Strobemers: an alternative to k-mers for sequence comparison

Kristoffer Sahlin[1,✉]

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

K-mer-based methods are widely used in bioinformatics for various types of sequence comparison. However, a single mutation will mutate k consecutive k-mers and makes most k-mer based applications for sequence comparison sensitive to variable mutation rates. Many techniques have been studied to overcome this sensitivity, *e.g.*, spaced k-mers and k-mer permutation techniques, but these techniques do not handle indels well. For indels, pairs or groups of small k-mers are commonly used, but these methods first produce k-mer matches, and only in a second step, a pairing or grouping of k-mers is performed. Such techniques produce many redundant k-mer matches due to the size of k.

Here, we propose *strobemers* as an alternative to k-mers for sequence comparison. Intuitively, strobemers consists of linked minimizers. We show that under a certain minimizer selection technique, strobemers provide more evenly distributed sequence matches than k-mers and are less sensitive to different mutation rates and distributions. Strobemers also produce a higher total match coverage across sequences. Strobemers are a useful alternative to k-mers for performing sequence comparisons as commonly used in sequence alignment, clustering, classification, and error-correction. A reference implementation with code for analyses is available at https://github.com/ksahlin/strobemers.

k-mers | minimizers | sequence matching | data structures

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

DRAFT

Introduction

The decreased sequencing costs have led to a dramatic increase in sequencing data over the past two decades, which has prompted development of computational methods for sequence comparison. A popular sequence comparison paradigm is k-mer based analysis, where k-mers are substrings of length k of, *e.g.*, genomic, transcriptomic, or protein sequences. K-mer-based methods have been applied for sequence comparison for error correction (1), genome assembly (2, 3), metagenomic (4) and chromosome (5) sequence classification, sequence clustering (6), database search (7, 8), structural variation detection (9, 10), transcriptome analysis (11, 12), and many other applications. Because of this widespread use, many data structures and techniques for efficiently storing, querying, and counting k-mers have been proposed (see (13) for a review).

While k-mers has proven to be practical in several sequence comparison problems, they are sensitive to mutations. A mutation will mutate k consecutive k-mers across a string. As the mutation rate increases, the number of matching k-mers quickly reduces. In (14), the distribution of mutated k-mers were studied in detail. The authors provided closed-form expressions for the mean and variance estimates on the number of mutated k-mers under a random mutation model. For sequence comparison methods, while the number of k-mer matches between sequences is of interest, it is often more informative to know how they are distributed across the matching region. K-mer matches, because of their consecutive nature, cluster tightly in shared sequence regions, while matches may be absent in regions with higher mutation rates. Spaced k-mers (or spaced seeds) have been studied in several sequence comparison contexts to overcome the k-mers' sensitivity to mutations (15–17). The advantage of spaced k-mers is that matches of spaced k-mers at different positions are less correlated with each other than k-mer matches. In fact, k-mers are the worst seed pattern for the problem of similarity search (18). Another innovative idea has been to permute the letters in a string before comparison (19, 20). The main idea is that, under several different permutations in a region of a string, at least one permutation will, with statistical certainty, have pushed any mutation(s) towards the end of the strings under comparison, which allows for constant-time query of the prefix of the permuted strings. However, both spaced k-mers and permutation techniques are only practical for substitutions. An insertion or deletion (*indel*) will shift the sequence and, similarly to k-mers, result in a long stretch of dissimilar k-mers. For certain applications such as genome assembly, selecting several sizes of k for inference has also shown to help sequence comparison (21), but it significantly increases runtime and complexity of analysis. There are also methods to collapse repetitive regions before k-mer based comparison (22), which reduces the processing time of repetitive hits. However, such techniques is usually employed for reference-based analysis and do not apply to general sequence comparison problems.

As third-generation sequencing techniques appeared with sequencing errors mostly consisting of insertions and deletions, many of the previously developed sequence comparison techniques for short-read data became unsuitable. For the third generation sequencing, MinHash (23) and minimizers (24, 25) proved to be useful data structures for such sequence comparison as minimizers can be preserved in a window affected by an indel. In addition, they also reduce the size of the index by subsampling the data. This has made MinHash and minimizers a popular technique for subsampling k-mers employed for sequence comparison in a range of applications such as metagenome distance estimation (26) and alignment (27, 28), clustering (29) error-correction (30),

and assembly (31) of long-read sequencing data. Because of the widespread practical use of minimizers, several methods have been proposed for sampling them with as low density as possible (32), more evenly (33, 34).

Due to the error rates of long-read sequencing, minimizers are often chosen much shorter (about 13-15nt) than what is considered to produce mostly unique k-mers in, e.g., the human genome (around $k > 20$nt). With this length of minimizers, they produce many candidate sequence matches. Therefore, it would be useful to combine the robustness of minimizers to indels and mutation errors with larger k-mer sizes that would offer more unique matches. One approach is to use a small k-mer size and identify pairs (35) or groups (36) of them clustered tightly together, and it has been studied how to design the sampling distribution of seeds to optimize alignment sensitivity (37, 38). Multi-seed methods are robust to any mutation type and have shown to, e.g., improve overlap detection between long reads (39). However, they still match single k-mers individually and group them based on statistics after individual k-mer hits have been found. To remove the redundancy in matches, we suggest that it is beneficial to couple the k-mers in the initial matching step and perform a single constant-time lookup of coupled k-mers. Coupled k-mers have been explored in, e.g., (30, 40) where *paired minimizers* are generated and stored as a single hash. A paired minimizer-match signals that the region is similar between sequences. Due to the gap between the minimizers, such a structure is not as sensitive to indels or substitutions as k-mers. Paired minimizers were shown to be useful for both genome assembly (40) and error correction of long cDNA reads (30) where the reads are similar only in some regions due to alternative splicing. However, in both (40) and (30), they consider pairs of minimizers where each minimizer is produced independently, and paired up after the minimizer generation. Here, we show that we can substantially improve on paired minimizers for sequence matching by producing minimizers chosen jointly depending on previous windows. We also generalize paired minimizers to link more than two together.

We propose a method to extract subsequences from a string and we call those subsequences *strobemers*. The name is inspired by strobe sequencing technology (an early Pacific Biosciences sequencing protocol), which would produce multiple subreads from a single contiguous fragment of DNA where the subreads are separated by 'dark' nucleotides whose identity is unknown, illustrated in (41). Strobemers introduced here are, however, produced computationally. Intuitively, strobemers are groups of linked short k-mers (*strobes*) from adjacent windows. The strobes can be chosen as minimizers either independently within windows (*e.g.*, as in (30, 40)), which we call *minstrobes*, or dependent on previous k-mers called *randstrobes*. We show that strobemers, and particularly randstrobes, improve sequence matching by providing more evenly distributed matches than k-mers, are less sensitive to different mutation rates and give a higher total coverage of matches across strings. We also show on human chromosomes that strobemers can offer higher uniqueness, and therefore confidence in a match, than k-mers due to the spacing of the strobes. Strobemers generalizes paired minimizers and we show that choosing minimizers dependent on previous k-mers (randstrobes) significantly improves over independently generated paired minimizers as in (30, 40). Strobemers are easy to both construct and query, making it a compelling alternative to k-mers for sequence comparison. Furthermore, strobemers can, similarly to k-mers, be subsampled using MinHash, minimizers, syncmers (34) or any other thinning protocol.

