Syncmers are more sensitive than minimizers for selecting conserved *k*-mers in biological sequences

Robert C. Edgar
Independent Investigator
Corte Madera, California, USA.
robert@drive5.com

Abstract

Minimizers are widely used to select subsets of fixed-length substrings (*k*-mers) from biological sequences in applications ranging from read mapping to taxonomy prediction and indexing of large datasets. Syncmers are an alternative method for selecting a subset of *k*-mers. Unlike a minimizer, a syncmer is identified by its *k*-mer sequence alone and is therefore synchronized in the following sense: if a given *k*-mer is selected from one sequence, it will also be selected from any other sequence. Bounded syncmers are defined by a small and fast function of the *k*-mer sequence which exploits correlations between overlapping *k*-mers to guarantee that at least one syncmer must appear in a window of predetermined length, and therefore comprise a universal hitting set which does not require a precomputed lookup table. Bounded syncmers are shown to be unambiguously superior to minimizers because they achieve both lower density and better conservation in mutated sequences.

Introduction

K-mers, submers and minimizers

Next-generation sequencing (NGS) has enabled dramatic advances in fields ranging from human functional genomics (Morozova and Marra, 2008) to microbial metagenomics (Gilbert and Dupont, 2011), generating vast amounts of data currently at petabase scale and increasing exponentially (Schmidt and Hildebrandt, 2017). Computational methods for analysis of NGS data are often based on *k*-mers, short subsequences of fixed length *k*. The number of *k*-mers in an index is often very large, motivating methods for selecting subsets as a time and space optimization. I use the term *submers* for a method designed to select a common subset of *k*-mers from similar sequences. The canonical example of submers is *minimizers* (Roberts *et al.*, 2004). Minimizers are used in applications including read mapping (Li, 2018), taxonomy prediction (Wood *et al.*, 2019) and assembly (Sommer *et al.*, 2007; Ye *et al.*, 2012). Minimizers are defined by a choice of *k*, a window length *w*, and a *coding* function which assigns an integer value (*code*) to a sequence of length *k*. A coding function defines a partial order on the *k*-mers; not necessarily a total order because a coding function may assign multiple *k*-mers the same code. The minimizer in a given window is a *k*-mer with smallest code value (ties may be handled in different ways), and the set of minimizers for a sequence is constructed by taking the union of minimizers over all windows. A simple and popular coding function is to consider letters in the nucleotide or amino acid alphabet to be base-4 or base-20 digits respectively (*lexicographic coding*). Often the coding function hashes the lexicographic code to improve the pseudo-randomness of the codes and/or to reduce the range of codes, e.g. for hash table indexes, especially with larger values of *k* where a table of size 4^k or 20^k would exceed available memory. The read mapper Winnowmap (Jain *et al.*, 2020) introduced the concept of *weighted minimizers* which assigns a weight to each *k*-mer biasing its probability of being selected in the event of ties. For given *k* and *w*, the coding function can be optimized with the goal of minimizing the number of minimizers (Zheng *et al.*, 2020; Marçais *et al.*, 2018). Another related approach is a *universal hitting set* (*UHS*) (Orenstein *et al.*, 2016, 2017), a pre-computed set of *k*-mers such that any sequence of length *w* must contain at least one member.

Submer conservation

Methods based on *k*-mers are generally used for comparison of similar sequences where differences may have been introduced by biological mutations and/or sequencing error. Detection of similarity by a *k*-mer requires that at least *k* letters and perhaps more are exactly conserved. A given sequence position is covered by *k* distinct *k*-mers, all of which are changed if that position is mutated. Therefore, it is desirable to select submers with as few overlaps as possible because a difference in an overlapped position disrupts multiple submers. Overlaps are minimized by uniform spacing between submers. Thus, for a given difference rate, a measure of spacing uniformity will correlate with the fraction of submers conserved. Uniform spacing also maximizes the number of letters covered by a given number of submers. Submers would therefore ideally be uniformly spaced, but in the case of minimizers they are not. The distance between adjacent pairs of minimizers ranges from one to a maximum of w. Each possible distance occurs with approximately the same frequency in a random sequence because the probability of introducing a new minimizer when the window is moved by one position is constant. Thus, for example, $1/w$ of adjacent pairs of minimizers have distance 1 (consecutive positions). Such pairs share $k - 1$ letters and two or more minimizers will be disrupted by a mutation in any of those positions. Further, a minimizer is deleted by a mutation in a position outside of its *k*-mer but within its window if that mutation causes some other *k*-mer to have a smaller code. Thus, a minimizer is guaranteed to be conserved only if its *k*-mer sequence plus left- and right-flanking sequences of length $w - 1$ are conserved.

Methods

Coding function

Nucleotide *k*-mers are converted to integers by lexicographic coding. Extension to amino acid sequences is straightforward but not considered further in this report. Unless otherwise stated, lexicographic codes are hashed by murmur64 (https://en.wikipedia.org/wiki/MurmurHash).

Minimizers

Definition

A k-mer is identified as a minimizer is if it has the smallest code value in any of the windows of length $k + w - 1$ letters which cover it.

