Multi-context seeds enable fast and high-accuracy read mapping

Ivan Tolstoganov[1], Marcel Martin[2], Kristoffer Sahlin[1,*]

[1] Department of Mathematics, Science for Life Laboratory, Stockholm University, 106 91, Stockholm, Sweden
[2] Department of Biochemistry and Biophysics, National Bioinformatics Infrastructure Sweden, Science for Life Laboratory, Stockholm University, Box 1031, SE-17121 Solna, Sweden
*ksahlin@math.su.se

Abstract. A key step in sequence similarity search is to identify seeds that are found in both the query and the reference sequence. A seed is a shorter substring (e.g., a k-mer) or pattern (e.g., a spaced k-mer) constructed from the sequences. A well-known trade-off in applications such as read mapping is that longer seeds offer fast searches through fewer spurious matches but lower sensitivity in variable regions as longer seeds are more likely to harbor mutations. Some recent developments on seed constructs have considered approximate (or fuzzy) seeds such as k-min-mers, strobemers, BLEND, SubSeqHash, TensorSketch, and more, that can match over smaller mutations and, thus, suffer less from sensitivity issues in variable regions. Nevertheless, the sensitivity-to-speed trade-off still exists for such constructs. In other applications, such as genome assembly, using multiple sizes of k-mers is effective. While this can be achieved in read mapping through, e.g., MEM construction from an FM-index, such seed constructs are typically much slower than hash-based constructs.

To this end, we introduce multi-context seeds (MCS). In brief, MCS are strobemers where the hashes of individual strobes are partitioned in the hash value representing the seed. Such partitioning enables a cache-friendly approach to search for both full and partial matches of a subset of strobes. For example, both the full strobemer and the first strobe (a k-mer) can be queried. We demonstrate that MCS improves sequence matching statistics over standard strobemers and k-mers without compromising seed uniqueness. We demonstrate the practical applicability of MCS by implementing them in strobealign. Strobealign with MCS comes at no cost in memory and only little cost in runtime while offering increased mapping accuracy over default strobealign using simulated Illumina reads across genomes of various complexity. We also show that strobealign with MCS outperforms minimap2 in short-read mapping and is comparable to BWA-MEM in accuracy in high-variability sequences. MCS provides a fast seed alternative that addresses the trade-offs between seed length and alignment accuracy.

Introduction

In bioinformatics, a seed refers to a short substring or subsequence of a biological sequence that is used to initiate a sequence similarity search between sequences, e.g., between a read and a reference genome. Seeds are crucial for heuristic (i.e., not guaranteed to find the optimal solution) algorithms for sequence comparison such as read mapping algorithms [2, 36]. In the seeding step, the seeds help to quickly identify potential *matches*, reducing the search space. A match (also sometimes referred to as an anchor) is a seed that occurs both in the query and the reference.

A k-mer is a string of length k and is commonly used in sequence similarity search algorithms. One such common application is read mapping. In read mapping, the choice of k matters: Longer k-mers enhance seed specificity and therefore mapping speed, but shorter k-mers increase sensitivity as they are less affected by sequencing errors and biological mutations. Several alternative methods have been proposed to enable using longer seeds that can match in the regions between mismatches and indels. These include spaced seeds [27], TensorSketching [16], context-aware seeds [40], k-min-mers [10], strobemers [34], BLEND [13], and SubSeqHash [26, 25]. Some of these seeds were shown to be efficient for short [35] and long-read mapping [11, 13, 16] and can enable one or two orders of magnitude speedup over using k-mers [11, 35, 13, 16]. At a high level, the seeds in [11, 35, 13] are based on joining together several k-mers into a single seed, thus providing a longer range of a match, which gives more unique seeds and, thus, less candidate locations to explore. However, long seeds, although they allow approximate matches, come with an undesirable reduction in accuracy, which makes more resource-expensive but accurate traditional alternatives the most popularly used tools [21, 23, 24, 38].

The trade-off between seed length and accuracy makes the idea of having seeds at multiple levels or resolutions compelling. The longer seeds would provide speed through their low match redundancy, while one could fall back on shorter seeds in case no longer matches were found. The concept of using different k-mer sizes for various resolutions has been used in, e.g., genome assembly [31, 4, 3], transcriptome assembly [33], RNA-Seq analysis [18], and k-mer counting [17]. In read mapping, variable-length k-mer-based seeds such as MEMs [7, 23] and MCAS [15] are used, but they are typically slow to query. This is because the query process to construct such seeds needs to access scattered locations in memory. For example, the FM-index [12] is used to produce MEMs. However, aforementioned multi-resolution k-mer methods only increase or decrease the resolution with one nucleotide [23] or store multiple sizes of k [4], which is memory consuming.

Our contribution

We propose using strobemers that support queries at multiple resolutions. A strobemer consists of $n \geq 2$ k-mers, and with our idea, it is possible to query all or a certain subset of these k-mers in the index. Naively, we could store both k-mers and strobemers with $2, \ldots n$ k-mers in the same index. However, this would increase space requirements and would also slow down mapping through cache misses if the seeds are scattered in memory. Instead, we present *multi-context seeds* (MCS). MCS are strobemers produced with the randstrobe method [34] that distribute the bits allocated to store the (hash of the) seed separately between k-mers in the strobemer. While some technical details are needed for MCS to mimic the canonicity of k-mers, the principle idea of MCS is as follows. For an N bits allocation of a seed hash value, an MCS will store the leftmost $k_1 < N$ bits to represent the first k-mer in the strobemer, the following $k_2 < N$ bits for the second k-mer, etc., with $k_1 + k_2 + \ldots = N$. See Fig. 1 for a conceptual illustration. When indexed efficiently, for example as a vector sorted by hash values paired with a look-up table, as is standard in k-mer based mapping [24, 42, 35], this seed enables cache-efficient lookup for both *full matches* covering the whole strobemer and *partial matches* covering the first several strobes.

Being able to search for seed matches at several resolutions compared to the binary "match or no match" of the full seed increases both accuracy and percentage of aligned reads. We first demonstrate that MCS improve sequence matching characteristics over randstrobes and k-mers but retain roughly the same seed uniqueness benefits as randstrobes. We then show the practical applicability of MCS by implementing them in strobealign [35], which uses randstrobes as default. We benchmark strobealign with MCS and compare it to other state-of-the-art alignment tools using simulated Illumina reads of different lengths from four genomes at various sizes and sequence complexities. Strobealign with MCS improves mapping-only and extension-based read alignment accuracy over default strobealign in all experiments with typical Illumina

read lengths ≤ 150 nt for both single-end and paired-end data, without a substantial slowdown. Particularly, as sequence variability gets high, MCS have a higher relative increase in accuracy over default strobealign. Unlike default strobealign, strobealign with MCS now surpasses minimap2 accuracy on all experiments while also being substantially faster. When compared with the gold standard BWA-MEM, strobealign with MCS has comparable accuracy on datasets with high diversity, but with much lower mapping and indexing time.