Methods

Definitions. 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. We use i to index the position in a string s and let $s[i : i+k]$ denote a *k-mer* substring at position i in s. We will consider 1-indexed strings. If $s[i : i+k]$ is identical to a k-mer $t[i' : i'+k]$ in string t, we say that the k-mers *match*, and that the match occurs at position i in s (and j in t). Similarly, let $f(i,k,s)$ be any function to extract a subsequence of length k with first letter at position i from s. If $f(i,k,s)$ is identical to $f(i',k,t)$, we say that the subsequences match, and that the match occurs at position i in s (and j in t). For example, for k-mers we have $f(i,k,s) = s[i : i+k]$. We let $|\cdot|$ denote the length of strings. We use h to denote a hash function $h : \sum^* \rightarrow \mathbb{Z}$ mapping strings to random numbers. Given two integers w and k such that $1 \leq k \leq w \leq |s|$, the minimizer at position i in s is the substring of s of length k starting in the interval of $[i, i+w-k+1)$ that minimizes h.

Aim. We will introduce strobemers by describing the problem they aim to solve. Consider two strings s and t that are identical up to m mutations. We desire a function f to produce a set of subsequences from s and t that have two characteristics: (1) as few possible placements of the m mutations result in no matches between s and t, and (2) the subsequences of length k should be as unique as k-mers on s and t. Characteristic 1 and 2 relates to the sensitivity and precision of searching for matches and we will discuss this in an example below. For practical purposes we also require that at most one subsequence is produced per position to mimic how k-mers are derived in a string (and limit the amount of data we store for each string). Certainly, we could produce all possible subsequences at each position to minimize criteria 1, but this is not feasible. A similar objective to characteristic 1 was studied for multi-seed design (37), where the authors want to find a set of seeds so that at least one seed matches a gapless aligment between two sequences.

A motivational example. Consider two strings of 100 nucleotides with $m = 3$ mutations between them. This could occur, *e.g.*, in splice alignment to an exon, or in sequence clustering. If we use a k-mer of size 30 to find matches, and the two strings differ at position 25, 50 and 75, there will

DRAFT

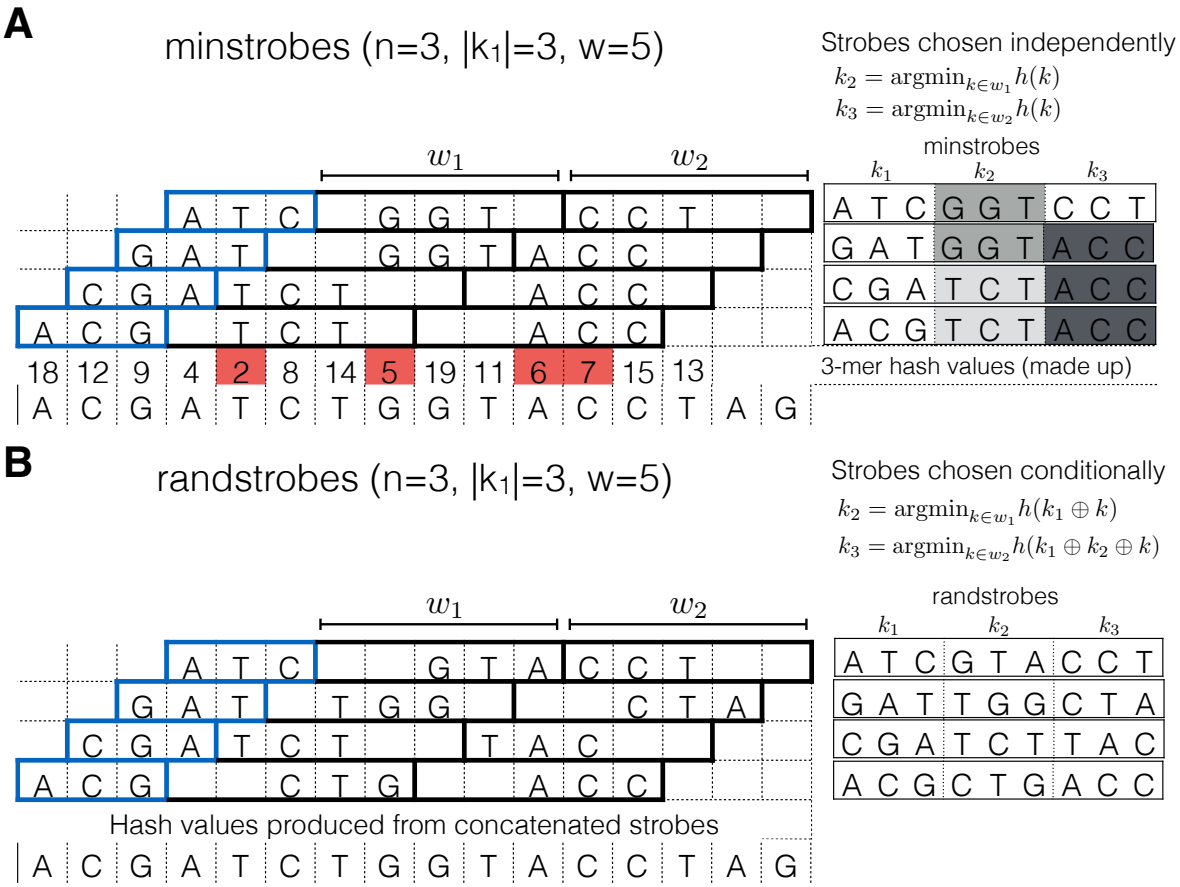

Fig. 1. An illustration of four minstrobes (A) and randstrobes (B) with $n = 3, |k_1| = 3, w = 5$ generated from a DNA string of 16 letters. The values on top of the nucleotides in A are made up hash values computed for each 3-mer under some random hash function h. The start k-mer k_1 in each of the four strobemers is highlighted in blue, and the adjacent window of five nucleotides is displayed in a boldfaced box. The 3-mer minimizers from all windows in the figure are indicated by red squares at first position of the 3-mer. In the minstrobe protocol (A), the minimizers in w_1 and w_1 are selected independently in each window from computing hash values $h(k)$, which gives rise to a high similarity between nearby strobemers (from sharing minimizers). The four minstrobes produced are shown to the right in A with shared minimizers between minstrobes in gray scale. In the randstrobe protocol (B), minimizers in w_1 and w_1 are selected dependent on the previous k-mers, *i.e.*, $h(k|k_1, \ldots, k_{i-1})$. The function producing the conditional dependence is irrelevant for the purpose of illustration. Here we use string concatenation of previous strobes to produce the dependence, but any other function will suffice. Because of the conditional dependence in the hash function, randstrobes are more randomly (but deterministically) distributed across the sequence.

be no matching k-mers. Similarly, this holds for mutations at positions 15, 40, and 65, and many other combinations. As described, we want as few possible placements of errors leading to the region being unmatched.

Using spaced k-mers (15) or permutations of the string (20) would help if the mutations were substitutions. We could consider lowering k, but this would generate more matches to other strings as well. To achieve the same uniqueness as longer k, we could consider coupled k-mers (35) of say 15nt per pair, with some gap in between them. Note that, to avoid many matches to other sequences, the k-mers would need to be coupled before searching for matches, not simply look for clustered k-mers. Furthermore, if the coupled k-mers have a fixed distance from each other, we have just created a specific type of spaced k-mers, which are only robust to substitutions. We, therefore, could consider coupled minimizers (30, 40) to select a random gap size for us, but in a deterministic manner.