Compression factor

Let the *compression factor* (c) be the number of k-mers divided by the number of submers, so $c \geq 1$ and larger c indicates a smaller subset. This is the inverse of density as defined elsewhere in the literature (number of submers divided by number of k-mers). For minimizers, c can be estimated as follows, assuming code values are independent and identically distributed (IID) and there are no ties in a window. Consider a pair of adjacent windows in a random sequence. Sliding one position to the right, one k-mer (κ_1) is discarded from the first window and one (κ_2) is added to the second. The minimizer in the second window is different if κ_1 or κ_2 has the smallest code over both windows; otherwise the minimizer of both windows is found in their intersection and does not change. There are $w + 1$ k-mers in the two windows combined, and the probability that a given one of these has the smallest code is therefore $P_m = 1/(w + 1)$. Thus, the probability that a new minimizer is introduced by sliding the window one position is $2\ P_m = 2/(w + 1)$ and hence

$$c_{minimizer} \approx (w + 1)/2. \qquad \text{(Eq.1)}$$

The maximum distance between consecutive minimizers, measured as the difference in their start coordinates, is w, as shown by the following argument. Consider a window of w consecutive k-mers where the minimizer is the first k-mer. Slide the window one position to the right. This must introduce a new minimizer. The maximum distance from the previous minimizer is obtained when the newly-created k-mer at the right-hand end of the window is the new minimizer, and this is at distance w from the previous minimizer which was lost at the left-hand end.

Syncmer

A *syncmer* is a submer chosen by a rule that considers only the k-mer sequence. A minimizer is a submer but not a syncmer because the rule considers flanking sequence in addition to the k-mer itself. A syncmer rule must be a function of its sequence that can be calculated on the fly. Thus, previously published UHS implementations do not define syncmers because the rule is computationally expensive and the set must be precomputed. A syncmer rule defines a UHS if it can be shown that every window of a predetermined length contains at least one syncmer.

Mincode syncmers

Definition

Mincode syncmers are defined by a compression factor $c > 1$ which sets a minimum value for the code, as follows: a k-mer κ is a mincode syncmer if $\text{code}(\kappa) \geq H/c$ where H is the maximum possible code.

Distance distribution

Assuming that k-mer codes are IID, the distance distribution for mincode syncmers can be calculated as follows. The probability that a given k-mer is selected is $1/c$. Thus, the probability that the distance is one (immediately adjacent) is $P(1) = 1/c$, the probability of distance two is $P(r{=}2) = (1 - P(1))P(1)$ and the probability of distance r is

$$P_{mincode}(r) = (1/c)\,(1 - 1/c)^{r-1}. \qquad \text{(Eq.2)}$$

For example, with c=2 this gives $P(1)$=0.5, P(2)=0.25, P(3)=0.125, and with t=3, $P(1)$=0.33, $P(2)$=0.22, $P(3)$=0.15. For any $c > 1$, the maximum is $P(1)$ and the probability of longer distances falls exponentially with r. With larger c, $(1 - 1/c)$ is closer to one and the exponential falls more slowly. The maximum at r=1 implies that overlaps are common, causing multiple submers to be disrupted by a single mutation. The slow fall of the exponential tail will cause longer gaps between covered positions. Thus, mincode syncmers have a worse spacing distribution than minimizers and are generally not recommended for practical use. Here, they serve as a "control" for comparison with better rules.

Bounded syncmers

Definition

Bounded syncmers are parameterized by k, a substring length $s < k$, and a coding function. Given a string S, let $\mu(S, s)$ be the s-mer substring with smallest code value, taking the first in left-to-right order if there is a tie. A k-mer κ is a bounded syncmer if and only if $\mu(\kappa, s)$ is the first or last s-mer in κ.

Window size

Consider a window W containing $w = (k - s)$ k-mers. Let $\sigma^+ = \mu(W, s)$, i.e. the first s-mer with smallest word value in W. For any s-mer σ in W, there is a k-mer κ such that σ is at the beginning or end of κ (Fig. 1). Therefore σ^+ is at the start or end of a k-mer in W; call this k-mer κ^+. The s-mers in κ^+ are a subset of s-mers in W, and therefore σ^+ has the smallest word value in κ^+, i.e. $\mu(\kappa^+, s) = \sigma^+ = \mu(W, s)$. Therefore, κ^+ is a syncmer, and all windows of length $w = (k - s)$ k-mers contain at least one bounded (k, s) syncmer. This proof demonstrates that bounded syncmers have the motivating property of minimizers and UHSs: every window of a predetermined length necessarily contains at least one submer.

Compression factor

A k-mer has $n = (k - s + 1)$ s-mer substrings. Under the approximation that that s-mer code values are IID, the probability a given s-mer is the smallest is $1/n$ and the probability that the first or last is the smallest is $2/n$. Then the probability that a given k-mer is a syncmer is $2/(k - s + 1) = 2/(w + 1)$, and the compression factor for a bounded syncmer is

$$c_{bounded} \approx (w + 1)/2 = (k - s + 1)/2. \qquad \text{(Eq.3)}$$

This approximation is reasonable when $4^s \gg k$, but breaks down when $4^s \lesssim k$. For example, if s=1, then a k-mer is a syncmer if the first or last letter has lowest code value (call it A), which occurs with probability 1/4 (first letter is A) + 1/4 (last letter is A) − 1/16 (subtract double-counting when both first and last letters are A) = 0.56 for any $k > 1$, and this is an under-estimate because there are additional syncmers containing no As. Thus, with s=1 the compression factor is <1/0.56 = 1.79 for any k.

Maximum distance

By the same argument as minimizers, the maximum distance between two bounded syncmers is $w = (k - s)$, and the minimum overlap between adjacent syncmers is therefore s.