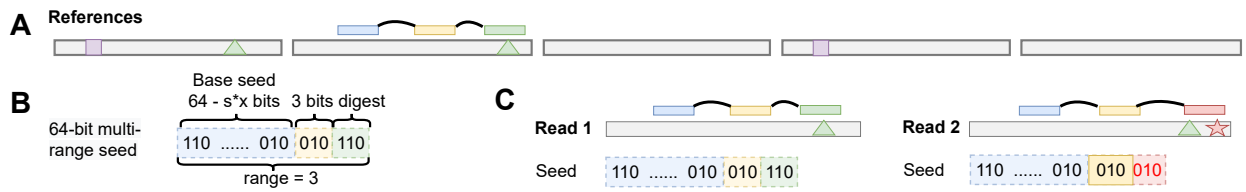

Fig. 1. Multi-context seeds concept. **A** The reference sequence consisting of five similar copies with some variation (colored shapes), and a strobemer seed sampled from one of the copies. **B**. The forward multi-context seed representation of the strobemer consisting of 58-bit prefix for the base strobe, and 3-bit prefixes for the remaining strobes. **C** Read mapping of a read without sequencing errors (read 1) and a read with a sequencing error or non-reference mutation in the third strobe (read 2). While read 2 cannot produce a full match, the part of the seed limited to the first two strobes is still able to produce a match to the reference.

Methods

Preliminaries

Notation Let R be a string over the alphabet $\Sigma = \{A, C, G, T\}$. We refer to a string obtained by deleting a subset of characters in R as a *subsequence* of R. We define the consecutive subsequence as a *substring*. We will use zero-indexed notation for strings. Given nucleotide string R, a *k-mer* is defined as a substring of R of length k. Let $R[i, k]$ denote a *k-mer* covering the positions from i to $i + k - 1$. We use h_n to denote a hash function that maps k-mers to n-bit strings $h_n : \Sigma^k \to \{0, 1\}^n$. The k-bit prefix of the n-bit hash h_n is denoted as h_n^k.

Strobemers Strobemers were introduced in [34] as a way of handling indels and reads with a high mutation rate. Strobemers are constructed from an array $K = (s_1, \ldots, s_N)$ of N k-mers in a string. Given K and parameters $(n, m, w_{\min}, w_{\max})$, a *strobemer* of order n is defined as a subsequence of n k-mers $t = (s_{i_1}, s_{i_2}, \ldots, s_{i_n})$ in K. The $w_{\min}$ and $w_{\max}$ parameters limit the distance between consecutive k-mers in t, i.e. for every $j \in [1, n-1]$, we have $i_{j+1} \in [i_j + w_{\min}, i_j + w_{\max}]$. The k-mers selected for a strobemer are called *strobes*.

Strobemers as initially described were constructed from complete k-mer arrays K of a reference. However, strobemers can also be constructed from a subset of k-mers. For example, in strobealign [35], strobemers are constructed from the array of *open syncmers* [9] generated in the reference. In this work, we abstract over the array of k-mers from which a strobe is chosen by denoting $\mathcal{W}_j$ to be the indices of k-mers available in a window where strobe j is selected. When selecting strobes from all k-mers, the window corresponds to the consecutive indices of a substring of the reference of length $w_{\max} - w_{\min} + 1$, while it is a subsequence of reference indices with larger span if selecting over a subset of k-mers such as minimizers [37, 32, 29] or syncmers [9].

Randstrobes Various strategies for selecting the next strobe were described in [34, 28]. Randstrobes are constructed iteratively from the starting strobe s_{i_1}. If strobes $s_{i_1}, \ldots, s_{i_j}$ were already selected for a randstrobe, the strobe $s_{i_{j+1}}$ is selected as

$$s_{i_{j+1}} = \underset{\ell \in \mathcal{W}_{j+1}}{\operatorname{argmin}} f(s_\ell, s_{i_1}, \ldots, s_{i_j}), \tag{1}$$

where f is a function that maps a strobe to an integer. Several functions were investigated in [19]. For example, strobealign uses

$$s_{i_{j+1}} = \operatorname*{argmin}_{\ell \in \mathcal{W}_{j+1}} B\big((h(s_\ell) \oplus h(s_{i_1}) \oplus \cdots \oplus h(s_{i_j}) \ \&\ P\big), \tag{2}$$

where h is a hash function from a sequence to a 64-bit number, B is the number of set bits, P is a bitmask with its p least significant bits set, and $\oplus$ is the bitwise XOR operator. The B and p make strobealign pick strobes earlier in the window more often, which is deemed suitable for the shortest reads (for details, see [35]). To combine the hashes of individual strobes into a single hash value for the randstrobe, strobealign uses the symmetric ($h(a,b) = h(b,a)$) hash function $h(s_{i_1}, \ldots, s_{i_n}) = \sum_j h(s_{i_j})$, to hash strobes from two different orientations into the same hash value, as described in [35].

Multi-context seeds

We introduce multi-context seeds, a seeding strategy based on the strobemers [34] technique. Strobemers consist of several k-mers, or *strobes*, such that the distance between two consecutive k-mers does not exceed a constant parameter. While this strategy allows to effectively increase the length of the seed, it results in lower sensitivity, especially in shorter reads. To alleviate this, we change the strobemer hash function to a concatenation of prefixes of individual strobe hashes. This hash function allows searching for the first consecutive strobes instead of the whole strobemer by using a prefix of the hash in a search, as illustrated in Figure 1. Effectively, this means that for strobe length s, seeds of lengths $s, 2s, \ldots, ns$ are stored simultaneously in a single index. A formal description of multi-context seeds and their implementation in strobealign [35] follows.

Orientation-specific multi-context seeds Given integers N, n, and $k_1, \ldots, k_n$ such that $\sum_{i=1}^{n} k_i = N$, the *forward multi-context* seed (forward MCS) is a strobemer consisting of strobes $s_{i_1}, \ldots, s_{i_n}$, which is stored as

$$h(s_{i_1}, \ldots, s_{i_n}) = \overline{h_N^{k_1}(s_{i_1}) h_N^{k_2}(s_{i_2}) \ldots h_N^{k_n}(s_{i_N})}$$

where $\overline{t_1 \ldots t_n}$ denotes the concatenation of bit strings $t_1, \ldots, t_n$.

For example

$$h(s_{i_1}, s_{i_2}) = \overline{h_{64}^{40}(s_{i_1}) h_{64}^{24}(s_{i_2})}$$

describes a strobemer with two strobes hashed in a 64-bit integer where the first strobe is represented with 40 bits and the second strobe with 24 bits. For an illustration, see Fig. 1. We say that the first k_1 bits of the MCS hash are the *base part* of the hash, and the following allocations are (shorter) *digests*. Typically these are given more bits than downstream strobes to avoid hash collisions of the base k-mer.