This brings us to the strobemers. In the scenario above, we could pick a k-mer of size 15 at a position we want to sample and couple it with a minimizer of length 15 derived from a window downstream from the k-mer. Together, they have sequence length 30 and are therefore robust to false matches. They can also match across the mutations, where the mutations could be both substitutions and indels. If we increase the mutation density on our string, eventually, our two k-mers of length 15nt will also fail to produce any matches. Therefore, we could consider triplets of a k-mer and two minizizers of length, *e.g.*, 10nt. Finally, as we will show in this study, we could further reduce the sampled minimizers' dependency, and therefore the matches, by a joint hash function (over k-mers) to compute minimizers.

Strobemers. Consider a string s. A strobemer of order n in s is a subsequence of s at some position $i \in [1, |s| - k + 1]$. Specifically, a strobemer at position i in s is composed of a set of ordered non-overlapping substrings $k_1, ...k_n$ on s of equal length, that we call *strobes*, where $k_1 = s[i : i + |k_1|]$ and k_j, $j \geq 2$, are minimizers selected from adjacent windows of size $w > |k_j|$ such that the starting position of k_j falls within the window $w = s[i + |k_1| + (j-2)w : i + |k_1| + (j-1)w - |k_j| + 1]$ to assure that the minimizers do not overlap. Since we consider strobes of the same length we can rewrite the

expression as

$$w_j = s[i + |k_1| + (j-2)w : i + (j-1)w + 1].$$

We will from now on parametrize a strobemer as $(n, |k_1|, w)$ denoting the order, the length of the strobe, and the window size, respectively.

We select $k_2, ...k_n$ based on some hash function. We will consider two different hash functions for producing them, which give significantly different results. First we denote as *minstrobe*, a strobemer where strobes $k_2, \ldots, k_n$ are independently selected as minimizers in their respective windows under a hash function h.

Second, we denote as *randstrobe*, a strobemer where strobe k_j is selected as minimizer dependent on the previous $k_1, \ldots k_{j-1}$ strobes. Any asymmetric hash function with conditional dependence on previous strobes suffice for our purposes in this study. Here, we chose the hash function $h(k'|k_1, \ldots, k_{j-1}, k) = k_1 \oplus \ldots \oplus k_{j-1} \oplus k'$, where $\oplus$ denotes string concatenation, where h concatenates the previous selected strobes $k_1, \ldots, k_{j-1}$. Thus, the k-mers in the randstrobe are produced iteratively from $i = 1, .., n$ and yields a more randomly distributed set of strobes. Fig. 1 gives an illustration of minstrobes and randstrobes.

There are two important practical differences between minstrobes and randstrobes. Firstly, for two different starting strobes k_1 and k_1', k-mers $k_2, .., k_n$ will most frequently be the same under the minstrobe generation due to independent minimizers, and most frequently differ in a randstrobe. This means that under the same parameters in the protocols, the randstrobes will (in all likelihood) contain more uniquely sampled positions and, hence, more unique randstrobes, while minstrobes more frequently share minimizers. Secondly, generating minstrobes is in practice as fast as producing minimizers while generating randstrobes, under the function we consider here, is not. We elaborate on this in the section on time complexity.

Construction. We aim to produce a minstrobe or randstrobes of a string s in a similar manner to how k-mers are produced, *i.e.*, one strobemer per position $i \in [1, |s| - k + 1]$. This would mean that we extract the same amount of k-mers and strobemers from a string s, and consequently for equal length k, the same amount of raw data. Note however that the number of unique k-mers and strobemers may differ. We construct strobemers as follows. We denote the total length of the string considered to construct the strobe of order n as $W = |k_1| + nw$, and the total subsequence length as $k = \sum_{j=1}^{n} k_j$. If $W \leq |s| - i$ we use the predefined window size w and compute the strobemers under the respective hash functions described above. If $W > |s| - i$, we can narrow the window sizes until k_1 to k_n are all adjacent to each other producing a substring of length k. Any way to narrow the windows can be considered. Here, we choose to shorten each window w to $\frac{\lfloor |s| - i \rfloor}{n}$.

Time complexity. If we ignore the time complexity of the hash function, the time complexity of generating minimizers is $O(|s|w)$ for a window size w. However, as (27) noted, computing minimizers is in practise close to $O(|s|)$ if we use a queue to cache previous minimizer values in the window. The expensive step is when a previous minimizer is discarded from the queue and a new minimizer needs to be computed for the window.

Similar to computing minimizers, both minstrobes and randstrobes have the same worst-case time complexity of $O(|s|W)$. However, the independence of hash values in the minstrobes protocol makes it $O(|s|)$ in practice by using a queue in the same manner as computing minimizers independently. The randstrobe protocol does not have this independence under the hashing scheme we consider in this study, which means that all hash values have to be recomputed at each position. This means its practical time complexity is therefore $O(|s|W)$. We did not study different schemes to produce randmers but note that this is subject for future study. For example, it is possible that, by exploring symmetry properties (42), we may come up with faster methods to generate them.

Results

Here we will compare sequence matching performance for k-mers, spaced k-mers under two densities, minstrobes, and randstrobes (order 2 and 3). We first compare how effective the different protocols are at finding matches under different error rates. We then compare how effective they are at providing the same uniqueness (or confidence) in a match. Naturally, the size of k is central in these two aspects. We are interested in comparing sizes of subsequences that are similar between the protocols. Specifically, if the size of the k-mer is 30, we want to compare the k-mers to strobemers parameterized, e.g., by $(2, 15, \cdot)$ and $(3, 10, \cdot)$ as all the extracted subsequences have a length of 30 on the strings. The spaced k-mers consists of a window of size L with k *fixed* positions and a set of $L - k$ wildcard (or "don't care") positions. This is commonly represented as a binary string where 1's are sampled, and 0's are wildcard positions. For example, in the string $AGGTCA$ with $L = 6$, the spaced k-mer 101011 is $AGCA$. In our evaluations, we choose two densities of fixed positions for the spaced k-mers. First, we denote as *spaced-dense* a strategy where 2/3 of the positions are fixed, and *spaced-sparse* where 1/3 of the positions are fixed. The spaced-dense and the spaced-sparse frequency of fixed positions roughly correspond to the densities used in (16) and (43), respectively. To keep k fixed, $L = 1.5k$ in the spaced-dense protocol and $L = 3k$ in the spaced-sparse protocol. The windows' first and last positions are always fixed (as in (16, 43)) to assure the length of the spaced k-mer. The remaining fixed positions are randomly chosen. In, *e.g.*, (43), the sampled positions are handpicked. While handpicking positions may be more suitable for optimizing lower correlation between matches, it is out of scope for this study. We focus on designing a protocol robust to indels, and we will observe that spaced k-mers do not work well for mutations other than substitutions.

DRAFT

Evaluation metrics. We say that a match between two sequences s_1 and s_2 occur at position i and i' in the two strings respectively, if the k-mer (strobemer) extracted from position i in s and i' in t produce the same k-mer (strobemer). Furthermore, we say that this match *covers* positions $[i, i+k]$ for k-mers, and $[i, i+k_1]$ for strobemers in s. We adapt similar terminology as in (14) and denote a maximal interval of consecutive positions without matches between s and t as an *island*.