Down-sampled syncmers

Definition

The compression factor of a bounded syncmer is limited by the requirement that $s < k$. To address this, down-sampled syncmers are defined by introducing an additional parameter $d > 1$ which enables arbitrarily large compression factors. A k-mer κ is a down-sampled syncmer if κ is a bounded syncmer and κ is a mincode syncmer with factor d.

Compression factor

With a well-chosen coding function, the probability that a k-mer is a bounded syncmer is effectively independent of the probability that it is a mincode syncmer. The probabilities are therefore multiplicative and the compression factor of a down-sampled syncmer is

$$c_{down\text{-}sampled} = d\, c_{bounded} \approx d\,(k - s + 1)/2. \qquad \text{(Eq.4)}$$

Distance distribution

Down-sampled syncmers are a subset of mincode syncmers with compression factor d, and therefore by Eq.3

$$P_{downsampled}(r) > (1/d)\,(1 - 1/d)^{r-1}. \qquad \text{(Eq.5)}$$

The value of d required to get a given compression factor c is smaller for down-sampled syncmers than for mincode syncmers, and the spacing distribution may therefore be better (i.e., decrease faster as a function of r).

Submer properties on random sequences

Submer properties including the compression factor (c) were measured on a random sequence R of length 10^6 letters {ACGT}. The compression factor was measured as the

number of k-mers divided by the number of submers. Other metrics are *depth*, the mean number of k-mers covering a letter and f_1, the fraction of letters covered by at least one submer.

Submer conservation

Conservation was measured by introducing random substitutions into R at a given rate, giving a mutated sequence R'. The mutation rate is expressed as percentage identity of R to R', e.g. rate 0.2 is identity 80%. A submer is conserved if the k-mer at a given position is unchanged (no substitutions) and this k-mer is selected as a submer in both R and R'. The fraction of conserved submers (*FSid*) at identity *id* (e.g., FS80) is defined as the number of conserved submers divided by the number of submers selected in R. The fraction of letters which are both indexed and conserved (*FLid*, e.g. FL80) is defined as the number of positions appearing in conserved submers divided by the length of R. If a given position appears in multiple conserved submers, it is counted only once. This definition of conserved letters is designed to quantify the fraction of the mutated sequence covered by alignment seeds.

Dictionary size

Let the *dictionary* be the set of all possible submers with given parameters. If k is not too large, it is tractable to enumerate all k-mers and count how many are syncmers with a given s, giving the dictionary size. For bounded syncmers, the dictionary is a UHS and the dictionary size is the number of k-mers in the UHS. I measured dictionary sizes for some representative parameters using a lexicographic coding function without hashing as this enables a fast and simple implementation using bit-wise operations. While lexicographic coding has disadvantages in practice, e.g. A-rich k-mers have systematically smaller lexicographic codes causing composition bias, the dictionary size is not sensitive to the choice of coding function and here it was convenient to use lexicographic coding. For minimizers, the size of the dictionary can be estimated as follows. The position of a k-mer in a window can be chosen in w different ways. There are $(w - 1)$ letters in the rest of the window. The number of distinct window sequences containing a given k-mer is therefore $N = w4^{(w-1)}$. Assuming that the probability a k-mer is a minimizer in a given window is $1/w$,

the probability that it is not a minimizer in any possible window is $P_{not} = (1 - 1/w)^N$. For example, with w=10 (as for minimap2), N=10 × 4^9 = 2.6 × 10^6 and $P_{not} = 0.9^N$ which is astronomically small because N is very large. Thus, for many practically useful values of w almost all k-mers are potential minimizers and the dictionary size is close to 4^k.

Evaluation on whole-genome alignment

I implemented a pair-wise whole-genome aligner as follows. Submers and their positions in two genomes are identified and indexed in a hash table. For each pair of submers with the same sequence, one in each genome, an ungapped seed-and-extend alignment (Altschul *et al.*, 1990) is constructed. The seed alignment implied by the matching submers is extended until (maximum score) − (current score) > X, where X is a fixed parameter, and the alignment with the maximum score is reported. After selecting submers, this algorithm has only three free parameters relative to the match score (1): mismatch score (−3), X (16) and minimum score to report an alignment (100). The minimum alignment score is set to a high value to suppress short and/or low-identity alignments where homology is ambiguous. Submer sensitivity was compared by measuring the aligned fraction (*AF*) of the input sequences obtained by different submer types. AF is calculated as $(n_1/L_1 + n_2/L_2)/2$ where n_1 is the number of bases in the first genome which appear in an alignment, L_1 is the length of the first genome, and similarly for the second genome. The value of AF ranges from 0 (no alignments) to 1 (all bases in both genomes appear in at least one alignment). Gapped alignments were not considered because alignment length is sensitive to gap penalty values, and the difference in AF between two submer types may therefore vary with gap penalty choice. With gapless alignments, if one submer type has substantially higher AF, it Is reasonable to conclude that this submer is more sensitive to homologous seeds. If two submer types have similar AF, then the result is inconclusive because it cannot be ruled out that one submer type would find more low-scoring alignments that are in fact homologous.

Genome pairs

Whole-genome alignments were assessed using the bacterial assemblies shown in Table 1. Pair-wise average nucleotide identity (ANI) and aligned fraction (AF) were measured from MEGABLAST (Morgulis *et al.*, 2008) alignments having E-values ≤ 10^{-9} and length ≥100.