Multi-context seeds and symmetric hashes for canonicity Forward multi-context seeds are not canonical, i.e. the forward multi-context hashes of a strobemer and its reverse complement are not guaranteed to be the same. This is both because (i) the strobe hash prefixes are stored in order, and (ii) because if s_{i_1} is paired up with s_{i_2} in the forward direction, it does not guarantee that s_{i_2} pairs up with s_{i_2} in the reverse complement direction. We have not found any computationally efficient way to make MCS fully canonical. However, it is still likely that a pair of strobes s_{i_1}, s_{i_2} will form a strobemer in both directions. To avoid mapping bias between orientations (see [35]) we want the hash function to be symmetrical on such pairs. Therefore, we describe an adjustment to the forward MCS construction above which removes the differing hash prefix order for seeds in different orientations. Namely, we let the MCS for $s_{i_1}, \ldots, s_{i_n}$ be stored using one of the two hash functions

$$h_1(s_{i_1}, \ldots, s_{i_n}) = \overline{h_N^{k_1}(s_{i_1}) h_N^{k_2}(s_{i_2}) \ldots h_N^{k_n}(s_{i_n})},$$
$$h_2(s_{i_1}, \ldots, s_{i_n}) = \overline{h_N^{k_1}(s_{i_n}) h_N^{k_2}(s_{i_{n-1}}) \ldots h_N^{k_n}(s_{i_1})}$$

The choice between the *forward* hash function h_1 and the *reverse* hash function h_2 is made based on the comparison between the hashes of the first and the last strobe, resulting in the multi-context hash function h_{mcs}. Specifically,

$$h_{\text{mcs}}(s_{i_1}, \ldots, s_{i_N}) = \begin{cases} h_1(s_{i_1}, \ldots, s_{i_N}), & \text{if } h_N(s_1) < h_N(s_N) \\ h_2(s_{i_1}, \ldots, s_{i_N}), & \text{otherwise.} \end{cases} \tag{3}$$

We let Eq. 3 be the construction of an MCS that we benchmark in this paper. We refer to this construction as using a symmetric hash function (for $n \leq 3$). The concept is similar to the construction of symmetric randstrobes in strobealign, which uses $h(x, y) = x + y$) for symmetry instead of the originally proposed $h(x, y) = 2x - y$) as discussed in [19]. Note that neither MCS nor randstrobes with symmetric hash function are canonical due to (ii) above. Note also that only s_{i_1} and s_{i_N} change order in Eq. 3. This is however enough for strobemers consisting of two or three strobes to be symmetrical, which are the only strobemer constructions we investigate here, as higher order strobes have shown no additional benefit in the applications we investigated [28]. In our implementation, MCS with a symmetric hash function and two or three strobes require one extra bit to store which of the strobes is the base hash, which is necessary for computing the start position of the strobe.

Implementation in strobealign

Open syncmers Open syncmers is a k-mer subsampling method [9]. Given positive integer parameters (k, s, t), where $s \leq k$ and $t \in [0, k - s + 1]$, a k-mer is selected as an *open syncmer* iff the smallest of its $k - s + 1$ consecutive s-mers occurs at position t. The smallest s-mer is defined based on a hash function. In order to process queries both from the forward and the reverse strand, we store a canonical representation of the open syncmers, i.e., only the smallest syncmer out of the forward and the reverse-complement s-mers.

Multi-context seeds in strobealign As in the randstrobe version of strobealign, we use the canonical open syncmers in order to create the sequence of strobes for multi-context seed construction. We construct open syncmers using the parameters k, s, and $t = \lceil (k - s + 1)/2 \rceil$, where parameters k and s are dependant on the read length (see details in the Supplementary Section 1). Depending on the value of t, $14 - 20\%$ of k-mers are selected as strobes. Following the construction of the sequence of open syncmers $S = (s_1, \ldots, s_N)$ from the input references, we employ the *skewed randstrobe* selection strategy described in strobealign [35] for both randstrobe and multi-context seed constructs. Every syncmer s_i from S is used once as the first strobe in the seed construction, with the subsequent strobes chosen using Eq. 2. Both randstrobes and MCS are constructed as described above in the Randstrobes section, with the parameters $w_{\min}$ and $w_{\max}$ chosen based on read length. We take the bit to represent which strobe is the base hash value from the number of reference identifiers that strobealign can hold (now 2^{23} instead of 2^{24}). However, this bit could also be taken from one of the 64 bits for the hash value.

Indexing The individual MCS are stored as tuples $V_i = (h_i, p_i, o_i, r_i, f_i)$, where h_i is the MCS hash $h(t_1, \ldots, t_n)$, p_i is the starting position of the first strobe in the reference sequence, o_i is the offset of the last $n - 1$ strobes (i.e. the distance between the start of the current strobe and the start of the previous strobe), r_i is the reference index, and f_i is a flag specifying which hash function, forward or reverse (h_1 or h_2 in Eq. 3), was used to store the MCS. This flag allows the hash function h_{mcs} to be symmetrical with respect to the order of strobes. In memory, V_i takes up 16 bytes: 8 bytes for h_i, 4 bytes for p_i, 1 byte for o_i, 23 bits for r_i and 1 bit for f_i (default strobealign uses 24 bits for r_i as it does not need f_i).

The V_i tuples are stored in a vector V accompanied by an indexing structure *AEMB index*, described in [1] (forthcoming study). We describe it briefly here for self-containment. Given vector V of tuples V_i sorted in increasing order by h_i values, and a parameter $b < 64$, the AEMB index defines an array I_b, which stores in position j the first occurrence position i in V such that a bit representation of j is a b-prefix of i. If there is no tuple $(h_i, p_i, o_i, r_i, f_i)$ in V such that j is a b-prefix of h_i, a value from the preceding position $I_b[j - 1]$ is stored instead. We define $I_b[0]$ as 0, so a preceding value always exists. Since V is sorted by hash values h_i, hashes in V with the same b-prefixes are stored in a single consecutive block. I_b is sometimes referred to as a prefix-lookup vector and has been used in other sequence mapping algorithms [8].

If the hash h is queried in V, the first position in the block in V is the value $I_b[h^b]$. The last position of the block is the minimum value in I_b which follows h^b and is different from $I_b[h^b]$. The block in V can then be searched for the exact match h using either linear or binary search.

Querying Our implementation of strobealign with MCS contains all the same algorithm steps as strobealign, with the difference that MCS are used to search for partial matches in case a full match was not found. In more detail, for a query (read), strobealign [35] searches for seed matches in the index, merges the matches into a modified version of Non-overlapping Approximate Matches (NAMs) [34], and performs an extension alignment step. The seeds are constructed from the forward and the reverse complemented read sequence in tuples V_i. The randstrobe version of strobealign differs from the multi-context version in selecting matches from the constructed seeds. In the randstrobe version, the seed is considered a match if its hash is present in the index and if it is not hard masked (i.e. occurs less than 1000 times in the index). In the multi-context version, a seed can be either a full match or a partial match if the full hash value or only the base hash value is present in the index, respectively. The full match criteria are the same as for the randstrobe seed. If a query is not a full match, we search for a base part of a query hash in the index if it was not searched previously. If the base part of a hash is present in the index and is not hard masked, we select it as a partial match. The query and reference positions of the partial match depend on whether the first or the second strobe of the seed is base, and on the orientation of the strobe. Pseudocode for this procedure can be found in Supplementary Section 1.

Note that since the index V is sorted by the hash value, the prefix of the queried hash can also be searched in the index using the same algorithm as for the full hash search, as long as the prefix is longer than the parameter b. For the MCSs, we use this property of the AEMB index to search for the *partial matches*, entries in V with the same base part of the hash as the query. Thus, looking up the hash of the base (search for partial hit) is cheap as the part of the vector is often in the cache after the full seed has been searched. Similarly, nearby seeds may share the same base hash value, further reducing cache misses.

