then, strobealign finds the best candidate locations from a combined MAPQ score described below. 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 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 without merged match within a genomic segment of $[0, \mu + 5\sigma]$ nucleotides away in the expected direction from the location of the mate with a merged match.

Implementation details. Similar to the default parameters in minimap2, we consider the top 20 MAPQ (or joint MAPQ) scoring candidates for alignment, and we implement a drop-off score threshold of 0.5 (score to the highest score). In addition to these parameters, 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 an edit distance of 1. In that case, we do not call base-level alignment for remaining candidates since a call to base level alignment implies that the edit distance is at least 1, as we described in the single-end alignment section. Stobealign also supports multithreading using openMP. If more than one thread is specified, strobealign will parallelize the alignment step by splitting the reads into batches of 1 million reads to be processed in parallel.

As for selecting values of k, p, w_{min} and w_{max}, we set suitable parameters based on experimental evaluation of accuracy and runtime. Naturally, w_{min} and w_{max} was selected higher for longer read lengths $|q|$. We let $w_{min} = k/(k-s+1) + l$ and $w_{min} = k/(k-s+1) + u$ where l and u are integers that specify lower and upper offset. We let $(k, p, l, u) = (20, 8, -2, 2)$ for $|q| < 125$, $(20, 8, 1, 7)$ for $125 < |q| \leq 175$, $(20, 8, 4, 13)$ for $175 < |q| \leq 275$, and $(22, 2, 12)$ for $|q| > 275$. With a read length of length 200nt and the parameters in this study, roughly $(1/5) * 200 = 40$ syncmers are produced for each read, and roughly 30 strobemers are created in each direction. Naturally, these values

DRAFT

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 at the cost of memory.

Although it is exponentially less likely to have open syncmers sampled further away from the mean sampling density (22), they do not have a window guarantee and may be sparser sampled in some regions. Therefore, we have a hard limit on the maximum seed size as a parameter to strobealign (m), where it defaults to $|q| - 50$. This means that the maximum window offset of sample the second strobe will come earlier than w_{max}. With the parameters we use above, this happens in less than 0.1% of the seeds. Finally, on rare occasions there is no open syncmer in the downstream window within distance $m - k$, *e.g.*, due to regions of N's in centromeres. In these cases we use only the first syncmer as the strobemer seed. This happens for 0.0007% of the seeds (3,963 out of 544M seeds) on hg38.

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. Mohammed Alser, Jeremy Rotman, Dhrithi Deshpande, Kodi Taraszka, Huwenbo Shi, Pelin Icer Baykal, Harry Taegyun Yang, Victor Xue, Sergey Knyazev, Benjamin D. Singer, Brunilda Balliu, David Koslicki, Pavel Skums, Alex Zelikovsky, Can Alkan, Onur Mutlu, and Serghei Mangul. Technology dictates algorithms: recent developments in read alignment. *Genome Biology*, 22(1):249, 2021. doi: 10.1186/s13059-021-02443-7.
11. 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.
12. 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.
13. 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.
14. 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.
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. 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.
18. Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18): 3094–3100, 05 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty191.
19. 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.
20. 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.
21. 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.
22. Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, 2021. ISSN 2167-8359 (Print); 2167-8359 (Electronic); 2167-8359 (Linking). doi: 10.7717/peerj.10805.
23. Kristoffer Sahlin. Effective sequence similarity detection with strobemers. *Genome Research*, 31(11):2080–2094, 2021. doi: 10.1101/gr.275648.121.
24. Jim Shaw and Yun William Yu. Theory of local k-mer selection with applications to long-read alignment. *Bioinformatics*, 11 2021. ISSN 1367-4803. doi: 10.1093/bioinformatics/btab790. btab790.
25. Inc. Illumina. https://www.illumina.com/company/news-center/press-releases/press-release-details.html?newsid=e3ad6d4e-fcbe-4a0d-ac1d-5a3996d1bfd8, 2022.
26. William J. Bolosky, Arun Subramaniyan, Matei Zaharia, Ravi Pandya, Taylor Sittler, and David Patterson. Fuzzy set intersection based paired-end short-read alignment. *bioRxiv*, 2021. doi: 10.1101/2021.11.23.469039.
27. Robert Edgar. Urmap, an ultra-fast read mapper. *PeerJ*, 8:e9338, 2020. ISSN 2167-8359 (Print); 2167-8359 (Electronic); 2167-8359 (Linking). doi: 10.7717/peerj.9338.
28. 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.
29. Krisoffer Sahlin. Multithreading. https://github.com/rcedgar/urmap/issues/8, 2021.
30. 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.
31. 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.
32. 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.
33. Chen-Shan Chin and Asif Khalak. Human genome assembly in 100 minutes. *bioRxiv*, 2019. doi: 10.1101/705616.
34. 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.
35. 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.
36. Heng Li. Code examples of fast and simple k-mer counters for tutorial purposes. https://github.com/lh3/kmer-cnt/issues/4, 2020.
37. Mengyao Zhao, Wan-Ping Lee, Erik P. Garrison, and Gabor T. Marth. Ssw library: An simd smith-waterman c/c++ library for use in genomic applications. *PLOS ONE*, 8(12):null, 12 2013. doi: 10.1371/journal.pone.0082138.

DRAFT