Gap-open and -extension penalties were set to high values to ensure that alignments were gapless. I manually selected pairs of genomes with a representative range of ANIs, preferring pairs with higher AF. These pairs were used to measure submer sensitivity in whole-genome alignment.

Representative parameters

To select parameters representative of those used in practice, I chose minimizers (k=15, w=10) and (k=31, w=15) used by minimap2 (Li, 2018) and Kraken v1 (Wood and Salzberg, 2014), respectively. Comparable syncmer parameters were identified as those giving the closest compression factors.

Sparse submers

For a given k, the relative improvement in conservation of syncmers over minimizers is expected to increase as the minimizer window length increases, i.e. with higher compression / smaller depth, because a mutation anywhere in the window may delete a minimizer while a syncmer is deleted only by a mutation inside its k-mer. To demonstrate this, I compared (k=31, w=100) minimizers with (k=31, s=6, d=4) down-sampled syncmers; parameters were hand-tuned to yield similar depth. These submers are sparse, i.e. have depth < 1, leaving some letters uncovered. Down-sampling is required because bounded syncmers cannot be sparse (the minimum overlap between adjacent syncmers is s).

Results

Comparison with minimap2 and Kraken minimizers

Table 2 (A) shows properties for minimap2 minimizers and the closest comparable bounded syncmers with the same value of k. Minimap2 minimizers (k=15, w=10) have compression factor c=5.50; the closest compression factor (c=5.43) for a bounded syncmer is with s=5. With s=5, syncmers have substantially better conservation than minimizers: 24% more conserved submers at 80% identity and 14% more at 90% identity. Syncmers have 10% more conserved letters at 80% identity and 4% more conserved letters at 90% identity. Similar improvements are seen in Table 2 (B) with Kraken minimizers (k=31, w=15), where s=16 syncmers have the same compression factor (c=8.00) to three

significant figures with 27% more conserved submers and 10% more conserved letters at 80% identity. Higher compression factors are achieved by syncmers while still having higher conservation than minimizers as shown by Table 2 for (k=15, s=4) and (k=31, s=15) syncmers respectively. Fig. 2 shows conservation as a function of mutation rate in the range 80% to 90% identity, showing consistently higher conservation for syncmers with parameters chosen to achieve the closest possible compression factors. Fig. 3 shows the frequency distribution for distances between consecutive submers. As expected, minimizers have a uniform distribution, which implies non-uniform spacing and hence reduced conservation of closely-spaced minimizers where a single mutation can disrupt several k-mers. By contrast, the distance distribution for syncmers is strongly peaked at distance $k - s - 1$, indicating a more uniform tiling of the sequence. This peak occurs because if the smallest s-mer σ is at position 0, causing the k-mer to be a syncmer, then if the k-mer is slid $k - s - 1$ positions to the left, then σ is the last s-mer and this position is a syncmer unless a smaller s-mer occurs in the remaining sequence. Thus, syncmers maintain an approximately even tiling of a random sequence, and in this sense are synchronized within a single sequence as well as between two similar sequences.

Whole-genome alignment

Table 3 shows results for whole-genome alignment on 19 pairs of bacterial assemblies using (k=15, w=10) minimizers and (k=15, s=5) bounded syncmers. On 15/19 of these pairs (79%), syncmers had higher AF (more aligned bases, indicating higher sensitivity), there was one tie, and on two pairs minimizers had 0.1% higher AF. As expected from the earlier results, syncmers show a strong tendency to have better sensitivity at lower identities. The AFs obtained with submers at low identities are much lower than MEGABLAST, indicating that many alignable segments of length ≥100 do not contain a conserved submer.

Sparse submers

Results for sparse submers are shown in Table 4. Syncmers have 3% better compression and substantially better conservation on the measured identities (85%, 90% and 95%), ranging from 26% higher FL95 to 149% higher FS90. Syncmers are superior by all metrics

except f_1, the fraction of letters covered by at least one submer. However, bounded syncmers have higher conservation despite having lower f_1.

Dictionary size

Results are shown in Table 5, showing that for the selected bounded syncmer parameters, $c = (w + 1)/2$ (Eq.3) gives a good estimate of the compression factor of dictionary and of a random string. A UHS with window w requires at least one k-mer per window, giving an upper bound $c \leq w$. Thus, in these examples the UHS defined by bounded syncmers is within a factor of ~2 of the best possible compression factor, i.e. smallest possible set.

Comprehensive parameter sweep

Supplementary Table 1 gives quality metrics for a wide range of submers, enabling users to choose parameters giving a preferred trade-off between compression factor and conservation.

Discussion

Computational efficiency

Scanning a sequence for syncmers should have comparable cost to scanning for minimizers as the algorithms are similar. With minimizers, the algorithm must compute the code value of every k-mer and track the smallest-valued k-mer in a window of length w. For bounded syncmers, the algorithm must compute the code value of every s-mer and track the smallest value s-mer in a window of length k. The code value of a k-mer must also be computed if the smallest s-mer occurs at the start or end of the window. With current computer architectures, the cost of scanning for syncmers or minimizers in a production-quality implementation is likely to be small compared to the cost of reading the sequences from storage or post-processing (building an index, extending seed alignments etc.).