Bit allocation of the base and the digests

We want to allocate a fraction of bits to represent the base hash value such that it minimizes hash collisions of both full and partial searches. The optimal bit allocation depends on the underlying genome repeat structure, size of k, and the number of available bits B. However, since the repeat structure is unknown, we aimed to select a reasonable bit partition based on simulated experiments. We computed the fraction of bits allocated to the base hash that minimized the sum of E-hits [35] for full and partial matches with two strobes, for various combinations of k, $w = |\mathcal{W}|$, B, given a genome size $G = 2^{16}$. We investigate both a fully random genome and one with a simulated repeat structure. The details of the experiment are found in Suppl. Section 1. In brief, we found that when B is small compared to G, we want to allocate more bits to the base. At the same time, for relatively large B, the fraction tends towards 0.5 for both genomes (Suppl. Fig. 1). We also observed that the value of k in relation to G matters since it affects k-mer uniqueness. Finally, w did not have an impact on the optimal fraction. We use $B = 64$ in all our benchmarks (and in strobealign), which is sufficient for most genomic databases, and therefore picked a fraction 40/64 fairly close to 0.5, i.e., we use 40 bits for the base hash. The remaining 24 bits are allocated to the digest of the second strobe in 2-order MCS, and 12 bits are allocated to the digests of the second and the third strobe in 3-order MCS.

Results

Evaluation overview

We first compare multi-context seeds with randstrobes and k-mers, taking into account sequence matching coverage to assess sensitivity and seed uniqueness to assess specificity. This is a general-purpose sequence similarity search benchmark to assess the repetitiveness of seeds and match sensitivity in a general setting. We then implement multi-context seeds in place of randstrobes in the short-read mapping tool strobealign [35] and benchmark the multi-context version of strobealign (commit `022a721`) against the latest randstrobe version of strobealign (v0.14.0), BWA-MEM (v0.7.17) [23], and minimap2 (v2.28) [24] using simulated and

biological short-read data. We compare against BWA-MEM and minimap2 as they were the competitors with the highest accuracy (BWA-MEM) and most desirable speed to accuracy trade-off (minimap2) to strobealign in a previous benchmark [35] that included other tools such as Bowtie2 [21], SNAP2 [6], and Accel-Align [42].

Datasets

For the general-purpose sequence matching benchmark, we simulate pairs of nucleotide sequences of length 10,000 with mutation rates 0.01, 0.05, and 0.1 (insertions, deletions, or substitutions occurring with equal probability 1/3). Each sequence pair is replicated 1000 times to reduce sample variability. We refer to these pairs of simulated sequences as the **SIM-R** dataset. Such experiment setup was performed in [34] to evaluate different strobemer constructions. For the uniqueness evaluation, we measure the fraction of unique seeds (occurring once) and the E-hits [35] in the human cell line CHM13 assembly [30], referred to as **CHM13**.

For the read alignment benchmark, we use mason2 [14] to simulate single-end and paired-end Illumina reads of various lengths from 50–500 nt from the CHM13 (3.1 Gbp), fruit fly (144 Mbp), maize (2.2 Gbp) and rye [22] (7.3 Gbp) genomes. We generate three datasets differing in mutation rate that we call **SIM3**, **SIM4**, and **SIM5**. **SIM3** has the lowest mutation rate and mimics roughly the average error rates seen between humans. Datasets **SIM3** and **SIM4** have been used in [35]. **SIM5** with the highest mutation rate was generated for this study. We also performed a SNV and indel calling benchmark as performed in [35], using bcftools as the variant caller with two simulated (**SIM150**, **SIM250**) and two biological (**BIO150**, **BIO250**) datasets. See Supplementary Section 2 for relevant mason2 command-line options.

Sequence matching experiment

Benchmark. We perform the sequence matching benchmark of seeds similarly to the procedure described in [34]. Given a pair of simulated sequences (s, s') from the **SIM-R** dataset, we construct canonical k-mers, randstrobes, and MCS with symmetric hash functions from every possible starting position of the two sequences. We count the number of matches between seeds generated from s and seeds generated from s' (including partial matches for MCS) for every type of seed construct. Note that the strobemers study [34] did not use canonical k-mers and symmetrical hash functions. We use them here for more practical similarity with the read mapping problem. Similarly to the strobemers study [34], we compute match percentage, match coverage, and sequence coverage and the island E-size metric as defined in [34]. Briefly, match percentage is the fraction of seed matches, match coverage is the percentage of sequence in s spanned by seed matches, sequence coverage is the percentage of sequence in s covered by the actual strobes of a seed. Since k-mers are ungapped, they have identical match and sequence coverage. The island E-size is a measure of the size of gaps (islands [5]) without matches. We describe these metrics in detail in Supplementary Section 3.

		SIM-R											
		0.01				0.05				0.1			
		m	sc	mc	E	m	sc	mc	E	m	sc	mc	E
kmers	30	74.5	96.0	96.0	1.1	22.4	54.7	54.7	43.5	4.7	18.1	18.1	292.8
kmers	10	**90.4**	99.2	99.2	**0.0**	**62.3**	91.2	91.2	0.9	**38.7**	75.7	75.7	4.2
kmers	15	86.7	98.6	98.6	0.1	48.3	81.7	81.7	4.2	22.6	53.9	53.9	22.5
kmers	20	82.4	97.9	97.9	0.3	37.4	72.6	72.6	11.0	13.4	38.5	38.5	62.6
randstrobes	(2, 15, 25, 50)	70.7	98.2	99.9	0.0	18.1	72.7	87.8	8.2	3.4	31.2	44.7	117.8
randstrobes	(3, 10, 25, 50)	66.7	98.8	**100.0**	**0.0**	14.7	78.4	98.2	1.1	2.5	33.7	67.0	68.9
multi-context	(2, 15, 25, 50)	79.5	99.8	**100.0**	**0.0**	39.0	94.3	97.0	0.7	17.8	72.7	76.7	11.3
multi-context	(3, 10, 25, 50)	81.9	**99.9**	**100.0**	**0.0**	50.2	**97.7**	**99.9**	**0.0**	30.7	**88.4**	**96.4**	**0.6**

Table 1. Match statistics for k-mers, randstrobes, and MCS under mutation rates of 0.01, 0.05, and 0.1. Here, m denotes the number of matches (both full and partial for MCS) as a percentage of the total number of extracted subsequences for the protocol, sc (sequence coverage) and mc (match coverage) are shown as the percentage of the total sequence length, and E is the expected island size.

Results. Table 1 shows the mean match percentage, total match and sequence coverage, and island E-size for the **SIM-R** dataset. We benchmarked randstrobes and MCS with parametrizations $(2, 15, 25, 50)$ and $(3, 10, 25, 50)$ (as used in [34]). For the k-mers, we chose k size 30 corresponding to the total strobemer size, and k sizes 10, 15, and 20 corresponding to all the possible partial match sizes of the multi-context seed parametrizations. We observe that k-mers of size 10 have the highest fraction of matches, since the seeds of smaller size are more likely to be preserved between mutation positions. Between the seeds of the same length, i.e. randstrobes, MCS, and k-mers of size 30, MCS with parametrization $(3, 10, 25, 50)$ show the highest match percentage due to production of both full and partial matches. MCS have higher match coverage and sequence coverage than randstrobes due to the presence of additional partial matches.