To evaluate the ability to preserve matches under different error rates, we compare (i) the number of matches, (ii) the total fraction of covered positions across the strings, and (iii) the distribution of islands. We need to make some clarifications on these evaluation metrics. First, our experiments on simulated data are designed with parameters so that the event of observing a false match under any protocol has a negligible probability. This means that our simulated experiments only measure the raw ability to identify correct matches.

Secondly, to compute the total coverage of matches across the strings, we let all the k positions contribute to k-mer and spaced k-mer coverage, as all the sampled positions are known in these protocols. For the strobemers, we only let the first $|k_1|$ nucleotides contribute to the coverage for the strobemer protocols. This is unfavorable for the strobemer-protocols as it only provides a lower bound on coverage. While we could store the other strobes' positions and get exact coverage, this is not the intended use of the data structure and such information is lost (under a non-reversible hash function). Therefore, we simply omit them from the coverage computation. Consequently, strobemer protocols will not produce perfect coverage even for identical strings as the last segment between the strings cannot be added to the coverage. Storing positional information of other strobes is subject to future study.

Finally, as for the distribution of islands, we calculate the average island size. However, only measuring average island size is not a fair metric if the methods differ significantly in the number of islands they produce. The strobemer protocols (particularly randstrobe) produce many small islands, which deflates the mean. We are particularly interested in the larger islands. Therefore, we also calculate the island E-size (44), a commonly used metric in genome assembly that we will adapt for our purposes. For a string s and a set of islands lengths X on s we calculate the *island E-size* E as follows

$$E = \frac{1}{|s|} \sum_{x \in X} x^2$$

E measures the *expected island size*, and intuitively, we can think of E as follows. If we pick a position at random across s, what is the island size in that position? At first, we may pick a position at random that are covered by matches (i.e., island size 0), but if we keep picking positions at random and store our observations on the island lengths, we will end up with E according to the *law of large numbers*. We will also show the entire island distribution.

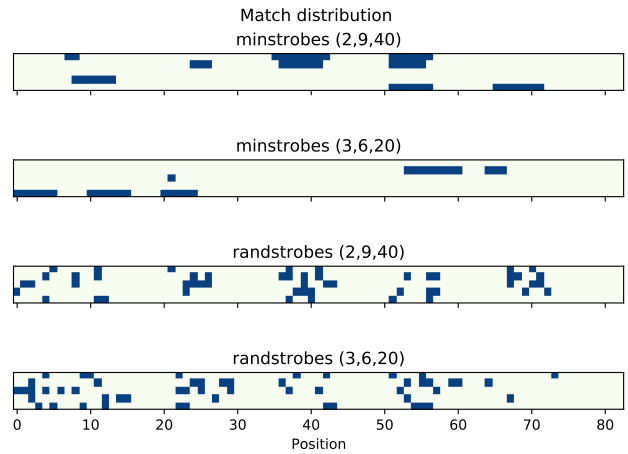

Fig. 2. An example of strobemer matches for minstrobes and randstrobes with two different parameterizations each (separate panels). Each panel shows matches between a string s of 100nt and a string t derived from simulating mutations every 15th position in s. Indels and substitutions are chosen at random with equal probability. The matches are plotted with respect to the positions in s on the 81 possible matching positions (x-axis). Each row in a panel corresponds to a separate simulation.

Strobemer vs k-mer matching. We compare how effective the different protocols are at preserving matches for different error rates. We start with a controlled scenario, where mutations are distributed with a fixed distance. In our second experiment, we use a random mutation distribution. We perform the fixed-distance mutation experiment to illustrate clearly the potential advantage of strobemer protocols.

Controlled mutations. First, we provide a small simulation to illustrate a scenario similar to the motivational example described earlier. We simulate a string s of 100 random nucleotides and a string t derived from simulating mutations every 15th position in s. Insertions, deletions, and substitutions are chosen at random with equal probability of 1/3 each. We simulate s and t five times to illustrate the variability in matches for the strobemers between simulations. The distribution of matches under two different parametrizations for minstrobes and randstrobes are shown in Fig. 2. We note that we would not not obtain any matches for k-mers of 15nt or larger in this scenario, and furthermore, no matches for spaced k-mers if the mutations were indels. Minstrobes, while more effective than k-mers in this scenario, fail to produce matches in simulation 3 for the $(2, 9, 40)$ parametrization and in simulation 1 and 4 for the $(3, 6, 20)$ parametrization. We observe that randstobes produce matches in all five experiments under both parametrizations and provide a more random match distribution across the string than minstrobes.

To better quantify the performance in this scenario, we increase the size of our controlled experiment. We simulate a string of length 10,000nt and construct a second string by generating an insertion, deletion, or substitution with a probability of 1/3 each, every 20 nucleotides. We then simulate k-mers with size 30, spaced-dense with $k = 30$, and $L = 45$, spaced-sparse with $k = 30$, $L = 90$, and strobemers with parameters (2,15,50) and (3,10,25) so that all protocols have the same sampled subsequence length, and compare the number

DRAFT

		SIM-C			
		m (%)	c (%)	g	E
k-mer	30	0	0	10,000	10,000
spaced k-mer	dense	1.1	17.6	259.3	600.2
	sparse	0.2	3.1	1,728.8	3251.0
minstrobe	(2,15,50)	4.7	19.2	99.7	169.6
	(3,10,25)	**9.0**	28.5	44.9	68.9
randstrobe	(2,15,50)	4.7	49.9	31.7	42.1
	(3,10,25)	**9.0**	**55.3**	**20.6**	**21.9**

Table 1. Statistics of the number of matches (m) as a percentage of the length of the original sequence of 10,000nt, the percentage of covered bases by matches (c) and the average island size (g) for the SIM-C dataset which has evenly spaced mutations with distance 20nt. The second column shows the parameters to the protocols.

		SIM-R											
		0.01				0.05				0.1			
		m	c	g	E	m	c	g	E	m	c	g	E
k-mer	30	**74.3**	95.7	38.4	7.8	**22.3**	54.5	75.3	78.9	**4.6**	18.1	206.7	343.6
spaced k-mer	dense	67.3	95.6	60.6	9.6	13.8	51.0	113.4	120.0	1.8	14.0	335.3	560.7
	sparse	50.0	87.7	143.9	44.5	3.5	21.7	415.1	622.6	0.1	2.1	2,231.8	3958.0
minstrobe	(2,15,50)	71.4	89.2*	26.6	8.3	18.8	40.8*	58.5	80.5	3.5	11.1*	185.9	360.2
	(3,10,25)	69.4	86.0*	19.8	7.0	16.7	36.1*	45.0	71.7	2.9	9.0*	153.3	330.2
randstrobe	(2,15,50)	71.4	**96.4***	20.4	3.2	18.8	**58.1***	38.4	43.0	3.5	**19.6***	101.2	194.1
	(3,10,25)	69.4	95.4*	**14.8**	**2.5**	16.7	52.9*	**28.9**	**37.3**	2.9	15.8*	**82.5**	**181.4**

Table 2. Statistics of the number of matches (m), the percentage of covered bases by matches (c) and the average island size (g) under mutations rates of 0.01, 0.05, 0.1. *In the strobemer protocols, only k_1 contributs to calculated covered positions, which is a lower bound of the actual covered positions. For the k-mers and the spaced k-mers the value is exact.