Sycnmers do not have edge bias

A k-mer at the start or end (edge) of a sequence appears in only one window, and thus has a probability $1/w$ of being a minimizer which is roughly half the probability $2/(w + 1)$ for a k-mer far enough from the edge that it appears in w windows (see Eq.1). The probability

increases until the w'th k-mer has no edge bias. This issue can be significant in practice for short sequences such as next-generation reads, e.g. for identifying overlaps in assembly. This was noted in (Roberts *et al.*, 2004), where it was proposed that the problem can be addressed by introducing a special case at sequence terminals they called *end-minimizers*. This problem does not arise with syncmers and therefore no special case is needed.

Syncmers are decisively better than previous UHS implementations

In previously published implementations, it is computationally expensive to check whether a k-mer is in a UHS, and the entire set must therefore be pre-computed and stored in a lookup table. For large k and practically useful w, the table may be prohibitively large for high-throughput applications (Ekim *et al.*, 2020). Even if the table can be calculated and stored in a reasonable amount of RAM, table lookups will often cause cache misses and this may be expensive because the time required to fetch a page into cache is equivalent to hundreds or thousands of instructions with current processors (Kowarschik and Weiß, 2003). Bounded syncmers define a UHS for which membership can be tested on the fly without a lookup table for any choice of k and s. UHS sets previously considered in the literature are often sparse, i.e. have $w > k$ so that some letters are not covered. Bounded syncmers cannot be sparse, so in this case cannot implement a strictly defined UHS. However, down-sampled syncmers represent a good approximation to a UHS which retain the advantage of efficient on-the-fly identification.

Conclusion

Compared to previous implementations of universal hitting sets, syncmers can be implemented over a much wider range of parameters and are more efficient in both space and time. Compared to minimizers, syncmers achieve more uniform sequence tiling, higher compression and greater conservation over a wide range of practically useful k values and compression factors, and are therefore unambiguously superior.

References

Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. (1990). Basic local alignment search tool. *J Mol Biol* **215**: 403–410.

Ekim B, Berger B, Orenstein Y. (2020). A randomized parallel algorithm for efficiently finding near-optimal universal hitting sets. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. e-pub ahead of print, doi: 10.1007/978-3-030-45257-5_3.

Gilbert JA, Dupont CL. (2011). Microbial Metagenomics: Beyond the Genome. *Ann Rev Mar Sci* **3**: 347–371.

Jain C, Rhie A, Zhang H, Chu C, Walenz BP, Koren S, *et al.* (2020). Weighted minimizer sampling improves long read mapping. *Bioinformatics*. e-pub ahead of print, doi: 10.1093/bioinformatics/btaa435.

Kowarschik M, Weiß C. (2003). An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms. e-pub ahead of print, doi: 10.1007/3-540-36574-5_10.

Li H. (2018). Minimap2: Pairwise alignment for nucleotide sequences. *Bioinformatics* **34**: 3094–3100.

Marçais G, Deblasio D, Kingsford C. (2018). Asymptotically optimal minimizers schemes. In: *Bioinformatics*. e-pub ahead of print, doi: 10.1093/bioinformatics/bty258.

Morgulis A, Coulouris G, Raytselis Y, Madden TL, Agarwala R, Schäffer AA. (2008). Database indexing for production MegaBLAST searches. In: *Bioinformatics*. e-pub ahead of print, doi: 10.1093/bioinformatics/btn322.

Morozova O, Marra MA. (2008). Applications of next-generation sequencing technologies in functional genomics. *Genomics* **92**: 255–264.

Orenstein Y, Pellow D, Marçais G, Shamir R, Kingsford C. (2016). Compact universal k-mer hitting sets. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. e-pub ahead of print, doi: 10.1007/978-3-319-43681-4_21.

Orenstein Y, Pellow D, Marçais G, Shamir R, Kingsford C. (2017). Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing. *PLoS Comput Biol*. e-pub ahead of print, doi: 10.1371/journal.pcbi.1005777.

Roberts M, Hayes W, Hunt BR, Mount SM, Yorke JA. (2004). Reducing storage requirements for biological sequence comparison. *Bioinformatics*. e-pub ahead of print, doi: 10.1093/bioinformatics/bth408.

Schmidt B, Hildebrandt A. (2017). Next-generation sequencing: big data meets high performance computing. *Drug Discov Today*. e-pub ahead of print, doi: 10.1016/j.drudis.2017.01.014.

Sommer DD, Delcher AL, Salzberg SL, Pop M. (2007). Minimus: A fast, lightweight genome assembler. *BMC Bioinformatics*. e-pub ahead of print, doi: 10.1186/1471-2105-8-64.

Wood DE, Lu J, Langmead B. (2019). Improved metagenomic analysis with Kraken 2. *Genome Biol*. e-pub ahead of print, doi: 10.1186/s13059-019-1891-0.

Wood DE, Salzberg SL. (2014). Kraken: Ultrafast metagenomic sequence classification using exact alignments. *Genome Biol*. e-pub ahead of print, doi: 10.1186/gb-2014-15-3-r46.

Ye C, Ma ZS, Cannon CH, Pop M, Yu DW. (2012). Exploiting sparseness in de novo genome assembly. *BMC Bioinformatics*. e-pub ahead of print, doi: 10.1186/1471-2105-13-S6-S1.

Zheng H, Kingsford C, Marçais G. (2020). Improved design and analysis of practical minimizers. *bioRxiv*. e-pub ahead of print, doi: 10.1101/2020.02.07.939025.