Since MCS can make use of the strengths of both k-mers and randstrobes, it is expected that they largely outperform both k-mers and randstrobes in this analysis. However, note that due to the usage of canonical k-mers, some of the match positions produced by k-mers will not be partial match positions of the MCS of corresponding size, i.e., they are never selected as the base part of the hash. This explains the higher match percentage of 10-mers and 15-mers compared to respective multi-context seed parametrizations $(3, 10, 25, 50)$ and $(2, 15, 25, 50)$

		CHM13								
		% Unique			E-hits			#Distinct (millions)		
		Full	1st	1st & 2nd	Full	1st	1st & 2nd	Full	1st	1st & 2nd
kmers	48	0.97	-	-	152.92	-	-	2706.93	-	-
kmers	16	0.43	-	-	2120.16	-	-	800.25	-	-
kmers	24	0.96	-	-	1051.20	-	-	2365.81	-	-
kmers	32	0.97	-	-	429.93	-	-	2528.78	-	-
randstrobes	(2,24,36,72)	0.98	-	-	181.70	-	-	2754.53	-	-
multi-context	(2,24,36,72)	0.98	0.57	-	181.70	2580.39	-	2754.49	1506.04	-
randstrobes	(3,16,36,72)	0.98	-	-	103.04	-	-	2808.30	-	-
multi-context	(3,16,36,72)	0.98	0.40	0.95	102.64	19988.10	614.14	2804.93	568.53	2540.08

Table 2. Uniqueness statistics for k-mers, randstrobes, and MCS. For the MCS, every given seed is estimated separately as its full hash value ("Full"), subhash corresponding to the first strobe ("1st"), and subhash corresponding to the first two strobes ("1st & 2nd") for 3rd order MCS. "% Unique" denotes the percentage of the total number of seeds of the given type that have unique hash value, "E-hits" denotes the E-hits metric of the given seed type, and "#Distinct" shows the number of distinct seeds of the given type (in millions).

Seed uniqueness

Benchmark. While MCS show higher match coverage than randstrobes and k-mers due to the presence of partial matches, they are also expected to result in a larger amount of spurious matches for the same reason. The previous experiment was designed to be without repeats to accurately measure (true) coverage of matches. In order to assess the potential increase in the spurious matches, we benchmarked MCS against k-mers and randstrobes on the CHM13 reference genome using three metrics, namely the percentage of unique seeds, number of distinct seeds, and the E-hits metric [35]. Roughly, E-hits measures the expected number of occurrences of a seed if a seed is selected uniformly at random in the database, see [35] for details. We measure the seed uniqueness metrics by indexing the CHM13 reference genome and counting the seed occurrence distribution. We estimate the k-mer occurrence distribution using KMC [20] k-mer counter tool, and strobealign for calculating randstrobes and MCS occurrence distribution. For this experiment, we use all k-mers in the reference in strobealign, i.e., we do not use the open-syncmers downsampling in strobealign. For the MCS, we provide the number of partial hits separately from the number of full hits. We use a range of typical k-mer sizes used for human genome applications and resulting matching strobemer parameterizations (dividing the k-mer size by 2 or 3). We used 64 bit hashes for randstrobes and multi-context seeds.

Results. We observe that MCS (full hits) and randstrobes are nearly identical in uniqueness (Table 2). This suggests that even though MCS splits the bits in the hash value representation to enable multi-level search, it does not result in significant hash collisions. At least not in a human genome. As expected, partial matches

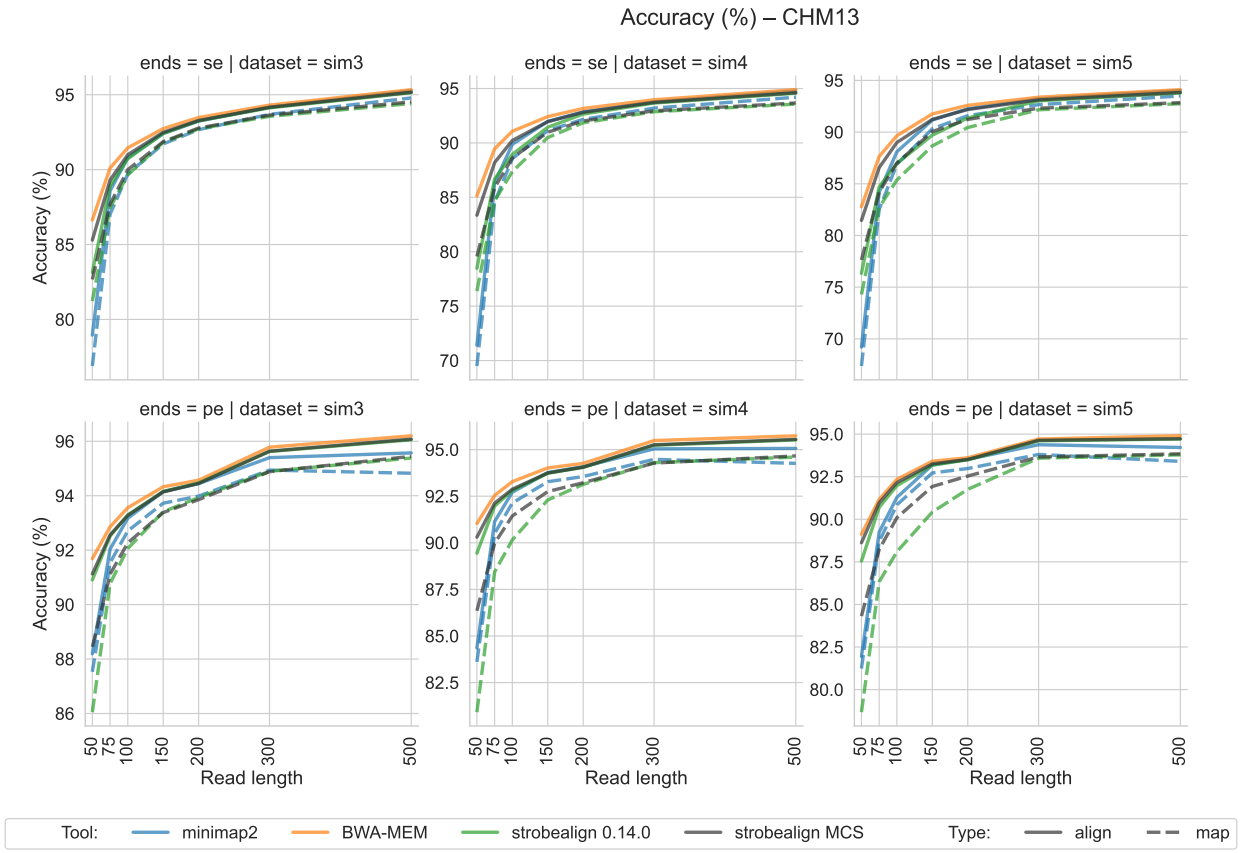

Fig. 2. Accuracy of single-end (top row) and paired-end (bottom row) read alignment for the **SIM3** (left), **SIM4** (middle), and **SIM5** (right) datasets. Type "align" is extension-alignment mode (SAM output), type "map" is mapping-only mode (coordinate output).

are more redundant than full matches. Also, the partial hits (1st strobe only) of the MCS are less unique than k-mers of the same length. This difference can be attributed to the symmetric MCS implementation, where we pick the strobe with the smallest hash value to act as the base. That is, not all k-mers will exist as a base hash value, making partial hits more repetitive.

Read mapping evaluation

We compared our variant of strobealign with MCS to strobealign, BWA-MEM and minimap2. For the single-end and paired-end reads experiments, we show the accuracy and runtime for the **SIM3**, **SIM4** and **SIM5** datasets for CHM13 in Fig. 2 and Fig. 3. Results for all four genomes (also rye, fruit fly, and maize) and measurements of memory usage and mapping rate (percentage of aligned reads) are found in Suppl. Figs. S2–S7.