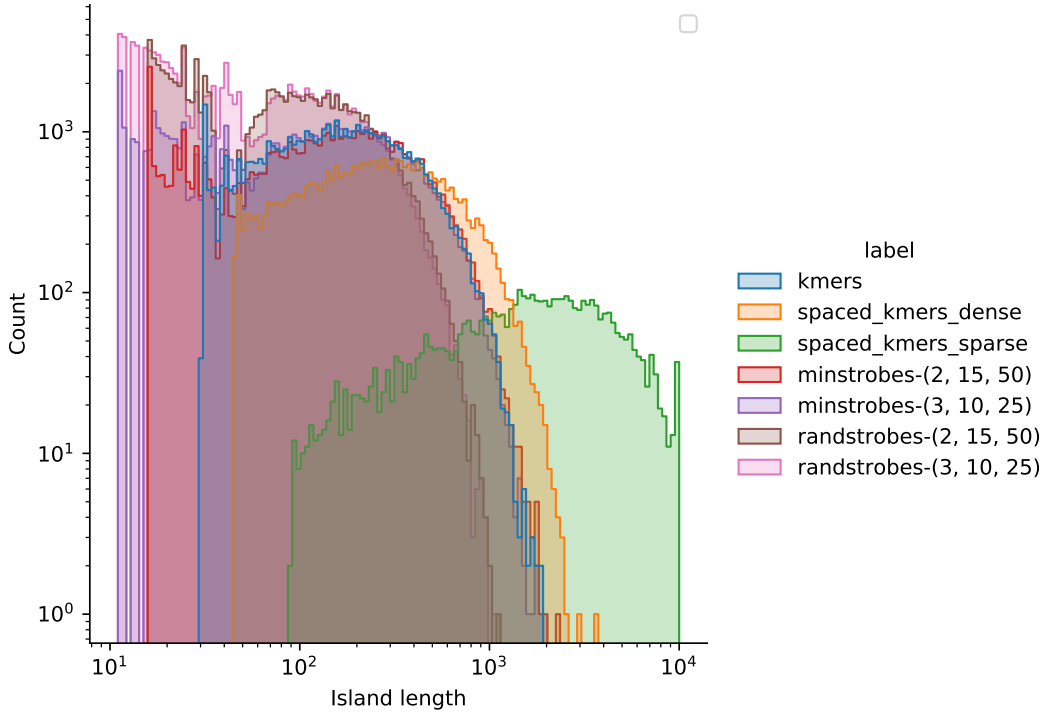

Fig. 3. Histogram of island lengths for the SIM-R experiment with mutation rate 0.1.

of matches, coverage, average island size, and island E-size under the different protocols (table 1). We repeat the experiment 1000 times to alleviate sample variance. For the spaced k-mer protocols, fixed positions are resimulated in each experiment. We refer to this as the SIM-C experiment (for simulation controlled). The spaced k-mer protocols offer an advantage over k-mers as they will match over some of the mutations that are substitutions, particularly the spaced-dense that has a lower window size than space-sparse and are therefore less affected by surrounding mutations. However, this particular experiment highlights the advantage of strobemers, which frequently produce matches between most

or all mutations. Furthermore, the experiment shows the difference in performance between minstrobes and randstrobes and their different parameterizations. The randstrobe protocols' matches cover the largest fraction of the sequences, and they also have the smallest average and expected island size (table 1). In this experiment, the randstrobe of order 3 produces the most favorable sequence matching result.

Random mutations. In our second experiment, we simulate a string of length 10,000nt and construct a second string by generating insertions, deletions or substitutions with equal probability of 1/3 each across the string with mutation rate $\mu \in 0.01, 0.05, 0.1$. This means that the positions for the mutations are randomly distributed. Each such simulation is replicated 1000 times to alleviate sample variation. We refer to this as the SIM-R experiment (for simulation random). In this scenario, spaced k-mer protocols generally perform worse than k-mers, with fewer matches, lower match coverage, and larger expected island size (table 2). When comparing the minstrobe protocols under the two given parametrizations, they have roughly the same performance as k-mers. Minstrobes have a slightly lower number of matches, match coverage, and roughly the same expected island size. However, the randstrobe protocols are also in this scenario significantly better at distributing matches across the sequences compared to all the other protocols. The randstrobe protocols have a substantially smaller expected island size under both parametrizations, which is an important aspect of sequence matching. Also, randstrobes of order 2 produce a higher lower-bound match coverage than k-mers, and randstrobes of order 3 produce a lower-bound match coverage close to the exact coverage of k-mers (table 2).

We also show the full distribution of island sizes in Fig. 3 for mutation rate 0.1 and in Fig. A.1 for mutation rate 0.01 and 0.05, which all illustrate the general trend in island sizes for the seven different protocols. For example, for a mutation rate of 0.1, we observe that the randstrobe protocols have roughly 1,000nt as the largest island size in our simulations, while k-mers have about 2,000nt Fig. 3 . The two minstrobe protocols follow the k-mer island distribution tightly Fig. 3 , while the spaced k-mer protocols perform worse.

Finally, we also computed minstrobes and randstrobes under a different window parametrization of (2,15,40) and (3,10,20). In this setting, the minstrobe protocols produce significantly better results than for the parametrization with window sizes 25 and 50 (table 3), and favorable expected island size compared to k-mers. However, the randstrobe protocols showed the opposite behavior and produced slightly worse results for this parametrization. This motivates further study on optimizing parameters for protocols.

Strobemer vs k-mer uniqueness. We also need to compare the confidence or uniqueness of a match. Strobemers offer more match flexibility, as they can preserve a match with indels in the sampled region. We refer to the ability for a protocol to match over indels as *flexible-position* protocols), contrary to k-mers and spaced k-mers (referred to as *fixed-position* protocols). Therefore, it is reasonable to assume that, for the same size k of extracted subsequence, the strobemer protocols will have lower uniqueness (precision) than k-mers and spaced k-mers. We study this by computing the percentage of unique k-mers, spaced k-mers, and strobemers on the three largest human chromosomes (Fig. 4). Similarly to the SIM-C and SIM-R experiments, for a k-mer size of k, we parametrize the strobemer protocols with $(n, k/n, 50/(n-1))$ for $n = 2, 3$ in order to have the same subsequence lengths to occupy the same amount of memory and can be stored and queried under the same conditions. Similarly, the spaced k-mers are parametrized by $L = 1.5k$ and $L = 3k$ and the positions are simulated as in previous experiments.

We observe that for the three fixed-position protocols, a larger window helps subsequence uniqueness. The spaced-sparse has the highest uniqueness across all the three chromosomes, followed by spaced-dense and finally the k-mers.

Contrary to our intuition, the strobemer protocols also offer higher precision than k-mers at sizes of $k \geq 24$ (Fig. 4). This may be due to the increased sampling window of the strobemer protocols compared to k-mers, as we observed was the trend between the three fixed sampling protocols. The strobemers, with their flexible-position sampling protocol, offer a uniqueness roughly similar to that of spaced-dense for $k \geq 24$ (Fig. 4). Out of the strobemer protocols evaluated here, strobemers of order 3 produce the highest percentage of unique matches for reasonably large subsequence lengths k. However, for $k = 18$, the strobemer protocols will be parametrized by $(2, 9, 50)$ and $(3, 6, 25)$, which with the flexible-position sampling appear too small to guarantee reasonable uniqueness on the largest human chromosomes.