Figures and tables

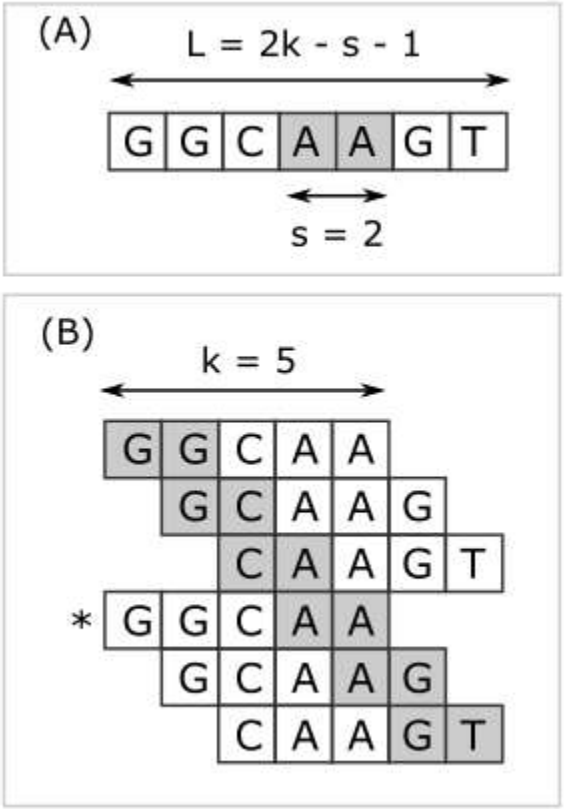

Figure 1. Bounded syncmers. Construction of k=5, s=2 bounded syncmers with lexicographic coding. Consider a window of length $L = 2k - s - 1 = 7$ with the sequence shown in panel (A). The smallest s-mer is AA (gray background). Every s-mer in this sequence is at the start or end of a k-mer, as shown in panel (B) where every s-mer is highlighted in the corresponding k-mer. Therefore, regardless of which s-mer has the smallest value, there is a k-mer in the window for which this s-mer appears at the first or last position. In this example, AA appears at the end of GGCAA, marked with an asterisk (*), and GGCAA is therefore a syncmer. This shows that every window of length L must contain at least one syncmer. Note that while flanking sequence is shown in the figure, GGCAA is recognized as a syncmer from its sequence alone because its smallest 2-mer appears at the end.

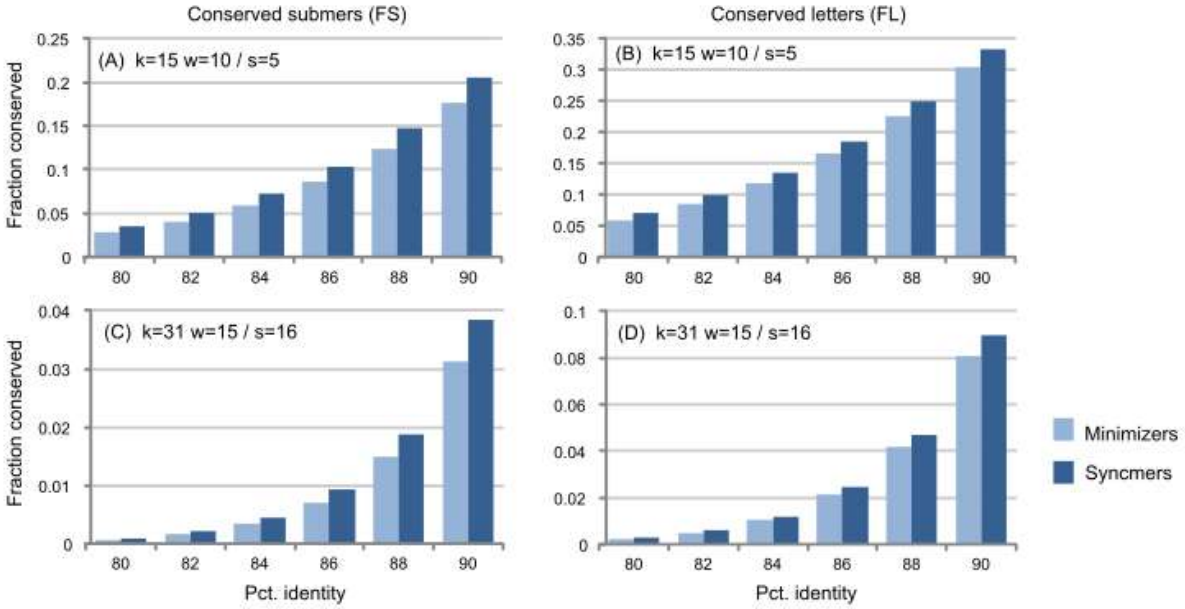

Figure 2. Conservation as a function of identity. The histograms show fraction of conserved submers (FS) and letters (FL) at identities from 80 to 90% for submers comparable to those used in minimap2 (panels A and B) and Kraken (panels C and D).