Read mapping results. For paired-end reads, strobealign with MCS improves both the accuracy (Figure 2, Suppl. Fig. S3) and the percentage of aligned reads (Suppl. Fig. S7) over the default randstrobe version of strobealign both in the mapping-only and extension-alignment mode in all genomes. The improvement of MCS over the default randstrobe seeds in the alignment mode is most apparent for reads up to length 100 bp in the **SIM5** dataset (high mutation rate), with 0.51%, 0.73%, 0.43%, 0.65% mean accuracy increase for CHM13, maize, fruit fly, and rye genomes, respectively. This is expected as the partial seed matches can produce hits around variations and other divergent regions where no full matches or randstrobes were produced. In the mapping-only mode (no extension alignment), the accuracy increase is noticeable for read lengths 150 or lower in the **SIM4** dataset (2.18% mean accuracy increase for CHM13, 2.44% mean accuracy

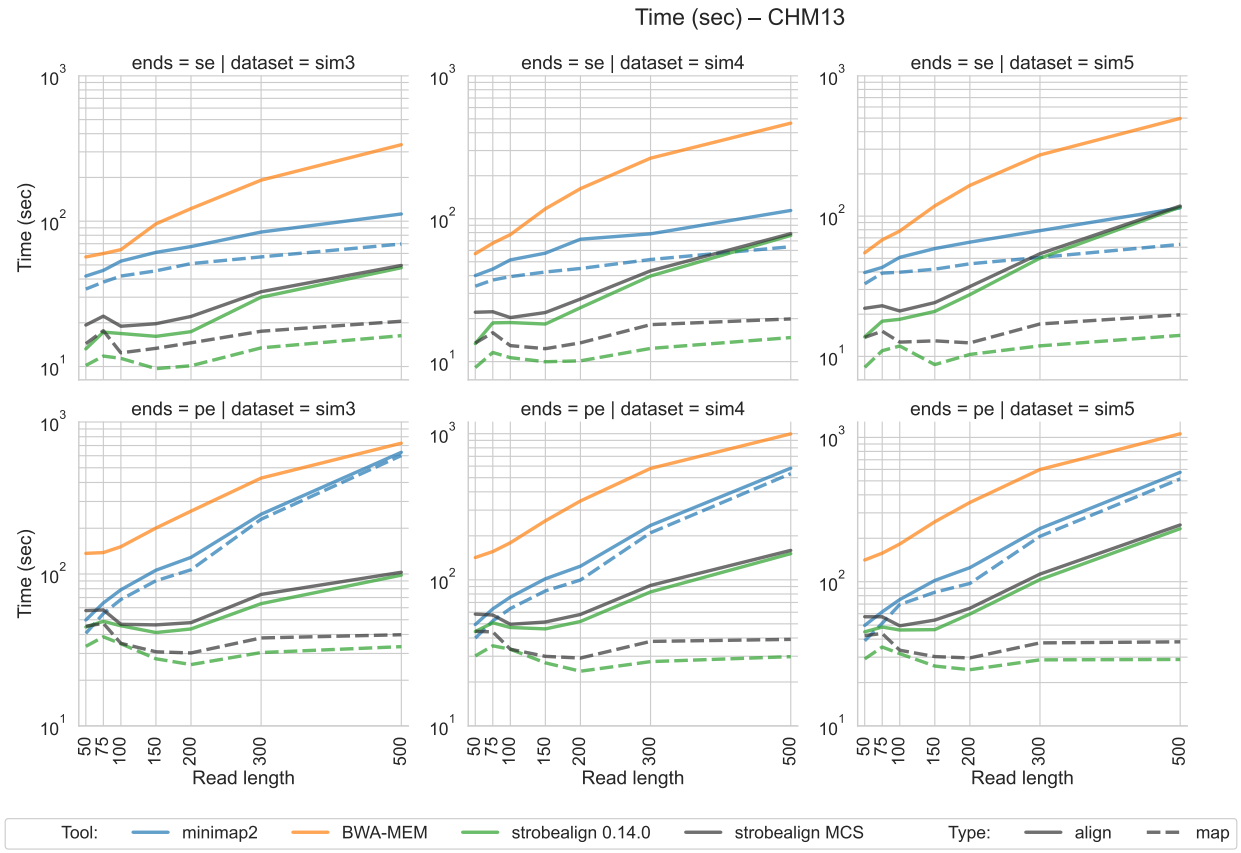

Fig. 3. Runtime of single-end (top row) and paired-end (bottom row) read alignment for the **SIM3** (left), **SIM4** (middle), and **SIM5** (right) datasets

increase for the fruit fly) and for read length 200 or lower in the **SIM5** dataset (2.36% mean accuracy increase for CHM13, 2.61% mean accuracy increase for fruit fly). The mapping-only mode is useful for rapid metagenomics analysis with the AEMB mode ([1], forthcoming study). In both read-mapping and extension modes, strobealign with MCS seeds further narrows the mean accuracy gap for shorter reads compared to gold-standard BWA-MEM, while still being $1.5 - 7.6$ (mean 3.78) times faster in the alignment mode.

For single-end reads, the relative increase in accuracy of MCS compared to default randstrobe-based strobealign is even larger with 1.8% mean accuracy increase for CHM13, 2.06% for fruit fly, 1.07% for rye, and 1.2% for maize genome for reads shorter than 150 in the alignment mode (Figure 2, Suppl. Fig. S2). Relatively better performance in the single-end mode can be explained by strobealign's rescue procedure in paired-end mode, which utilizes the matches of a read to recover its unmapped mate. In addition, strobealign with MCS also substantially improves the percentage of mapped reads over default strobealign for the shorter read lengths (Suppl. Fig. S6).

Strobealign with MCS is slightly slower than default strobealign (Figure 3, Suppl. Figs. S4, S5) with runtime increasing by 16% on average (range −1% to 62% for single-end reads and 11% on average (range −9% to 33%) for paired-end reads, owing to the higher redundancy and extra lookups of partial hits, as well as the extra alignment sites produced by partial matches found by strobealign in the absence of full matches. However, we believe this overhead can be reduced by further optimizing the implementation (`https://github.com/ksahlin/strobealign/pull/426#issuecomment-2379082142`).

Indexing time. Indexing speed is not measurably affected by enabling the MCS hash function. Using four threads and leaving all other indexing parameters (k, s, etc.) left unchanged, strobealign both with and without MCS takes 2 s, 29 s, 43 s and 111 s to index fruit fly, maize, CHM13, and rye, respectively. For

reference, BWA and minimap2 take 2420 s and 47 s, respectively, to index CHM13 on the same machine (using four threads for minimap2; BWA does not parallelize index creation).

Variant calling. We ran the variant calling benchmark on two simulated and two biological Illumina datasets as performed in [35] (Suppl. Fig. S1). In terms of F-score, the tools perform roughly similar. BWA-MEM has highest SNV recall for all datasets, while the tools perform similarly on indels. Note however that we use bcftools to call variants. Some callers may be developed or tuned based on popular aligners' mapping quality (MAPQ) scores. Specifically, it has been shown that *bcftools call* was the variant caller that produced the best results with BWA-MEM alignments out of seven variant calling tools [43]. Therefore, we view this analysis as a sanity check rather than a perfect indicator of mapping quality. As with the simulated data, strobealign is substantially faster than minimap2 and BWA-MEM. There is no significant difference between strobealign versions in calls. While MCS enables looking for partial matches and would enable split or supplementary mapping, it is not yet implemented in strobealign. We hope to include split read mapping in future work, which may improve variant calling further.