Discussion

We have studied strobemers, an alternative sampling protocol to k-mers and spaced k-mers. We have experimentally demonstrated that strobemers, particularly randstrobes, efficiently reduce gaps between matches in sequences under different mutation rates. We demonstrated that randstrobes are less sensitive to the distribution and density of mutations across all SIM-C and SIM-R experiments, providing smaller islands and a higher percentage of covered bases (table 1 and table 2). Under a random mutation model (SIM-R), we observed that for a specific parametrization, the minstrobe protocols produce match coverage and island sizes on par or worse than k-mers (table 2). However, they improved over k-mers with different window size (table 3), suggesting the future study of optimal strobemer parameterization.

While spaced k-mer protocols offer a higher number of unique k-mers (specificity) on the larger human chromosomes (Fig. 4), they sacrifice sensitivity compared to k-mers in matches when indels are present. Our experiments show that strobemers, particularly randstrobes, do not sacrifice one for the other and achieve a favorable result to k-mers in both sensitivity and specificity for some commonly used sizes of k. We measured specificity as the percentage of unique subsequences being produced on the largest human chromosomes for several sizes of $k \geq 24$ (Fig. 4). We find the in-

DRAFT

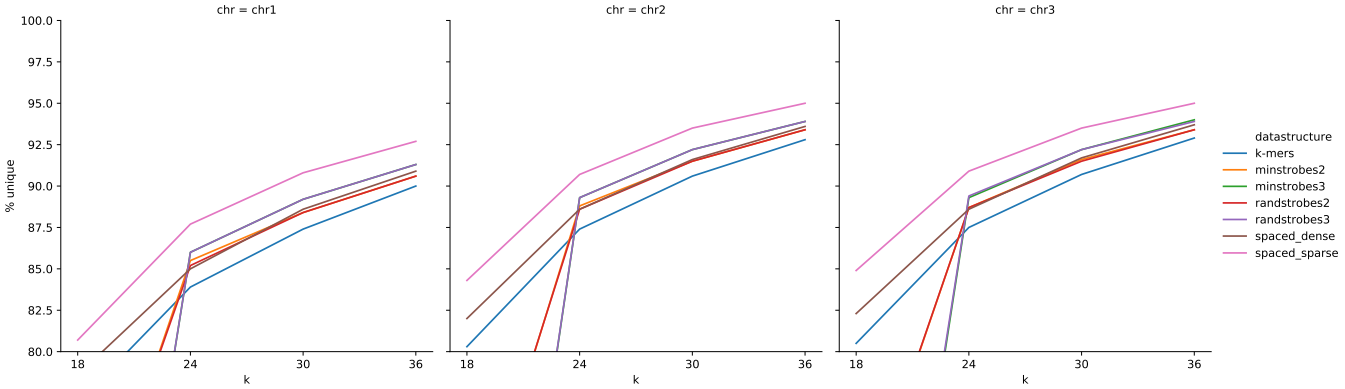

Fig. 4. The percent of unique k-mers, spaced k-mers, minstrobes and randstrobes (y-axis) on the three largest chromosomes (chr1-chr3) of the human genome for various sequence lengths of k (x-axis). Each panel shows a separate chromosome. For a given k in the plot, strobemers with $n = 2$ are computed with parameters $(2, k/2, 50)$ and strobemers with $n = 3$ are computed with parameters $(2, k/3, 25)$ so that the number of extracted nucleotides between the five methods are the same. Y-axes have been cut at 80% for illustration. The values for minstrobes and randstrobes with parameters $(3, 6, 25)$ are below 50% on all the three chromosomes. The value for minstrobes and randstrobes with parameters $(2, 9, 50)$ are below 70% on all the three chromosomes.

creased specificity to k-mers surprising, as the strobemers were designed to offer more flexibility in matches by allowing substitutions and indels between strobes. We hypothesize that the strobemers, due to their larger span than k-mers, offer more uniqueness in repetitive regions. This remains to be studied.

In all SIM-R experiments, k-mers provide a higher match count (table 2). However, due to the properties of k-mers, the matches are compactly clustered between mutations. Therefore, the number of matches is not always helpful as the matches may cluster due to local repeats. Randstrobes can offer more evenly distributed matches, higher match coverage, and higher uniqueness. These are features that are useful for several algorithms that require chains of matches between two sequences to be considered candidates for alignment or clustering, *e.g.*, as in (27, 29). Overall, the strobemers, particularly randstrobes, show a promising data structure for algorithms that rely on sequence comparison.

Similarly to k-mers, minstrobes and randstrobes can be subsampled as minimizers (25), syncmers (34), or any other thinning protocol that can be applied to k-mers. By stydying the mathematical properties of hashes and minimizers (32, 45), we may also find a effective subsampling techniques of strobemers.

With the methods used in this study, randstrobes are slower to generate than k-mers and minstrobes. In practice, generating randstrobes takes $O(|s|W)$ time compared to a runtime of $O(|s|)$ for k-mers, and an, in practice, runtime of $O(|s|)$ for minstrobes. By employing ideas like cyclic polynomial hash functions (42), we may come up with faster methods to generate them. In terms of memory requirements, for a given subsequence length k, k-mers and strobemers parametrized as $(n, k/n, \cdot)$ require the same memory to store as they can be treated as concatenated strings with the same length.

Future study of strobemers.

Parameterization. While our study provides an experimental evaluation of strobemers under some commonly used values of k and mutation rates, the statistics of strobemers remains to be explored. In (14), the authors derived the mean and variance of islands for k-mers and the number of mutated k-mers under given mutation rate. If we can derive analytic expressions for strobemers, it may suggest us how to optimize parameters of the strobemer protocols under various mutation rates, which will be useful for similarity comparison algorithms. Even without analytic expressions, we can evaluate the sizes on strobes and windows suitable for various mutation rates. This study tested four parameterizations of minstrobes and randstrobes, respectively, which produced different results. A grid search over parameter values may help us to better select parameterizations at various mutation rates.

Also, we may not limit the study of different parameters to selecting a single best size of strobes and window sizes. Instead, we could relax the constraint of equal size strobes and window sizes studied here to variable sizes over strobes and windows. As a start in this direction, we may derive more efficient parameter selection on window sizes by modeling the number of mutations after a certain number of nucleotides as a Poisson Process. Under such a model, the author hypothesizes that choosing larger window sizes downstream could be beneficial. This remains to be explored.

Construction, storing and queries. There are several aspects of construction, indexing, and storage of strobemers that could be explored. One such direction is to store and query the positions of the other strobes efficiently. This could give extra information about the coverage of matches and the span of matches across sequences (for sequence similarity applications). Another application is to efficiently index the data sets for abundance and presence of strobemers (46). For such applications, minstrobes may be advantageous due to the more frequently shared minimizers between the strobes. Finally, as we described earlier, the possibility of decreasing practical runtime for constructing randstrobes remains to be explored.

Span coverage for matching. Since strobemers are gapped sequences, it also motivates the study of match coverage and distribution of matches across regions (or positions) similarly

DRAFT

to what has been done for gapped experimental protocols such as mate-pair or paired-end reads (47). For example, one could compute the match coverage across a position or smaller region to decide the matches' confidence in the regions.