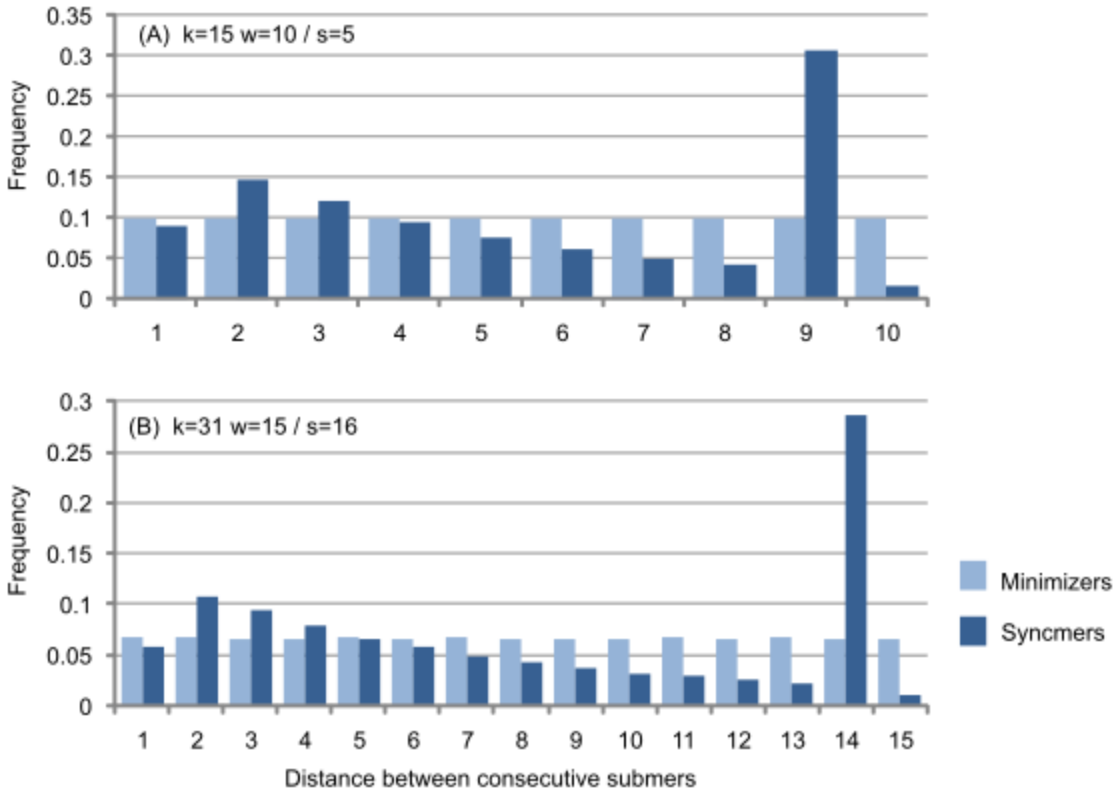

Figure 3. Frequency distribution of distances between consecutive submers. Results are shown on a long random sequence for submers comparable to those used in minimap2 (top, panel A) and Kraken 2 (bottom, panel B). Minimizers have equal frequencies at each distance, implying a strongly non-uniform tiling of the sequence. Syncmers show a strongly peaked distribution, implying more even spacing and hence a more uniform tiling. For a given compression factor, more uniform tiling (equal spacing) induces better conservation as fewer submers are disrupted per mutation.

Id	Assembly	Description
L11	GCA_002222595.2	Blautia hansenii DSM 20583
L13	GCA_004295125.1	[Clostridium] scindens ATCC 35704
L14	GCA_009684695.1	[Clostridium] scindens
L15	GCA_000178835.2	Cellulosilyticum lentocellum DSM 5427
L16	GCA_004210255.1	Blautia producta
L17	GCA_010669205.1	Blautia producta ATCC 27340 = DSM 2950
L18	GCA_008121495.1	[Ruminococcus] gnavus ATCC 29149
L26	GCA_008281175.1	[Clostridium] hylemonae DSM 15053
L27	GCA_900537995.1	Roseburia intestinalis L1-82
L30	GCA_002234575.2	[Clostridium] bolteae
L31	GCA_000225345.1	Roseburia hominis A2-183
L41	GCA_001689125.2	Blautia sp. YL58
L42	GCA_001688665.2	Lachnoclostridium sp. YL32
L43	GCA_900120345.1	Lachnoclostridium phocaeense
L46	GCA_003287895.1	Blautia sp. N6H1-15
L51	GCA_003990395.1	Cellulosilyticum sp. WCF-2
L52	GCA_007361935.1	Lachnospiraceae bacterium KGMB03038
F11	GCA_003019695.1	Fusobacterium gonidiaformans ATCC 25563
F13	GCA_000007325.1	Fusobacterium nucleatum subsp. nucleatum ATCC 25586
F14	GCA_000158275.2	Fusobacterium nucleatum subsp. animalis 7_1
F15	GCA_000162235.2	Fusobacterium nucleatum subsp. vincentii 3_1_36A2
F16	GCA_000163915.2	Fusobacterium nucleatum subsp. vincentii 3_1_27
F17	GCA_000400875.1	Fusobacterium nucleatum subsp. animalis 4_8
F18	GCA_001296085.1	Fusobacterium nucleatum subsp. animalis
F23	GCA_001433955.1	Fusobacterium nucleatum subsp. polymorphum
F24	GCA_001457555.1	Fusobacterium nucleatum subsp. polymorphum
F27	GCA_002211645.1	Fusobacterium nucleatum subsp. animalis
F35	GCA_003019715.1	Fusobacterium necrophorum subsp. funduliforme
F36	GCA_003732505.1	Fusobacterium necrophorum subsp. funduliforme
F40	GCA_003019755.1	Fusobacterium periodonticum

Table 1. Bacterial genome assemblies used in this study.