Discussion and conclusions

We presented a new seeding approach, multi-context seeds (MCS), that allows multiple layers of matches to be searched. A multi-context seed is a strobemer represented by a 64-bit hash in which a separate substring of bits is allocated for each strobe in a strobemer. This allows the seed to be queried at multiple levels. Particularly, in a sorted array such as the AEMB index (the index used in strobealign), this enables both full and partial matches to be searched in a cache-efficient manner.

We show that multi-context seeds produce more matches than k-mers and randstrobes with the same subsequence size (Table 1), which indicates increased alignment sensitivity. Additionally, we show that multi-context seeds are as unique as randstrobes and k-mers if the full hash is searched (Table 2). We further show the practical use of MCS by implementing them in the read mapper strobealign. We show that strobealign with MCS significantly increases accuracy in paired-end read alignment and mapping (Figures 2, Suppl. Figs. S2, S3, S6, S7) for reads shorter than 100 compared to randstrobe-based strobealign. Strobealign with MCS is slightly slower than randstrobe-based strobealign, while still being noticeably faster than BWA-MEM. BWA-MEM has a slightly faster but much more memory-consuming variant, BWA-MEM2 [39], which we have benchmarked against strobealign [35]. However, optimized implementations now also exist for strobealign [41]) that doubles the speed.

As for future work, we are currently integrating MCS in strobealign, where we plan to use it by default for shorter read lengths (about 100 nt for extension alignment and 200 nt for mapping), as we observe the largest improvements for those read lengths. MCS could also enable better split-read breakpoint detection with short reads and long-read alignment, where MCS of, e.g., three strobes enabling a three layer search may be beneficial for long reads.

Availability Strobealign is available at `https://github.com/ksahlin/strobealign`, see pull request #426 for the multi-context seeds implementation. The bit allocation analysis is found at `https://github.com/ksahlin/misc/tree/main/mcs`. The code for the sequence matching results is located at `https://github.com/Itolstoganov/strobemers/tree/multi-context-benchmarks`. The code for the uniqueness benchmark is available at `https://github.com/Itolstoganov/strobealign/tree/uniqueness_benchmark`. The code for the read alignment experiments is available at `https://github.com/NBISweden/strobealign-evaluation/`. Biological datasets BIO150 (Illumina WGS 2x150bp, HG004) and BIO250 (Illumina WGS 2x250bp, HG004) analyzed in this study are found at `https://github.com/genome-in-a-bottle/giab_data_indexes`.

Acknowledgements. Parts of the evaluation were enabled by resources in project snic2022-5-592 provided by the Swedish National Infrastructure for Computing (SNIC) at UPPMAX, partially funded by the Swedish Research Council through grant agreement no. 2018-05973.

Funding. Kristoffer Sahlin was supported by the Swedish Research Council (SRC, Vetenskapsrådet) under Grant No. 2021-04000. Part of this work was supported by the SciLifeLab & Wallenberg Data Driven Life Science Program, Knut and Alice Wallenberg Foundation (grants: KAW 2020.0239 and KAW 2017.0003), and by the National Bioinformatics Infrastructure Sweden (NBIS) at SciLifeLab.

Bibliography

[1] Strobealign pull request containing --aemb module implementation. https://github.com/ksahlin/strobealign/pull/394.

[2] 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.

[3] Anton Bankevich, Andrey V. Bzikadze, Mikhail Kolmogorov, Dmitry Antipov, and Pavel A. Pevzner. Multiplex de Bruijn graphs enable genome assembly from long, high-fidelity reads. *Nature Biotechnology*, 40(7):1075–1081, 2022.

[4] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, Alexey V Pyshkin, Alexander V Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max A Alekseyev, and Pavel A Pevzner. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J Comput Biol*, 19(5):455–477, May 2012.

[5] Antonio Blanca, Robert S. Harris, David Koslicki, and Paul Medvedev. The statistics of k-mers from a sequence undergoing a simple mutation process without spurious matches. *Journal of Computational Biology*, 29(2):155–168, February 2022.

[6] 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.

[7] Arthur L. Delcher, Adam Phillippy, Jane Carlton, and Steven L. Salzberg. Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Research*, 30(11):2478–2483, June 2002.

[8] 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, January 2013.

[9] Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, February 2021.

[10] Barış Ekim, Bonnie Berger, and Rayan Chikhi. Minimizer-space de Bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer. *Cell Syst*, 12(10):958–968, October 2021.

[11] Barış Ekim, Kristoffer Sahlin, Paul Medvedev, Bonnie Berger, and Rayan Chikhi. Efficient mapping of accurate long reads in minimizer space with mapquik. *Genome Research*, 2023.

[12] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. volume 2000, pages 390–398, February 2000.

[13] Can Firtina, Jisung Park, Mohammed Alser, Jeremie S Kim, Damla Senol Cali, Taha Shahroodi, Nika Mansouri Ghiasi, Gagandeep Singh, Konstantinos Kanellopoulos, Can Alkan, and Onur Mutlu. BLEND: a fast, memory-efficient and accurate mechanism to find fuzzy seed matches in genome analysis. *NAR Genom Bioinform*, 5(1):lqad004, March 2023.

[14] M. Holtgrewe. Mason: A read simulator for second generation sequencing data. *Technical Report FU Berlin*, October 2010.

[15] Chirag Jain, Arang Rhie, Nancy F. Hansen, Sergey Koren, and Adam M. Phillippy. Long-read mapping to repetitive reference sequences using Winnowmap2. *Nature Methods*, 19(6):705–710, 2022.

[16] Amir Joudaki, Alexandru Meterez, Harun Mustafa, Ragnar Groot Koerkamp, André Kahles, and Gunnar Rätsch. Aligning distant sequences to graphs using long seed sketches. *Genome Research*, 2023.

[17] Chelsea J.-T. Ju, Jyun-Yu Jiang, Ruirui Li, Zeyu Li, and Wei Wang. Tahcoroll: fast genomic signature profiling via thinned automaton and rolling hash. *Medical Review*, 1(2):114–125, December 2021.

[18] Chelsea J.-T. Ju, Ruirui Li, Zhengliang Wu, Jyun-Yu Jiang, Zhao Yang, and Wei Wang. Fleximer: Accurate quantification of RNA-seq via variable-length k-mers. In *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology,and Health Informatics*, ACM-BCB '17, pages 263—272, New York, NY, USA, 2017. Association for Computing Machinery.

[19] Moein Karami, Aryan Soltani Mohammadi, Marcel Martin, Barış Ekim, Wei Shen, Lidong Guo, Mengyang Xu, Giulio Ermanno Pibiri, Rob Patro, and Kristoffer Sahlin. Designing efficient randstrobes for sequence similarity analyses. *Bioinformatics*, 40(4):btae187, April 2024.

[20] Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, May 2017.

[21] Ben Langmead and Steven L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9(4):357–359, April 2012.