Generalization of strobemers. Finally, we can view the process of extracting a k-mer or a spaced k-mer at position i in a string s as applying a function $f(i, k, s)$ on s. Similarly, the process of extracting a strobemer from s can be viewed as applying the higher-order function $f'(i, k, s, h)$ on s where h is either some hash function or hash strategy (*e.g.*, iterative and conditionally dependent as in randstrobes). We demonstrated that applying f' on s is equally or more efficient than f for sequence matching for two different functions h (minstrobes and randstrobes), which poses the following question. Can we further improve over randstrobes for sequence matching by designing h differently?

Conclusions

We have presented strobemers as an alternative to k-mers for sequence comparison. Strobemers, particularly randstrobes, offer a more evenly distributed set of matches across sequences than k-mers, are less sensitive to the distribution of mutations across sequences, and can produce a higher match coverage under some parameterizations. We also showed that strobemers can offer higher uniqueness than k-mers at the same subsequence length for practical subsequence sizes. These features are useful for algorithms that perform sequence matching. Strobemers are also easy to both construct and query, making it a compelling alternative to k-mers. While we have empirically demonstrated the useful properties of strobemers, their statistical properties require further investigation.

ACKNOWLEDGEMENTS
We thank Camille Marchet, Rayan Chikhi, Paul Medvedev, Lior Pachter, Karel Břinda, Michael Hall, Michael Schatz, and Páll Melsted for their constructive comments and suggestions on an earlier draft of the manuscript. The computations were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) at Uppsala Multidisciplinary Center for Advanced Computational Science (UPPMAX).

Bibliography

1. Leena Salmela, Riku Walve, Eric Rivals, and Esko Ukkonen. Accurate self-correction of errors in long reads using de Bruijn graphs. *Bioinformatics*, 33(6):799–806, 06 2016. ISSN 1367-4803. doi: 10.1093/bioinformatics/btw321.
2. P A Pevzner. 1-tuple dna sequencing: computer analysis. *J Biomol Struct Dyn*, 7(1):63–73, Aug 1989. ISSN 0739-1102 (Print); 0739-1102 (Linking). doi: 10.1080/07391102.1989.10507752.
3. Rayan Chikhi and Paul Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, 06 2013. ISSN 1367-4803. doi: 10.1093/bioinformatics/btt310.
4. Derrick E. Wood and Steven L. Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):R46, 2014. doi: 10.1186/gb-2014-15-3-r46.
5. Samarth Rangavittal, Natasha Stopa, Marta Tomaszkiewicz, Kristoffer Sahlin, Kateryna D. Makova, and Paul Medvedev. Discovery: a classifier for identifying y chromosome sequences in male assemblies. *BMC Genomics*, 20(1):641, 2019. doi: 10.1186/s12864-019-5996-3.
6. Martin Steinegger and Johannes Söding. Clustering huge protein sequence sets in linear time. *Nature Communications*, 9(1):2542, 2018. doi: 10.1038/s41467-018-04964-5.
7. Brad Solomon and Carl Kingsford. Fast search of thousands of short-read sequencing experiments. *Nat Biotechnol*, 34(3):300–302, Mar 2016. ISSN 1546-1696 (Electronic); 1087-0156 (Print); 1087-0156 (Linking). doi: 10.1038/nbt.3442.
8. Robert S Harris and Paul Medvedev. Improved representation of sequence bloom trees. *Bioinformatics*, 36(3):721–727, 08 2019. ISSN 1367-4803. doi: 10.1093/bioinformatics/btz662.
9. Ryan P Abo, Matthew Ducar, Elizabeth P Garcia, Aaron R Thorner, Vanesa Rojas-Rudilla, Ling Lin, Lynette M Sholl, William C Hahn, Matthew Meyerson, Neal I Lindeman, Paul Van Hummelen, and Laura E MacConaill. Breakmer: detection of structural variation in targeted massively parallel sequencing data using kmers. *Nucleic Acids Res*, 43(3):e19, Feb 2015. ISSN 1362-4962 (Electronic); 0305-1048 (Print); 0305-1048 (Linking). doi: 10.1093/nar/gku1211.
10. Daniel S. Standage, C. Titus Brown, and Fereydoun Hormozdiari. Kevlar: A mapping-free framework for accurate discovery of de novo variants. *iScience*, 18:28–36, 2021/01/25 2019. doi: 10.1016/j.isci.2019.07.032.
11. Rob Patro, Stephen M Mount, and Carl Kingsford. Sailfish enables alignment-free isoform quantification from rna-seq reads using lightweight algorithms. *Nature Biotechnology*, 32(5):462–464, 2014. doi: 10.1038/nbt.2862.
12. Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic rna-seq quantification. *Nature Biotechnology*, 34(5):525–527, 2016. doi: 10.1038/nbt.3519.
13. Camille Marchet, Christina Boucher, Simon J. Puglisi, Paul Medvedev, Mikaël Salson, and Rayan Chikhi. Data structures based on k-mers for querying large collections of sequencing data sets. *Genome Research*, 2020. doi: 10.1101/gr.260604.119.
14. 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. *bioRxiv*, 2021. doi: 10.1101/2021.01.15.426881.
15. Bin Ma, John Tromp, and Ming Li. PatternHunter: faster and more sensitive homology search . *Bioinformatics*, 18(3):440–445, 03 2002. ISSN 1367-4803. doi: 10.1093/bioinformatics/18.3.440.
16. Karel Břinda, Maciej Sykulski, and Gregory Kucherov. Spaced seeds improve k-mer-based metagenomic classification. *Bioinformatics*, 31(22):3584–3592, 07 2015. ISSN 1367-4803. doi: 10.1093/bioinformatics/btv419.
17. Derrick E. Wood, Jennifer Lu, and Ben Langmead. Improved metagenomic analysis with kraken 2. *Genome Biology*, 20(1):257, 2019. doi: 10.1186/s13059-019-1891-0.
18. Uri Keich, Ming Li, Bin Ma, and John Tromp. On spaced seeds for similarity search. *Discrete Applied Mathematics*, 138(3):253 – 263, 2004. ISSN 0166-218X. doi: https://doi.org/10.1016/S0166-218X(03)00382-2.
19. Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, page 380â388, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1581134959. doi: 10.1145/509907.509965.
20. 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.
21. 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. ISSN 1557-8666 (Electronic); 1066-5277 (Print); 1066-5277 (Linking). doi: 10.1089/cmb.2012.0021.
22. Bo Liu, Hongzhe Guo, Michael Brudno, and Yadong Wang. deBGA: read alignment with de Bruijn graph-based seed and extension. *Bioinformatics*, 32(21):3224–3232, 07 2016. ISSN 1367-4803. doi: 10.1093/bioinformatics/btw371.
23. Andrei Broder. On the resemblance and containment of documents. 06 1997. doi: 10.1109/SEQUEN.1997.666900.
24. 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.
25. 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.
26. Brian D. Ondov, Todd J. Treangen, Páll Melsted, Adam B. Mallonee, Nicholas H. Bergman, Sergey Koren, and Adam M. Phillippy. Mash: fast genome and metagenome distance estimation using minhash. *Genome Biology*, 17(1):132, 2016. doi: 10.1186/s13059-016-0997-x.
27. 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.
28. Chirag Jain, Alexander Dilthey, Sergey Koren, Srinivas Aluru, and Adam M Phillippy. A fast approximate algorithm for mapping long reads to large reference databases. *J Comput Biol*, 25(7):766–779, Jul 2018. ISSN 1557-8666 (Electronic); 1066-5277 (Print); 1066-5277 (Linking). doi: 10.1089/cmb.2018.0036.
29. 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.
30. Kristoffer Sahlin, Botond Sipos, Phillip L. James, 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.
31. Konstantin Berlin, Sergey Koren, Chen-Shan Chin, James P Drake, Jane M Landolin, and Adam M Phillippy. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature Biotechnology*, 33(6):623–630, 2015. doi: 10.1038/nbt.3238.
32. Guillaume Marçais, Dan DeBlasio, and Carl Kingsford. Asymptotically optimal minimizers schemes. *Bioinformatics*, 34(13):i13–i22, 06 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty258.
33. 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.
34. Robert C. Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *bioRxiv*, 2020. doi: 10.1101/2020.09.29.319095.