(A) minimap2

k	*w/s/d*	*c*	*depth*	f_1	*FS80*	*FL80*	*FS90*	*FL90*
15	*w*=10	5.50	2.73	1.0	0.0286	0.0614	0.180	0.308
15	*s*=4	5.94	2.53	1.0	0.0355	0.0639	0.207	0.309
15	***s*=5**	**5.43**	**2.76**	**1.0**	**0.0355**	**0.0673**	**0.205**	**0.320**
15	*s*=6	4.99	3.01	1.0	0.0355	0.0712	0.204	0.334
15	*d*=5	5.01	2.99	0.964	0.0352	0.0671	0.203	0.311

(B) Kraken

k	*w/s/d*	*c*	*depth*	f_1	*FS80*	*FL80*	*FS90*	*FL90*
31	*w*=15	8.00	3.88	1.0	0.00070	0.00235	0.0312	0.0797
31	*s*=15	8.51	3.65	1.0	0.00116	0.00328	0.0381	0.0839
31	***s*=16**	**8.00**	**3.88**	**1.0**	**0.00089**	**0.00257**	**0.0376**	**0.0856**
31	*s*=17	7.49	4.13	1.0	0.00101	0.00297	0.00373	0.0884
31	*d*=8	8.00	3.87	0.984	0.00079	0.00224	0.0374	0.0801

Table 2. Metrics for comparable syncmers to (A) minimap2 and (B) Kraken minimizers. Metrics for minimizers are shown in the top row (gray background). Other rows are for syncmers with comparable compression factors (*c*). The row for the bounded syncmer with closest *c* is in bold type. For comparison, the closest mincode syncmer is in the last row.

Genome1	*Genome2*	*Blast ANI*	*Min. ANI*	*Syn. ANI*	*Blast AF*	*Min. AF*	*Syn. AF*	*Diff. AF(%)*
L14	L26	79.6	84.9	84.7	0.227	0.048	0.051	+6.5
L13	L43	80.7	87.6	87.6	0.121	0.030	0.032	+5.6
F11	F14	82.0	86.0	86.0	0.156	0.037	0.037	+0.5
L11	L46	82.7	88.4	88.2	0.346	0.137	0.142	+4.1
L43	L52	84.4	89.1	88.9	0.147	0.056	0.059	+5.2
L27	L31	84.6	90.4	90.3	0.236	0.105	0.106	+1.3
L17	L41	86.5	88.6	88.5	0.603	0.408	0.414	+1.3
L16	L41	86.5	88.6	88.6	0.607	0.407	0.411	+1.1
F23	F40	87.6	89.1	89.1	0.705	0.484	0.487	+0.7
L11	L18	89.4	98.2	97.9	0.122	0.074	0.075	+2.2
F13	F14	91.0	92.9	92.9	0.748	0.703	0.705	+0.2
L30	L42	92.2	92.6	92.6	0.503	0.475	0.476	+0.3
F18	F27	93.3	97.9	97.9	0.811	0.733	0.733	-0.1
F14	F17	93.8	98.0	98.0	0.858	0.772	0.771	-0.01
F23	F24	94.9	97.2	97.2	0.827	0.764	0.764	+0.1
F15	F16	96.1	98.5	98.5	0.898	0.860	0.859	-0.1
L15	L51	96.7	98.3	98.3	0.826	0.800	0.800	+0.1
F35	F36	98.4	99.1	99.1	0.869	0.832	0.832	0.0
L13	L14	99.4	99.7	99.7	0.786	0.775	0.778	+0.4

Table 3. Results of whole-genome alignment test. ANI is average nucleotide identity, AF is aligned fraction, Min. is minimizers, Syn. is syncmers. Diff AF(%) is ((Syn. AF)*100/(Min. AF) - 100).

Metric	*Minimizers*	*Syncmers*	*Difference*
c	50.8	52.1	2.6%
depth	0.61	0.60	-1.6%
f_1	0.52	0.47	8.9%
FS85	0.0037	0.0061	64%
FS90	0.016	0.039	149%
FS95	0.15	0.21	42%
FL85	0.0023	0.0033	48%
FL90	0.016	0.021	35%
FL95	0.085	0.11	26%

Table 4. Performance metrics for sparse submers. Parameters were chosen to give approximately depth=0.6, i.e. 40% of letters are not covered by a submer. Syncmers are substantially superior to minimizers on all metrics except f_1, the fraction of letters covered by a submer. Note that despite the larger fraction of letters covered by minimizers, conservation is substantially better with syncmers.

k	s	*Size*	c_{Dict}	c_{Random}	c_{Est}
7	3	7.0×10^3	2.3	2.5	2.5
7	4	8.8×10^3	1.9	2.0	2.0
7	5	1.6×10^4	1.4	1.5	1.5
15	3	1.8×10^8	6.0	6.3	6.5
15	4	1.9×10^8	5.5	5.9	6.0
15	5	2.1×10^8	5.1	5.4	5.5
15	6	2.3×10^8	4.6	5.0	5.0

Table 5. Dictionary size for some representative (k, s) parameters. *Size* is the dictionary size (total number of distinct k-mers which are syncmers); c_{Dict} is the compression factor of the dictionary = $4^k/Size$; c_{Random} is the compression factor measured on a random string; and $c_{Est} = (w + 1)/2$ is the value of the compression factor estimated by (Eq.3), where $w = k - s$.