[22] Guangwei Li, Lijian Wang, Jianping Yang, Hang He, Huaibing Jin, Xuming Li, Tianheng Ren, Zhenglong Ren, Feng Li, Xue Han, Xiaoge Zhao, Lingli Dong, Yiwen Li, Zhongping Song, Zehong Yan, Nannan Zheng, Cuilan Shi, Zhaohui Wang, Shuling Yang, Zijun Xiong, Menglan Zhang, Guanghua Sun, Xu Zheng, Mingyue Gou, Changmian Ji, Junkai Du, Hongkun Zheng, Jaroslav Doležel, Xing Wang Deng, Nils Stein, Qinghua Yang, Kunpu Zhang, and Daowen Wang. A high-quality genome assembly highlights rye genomic characteristics and agronomically important genes. *Nature Genetics*, 53(4):574–584, March 2021.

[23] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM, 2013.

[24] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, May 2018.

[25] Xiang Li, Ke Chen, and Mingfu Shao. Efficient seeding for error-prone sequences with SubseqHash2. *bioRxiv*, 2024.

[26] Xiang Li, Qian Shi, Ke Chen, and Mingfu Shao. Seeding with minimized subsequence. *Bioinformatics*, 39(Supplement_1):i232–i241, June 2023.

[27] Bin Ma, John Tromp, and Ming Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, March 2002.

[28] Benjamin Dominik Maier and Kristoffer Sahlin. Entropy predicts sensitivity of pseudorandom seeds. *Genome Research*, May 2023.

[29] Guillaume Marçais, Dan DeBlasio, and Carl Kingsford. Asymptotically optimal minimizers schemes. *Bioinformatics*, 34(13):i13–i22, June 2018.

[30] Sergey Nurk, Sergey Koren, Arang Rhie, Mikko Rautiainen, Andrey V. Bzikadze, Alla Mikheenko, Mitchell R. Vollger, Nicolas Altemose, Lev Uralsky, Ariel Gershman, Sergey Aganezov, Savannah J. Hoyt, Mark Diekhans, Glennis A. Logsdon, Michael Alonge, Stylianos E. Antonarakis, Matthew Borchers, Gerard G. Bouffard, Shelise Y. Brooks, Gina V. Caldas, Nae-Chyun Chen, Haoyu Cheng, Chen-Shan Chin, William Chow, Leonardo G. de Lima, Philip C. Dishuck, Richard Durbin, Tatiana Dvorkina, Ian T. Fiddes, Giulio Formenti, Robert S. Fulton, Arkarachai Fungtammasan, Erik Garrison, Patrick G. S. Grady, Tina A. Graves-Lindsay, Ira M. Hall, Nancy F. Hansen, Gabrielle A. Hartley, Marina Haukness, Kerstin Howe, Michael W. Hunkapiller, Chirag Jain, Miten Jain, Erich D. Jarvis, Peter Kerpedjiev, Melanie Kirsche, Mikhail Kolmogorov, Jonas Korlach, Milinn Kremitzki, Heng Li, Valerie V. Maduro, Tobias Marschall, Ann M. McCartney, Jennifer McDaniel, Danny E. Miller, James C. Mullikin, Eugene W. Myers, Nathan D. Olson, Benedict Paten, Paul Peluso, Pavel A. Pevzner, David Porubsky, Tamara Potapova, Evgeny I. Rogaev, Jeffrey A. Rosenfeld, Steven L. Salzberg, Valerie A. Schneider, Fritz J. Sedlazeck, Kishwar Shafin, Colin J. Shew, Alaina Shumate, Ying Sims, Arian F. A. Smit, Daniela C. Soto, Ivan Sović, Jessica M. Storer, Aaron Streets, Beth A. Sullivan, Françoise Thibaud-Nissen, James Torrance, Justin Wagner, Brian P. Walenz, Aaron Wenger, Jonathan M. D. Wood, Chunlin Xiao, Stephanie M. Yan, Alice C. Young, Samantha Zarate, Urvashi Surti, Rajiv C. McCoy, Megan Y. Dennis, Ivan A. Alexandrov, Jennifer L. Gerton, Rachel J. O'Neill, Winston Timp, Justin M. Zook, Michael C. Schatz, Evan E. Eichler, Karen H. Miga, and Adam M. Phillippy. The complete sequence of a human genome. *Science*, 376(6588):44–53, April 2022.

[31] Yu Peng, Henry C. M. Leung, S. M. Yiu, and Francis Y. L. Chin. *IDBA – A Practical Iterative de Bruijn Graph De Novo Assembler*, page 426–440. Springer Berlin Heidelberg, 2010.

[32] 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, July 2004.

[33] Gordon Robertson, Jacqueline Schein, Readman Chiu, Richard Corbett, Matthew Field, Shaun D Jackman, Karen Mungall, Sam Lee, Hisanaga Mark Okada, Jenny Q Qian, Malachi Griffith, Anthony Raymond, Nina Thiessen, Timothee Cezard, Yaron S Butterfield, Richard Newsome, Simon K Chan, Rong She, Richard Varhol, Baljit Kamoh, Anna-Liisa Prabhu, Angela Tam, YongJun Zhao, Richard A Moore, Martin Hirst, Marco A Marra, Steven J M Jones, Pamela A Hoodless, and Inanc Birol. De novo assembly and analysis of rna-seq data. *Nature Methods*, 7(11):909–912, October 2010.

[34] Kristoffer Sahlin. Effective sequence similarity detection with strobemers. *Genome Research*, 31(11):2080–2094, October 2021.

[35] Kristoffer Sahlin. Strobealign: flexible seed size enables ultra-fast and accurate read alignment. *Genome Biology*, 23(1), December 2022.

[36] Kristoffer Sahlin, Thomas Baudeau, Bastien Cazaux, and Camille Marchet. A survey of mapping algorithms in the long-reads era. *Genome Biology*, 24(1):133, 2023.

[37] 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/PODS03. ACM, June 2003.

[38] Fritz J. Sedlazeck, Philipp Rescheneder, Moritz Smolka, Han Fang, Maria Nattestad, Arndt von Haeseler, and Michael C. Schatz. Accurate detection of complex structural variations using single-molecule sequencing. *Nature Methods*, 15(6):461–468, 2018.

[39] Md. Vasimuddin, Sanchit Misra, Heng Li, and Srinivas Aluru. Efficient architecture-aware acceleration of bwa-mem for multicore systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 314–324, 2019.

[40] Hongyi Xin, Mingfu Shao, and Carl Kingsford. Context-aware seeds for read mapping. *Algorithms for Molecular Biology*, 15(1):10, 2020.

[41] Lifeng Yan, Zekun Yin, Jinjin Li, Yang Yang, Tong Zhang, Fangjin Zhu, Xiaohui Duan, Bertil Schmidt, and Weiguo Liu. Rabbitsalign: Accelerating short-read alignment for cpu-gpu heterogeneous platforms. In Wei Peng, Zhipeng Cai, and Pavel Skums, editors, *Bioinformatics Research and Applications*, pages 83–94, Singapore, 2024. Springer Nature Singapore.

[42] 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.

[43] Zhen Yao, Frank M You, Amidou N'Diaye, Ron E Knox, Curt McCartney, Colin W Hiebert, Curtis Pozniak, and Wayne Xu. Evaluation of variant calling tools for large plant genome re-sequencing. *BMC Bioinformatics*, 21:1–16, 2020.