DRAFT

35. S F Altschul, T L Madden, A A Schäffer, J Zhang, Z Zhang, W Miller, and D J Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–3402, Sep 1997. ISSN 0305-1048 (Print); 1362-4962 (Electronic); 0305-1048 (Linking). doi: 10.1093/nar/25.17.3389.
36. Laurent Noé and Gregory Kucherov. Improved hit criteria for dna local alignment. *BMC Bioinformatics*, 5(1):149, 2004. doi: 10.1186/1471-2105-5-149.
37. Yanni Sun and Jeremy Buhler. Designing multiple simultaneous seeds for dna similarity search. *Journal of Computational Biology*, 12(6):847–861, 2005. doi: 10.1089/cmb.2005.12.847. PMID: 16108721.
38. Jeremy Buhler, Uri Keich, and Yanni Sun. Designing seeds for similarity search in genomic dna. *Journal of Computer and System Sciences*, 70(3):342 – 363, 2005. ISSN 0022-0000. doi: https://doi.org/10.1016/j.jcss.2004.12.003. Special Issue on Bioinformatics II.
39. Nan Du, Jiao Chen, and Yanni Sun. Improving the sensitivity of long read overlap detection using grouped short k-mer matches. *BMC Genomics*, 20(2):190, 2019. doi: 10.1186/s12864-019-5475-x.
40. Chen-Shan Chin and Asif Khalak. Human genome assembly in 100 minutes. *bioRxiv*, 2019. doi: 10.1101/705616.
41. Anna Ritz, Ali Bashir, and Benjamin J. Raphael. Structural variation analysis with strobe reads. *Bioinformatics*, 26(10):1291–1298, 04 2010. ISSN 1367-4803. doi: 10.1093/bioinformatics/btq153.
42. Hamid Mohamadi, Justin Chu, Benjamin P. Vandervalk, and Inanc Birol. ntHash: recursive nucleotide hashing. *Bioinformatics*, 32(22):3492–3494, 07 2016. ISSN 1367-4803. doi: 10.1093/bioinformatics/btw397.
43. Miika Leinonen and Leena Salmela. Extraction of long k-mers using spaced seeds, 2020.
44. Steven L. Salzberg, Adam M. Phillippy, Aleksey Zimin, Daniela Puiu, Tanja Magoc, Sergey Koren, Todd J. Treangen, Michael C. Schatz, Arthur L. Delcher, Michael Roberts, Guillaume Marçais, Mihai Pop, and James A. Yorke. Gage: A critical evaluation of genome assemblies and assembly algorithms. *Genome Research*, 22(3):557–567, 2012. doi: 10.1101/gr.131383.111.
45. Yaron Orenstein, David Pellow, Guillaume MarÃ§ais, Ron Shamir, and Carl Kingsford. Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing. *PLOS Computational Biology*, 13(10):1–15, 10 2017. doi: 10.1371/journal.pcbi.1005777.
46. Camille Marchet, Zamin Iqbal, Daniel Gautheret, Mikaël Salson, and Rayan Chikhi. REINDEER: efficient indexing of k-mer presence and abundance in sequencing datasets. *Bioinformatics*, 36(Supplement_1):i177–i185, 07 2020. ISSN 1367-4803. doi: 10.1093/bioinformatics/btaa487.
47. Kristoffer Sahlin, Mattias Frånberg, and Lars Arvestad. Structural variation detection with read pair information: An improved null hypothesis reduces bias. *Journal of Computational Biology*, 24(6):581–589, 2017. doi: 10.1089/cmb.2016.0124. PMID: 27681236.

DRAFT

Supplementary Note A: Figures

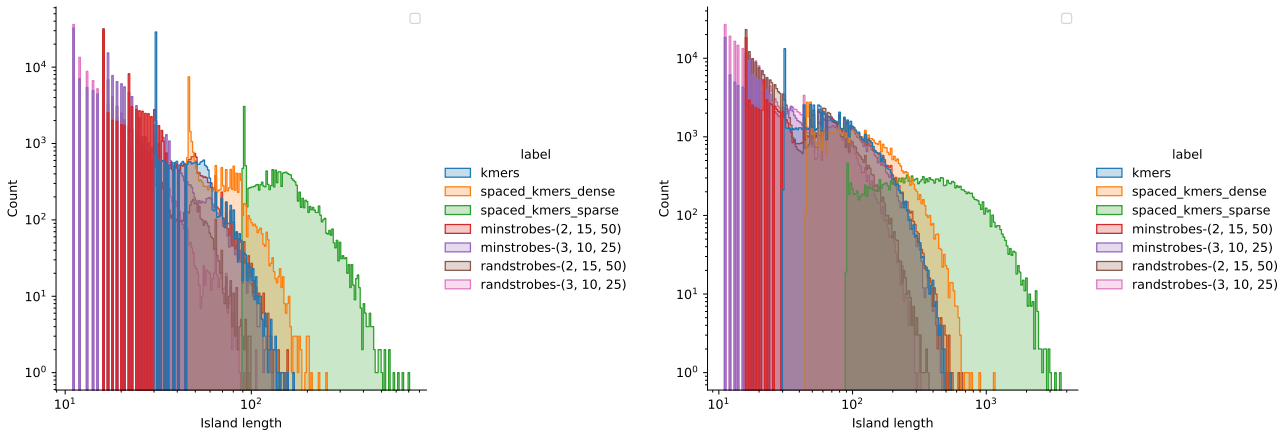

Fig. A.1. Histograms of island lengths for the SIM-R experiments for mutation rate 0.01 (a) and 0.1 (b).

Supplementary Note B: Tables

		SIM-R											
		0.01				0.05				0.1			
		m	c	g	E	m	c	g	E	m	c	g	E
minstrobe	(2,15,40)	72.8	91.4*	24.1	6.8	20.4	45.5*	51.2	65.4	4.0	13.2*	154.7	292.3
	(3,10,20)	70.7	88.6*	17.8	6.1	18.0	39.9*	39.2	61.2	3.2	10.5*	128.5	278.4
randstrobe	(2,15,40)	72.8	95.2*	21.1	4.2	20.5	54.1*	42.3	48.8	3.9	17.6*	114.3	212.7
	(3,10,20)	70.7	94.2*	15.1	3.2	18.0	50.8*	30.2	41.1	3.2	15.2*	86.7	189.4

Table 3. Performance of strobemer protocols under a different window parameterization. Statistics of the number of matches (m), the percentage of covered bases by matches (c) and the average island size (g) under mutations rates of 0.01, 0.05, 0.1. *In the strobemer protocols, only k_1 contributs to calculated covered positions, which is a lower bound of the actual covered positions. For the k-mers and the spaced k-mers the value is exact.

DRAFT