10-Minimizers: A Promising Class of Constant-Space Minimizers

Arseny Shur
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

Ido Tziony
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

Yaron Orenstein
Department of Computer Science and The Mina and Everard Goodman Faculty of Life Sciences, Bar-Ilan University, Ramat Gan, Israel

Abstract

Minimizers are sampling schemes ubiquitous in high-throughput sequencing analysis. Given an alphabet of size σ, a minimizer is defined by two positive integers k and w, and a linear order ρ on k-mers. A sequence is processed by a sliding window algorithm that chooses, in each window of length $w + k - 1$, its minimal k-mer with respect to ρ. A key characteristic of a minimizer is its density, defined as the expected frequency of chosen k-mers among all k-mers in a random infinite σ-ary sequence. Minimizers of lower density are preferred as they produce smaller samples, which lead to reduced runtime and memory usage in downstream applications. Recently developed methods generate minimizers with optimal and near-optimal densities, but these methods require explicit storage of k-mer ranks in $\Omega(2^k)$ space. Methods generating constant-space minimizers with low densities also exist, and some are asymptotically optimal as $k \to \infty$. However, in the non-asymptotic regime, no known class of minimizers had been proven to guarantee, on expectation, a lower density compared to a random minimizer.

In this paper, we introduce a class of 10-minimizers, which has promising properties. First, we prove that for every $k > 1$ and every $w \geq k - 2$, a random 10-minimizer has, on expectation, lower density than a random minimizer, under essentially the same simplifying assumption. This is the first provable guarantee for a class of minimizers in the non-asymptotic regime. Second, we present spacers, which are particular 10-minimizers combining three desirable properties: constant space usage, low density, and short k-mer key-retrieval time. In terms of density, spacers are competitive to the best known constant-space minimizers; in certain (k, w) regimes they achieve the lowest density among all known minimizers. Notably, we are the first to benchmark minimizers by the time spent for k-mer key retrieval, which is the most fundamental operation in many minimizers-based methods. We propose this benchmark as a standard objective for evaluating new minimizer schemes. Our empirical results show that spacers retrieve k-mer keys in competitive time – a few seconds per genome-size sequence – for all practical values of k and w. We expect 10-minimizers to improve minimizers-based methods, especially those using large window sizes.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Applied computing → Molecular sequence analysis

Keywords and phrases Minimizer, constant-space minimizer, local selection scheme, k-mer key retrieval, high-throughput sequencing

Digital Object Identifier 10.4230/LIPIcs.WABI.2026.5

Supplementary Material *Software (Source code and tables)*: `https://github.com/OrensteinLab/10-minimizers`, archived at `swh:1:dir:1a53d6e21b4b8a7d491152c78d360c1a688d240b`

Funding *Yaron Orenstein*: This study was supported by Israel Science Foundation grant no. 358/21.

Acknowledgements Ido Tziony acknowledges the VATAT fellowship for excellent PhD students in Data Science, the Bar-Ilan University presidential fellowship, and the cloud-computing credit support of the Israel Data Science and AI Initiative.

Author contribution A.S. performed the theoretical analysis and led the project, A.S. and I.T. developed, implemented, and benchmarked the algorithms, all authors analyzed the results and wrote the manuscript, and Y.O. supervised I.T. and acquired the funding.

1 Introduction

Sampling short substrings, termed k-mers, in long DNA sequences is a critical step in solving many bioinformatics tasks. Typically, a "window guarantee" is required: each window of w consecutive k-mers (i.e., a window of length $w+k-1$) in the input sequence should be represented by at least one k-mer in the sample. Minimizers, introduced in [20,22], are simple sampling schemes with the window guarantee: a linear order on k-mers is fixed, and in each window, the minimal (lowest-ranked) k-mer w.r.t. this order is selected, with ties broken to the left. Minimizers are the most popular k-mer sampling schemes (a comprehensive list of bioinformatics applications using minimizers can be found in [16]), and even more advanced sampling schemes still use minimizers as an intermediate step [2, 10, 21].

Minimizers are most commonly characterized by their density, which is the expected fraction of sampled k-mers in an infinite sequence of i.i.d. symbols, as lower density leads to reduced runtime and memory usage of downstream applications. The average density (a.k.a. the expected density of a random minimizer) is close to $\frac{2}{w+1}$ unless $w \gg k$ [5, 22]. Many methods were designed to generate low-density minimizers. Methods like DOCKS [18], PASHA [3], and GreedyMini [6], generate orders explicitly, which highly limits the k values that can be used due to both the exponential, in $k+w$, construction time and $\Theta(\sigma^k)$ space occupied by the order itself. GreedyMini pushes this bottleneck by using binary projections of DNA orders, which require $\Theta(2^k)$ space, but this is still restrictive.

Methods like miniception [23], double-decycling [19], and open-closed syncmer [8], define minimizer orders via functions that assign to each k-mer a unique numeric *key* such that the order of k-mers is the order of their keys. For example, for a hash-based random minimizer, the key of a k-mer is its hash value, while double-decycling minimizers use the "class" of the k-mer as the primary key and the hash value as the secondary key, to break ties on primary keys. The minimizers obtained in such a way require only constant space to represent them. The constant-space property enables the application of the minimizers to any value of k.

All the methods above generate, for each pair (k, w), a *class* of minimizers, parameterized in some way: by the auxiliary order for miniception and open-closed syncmer; by the secondary key choice for double-decycling; by the sequence of randomized choices during the course of the algorithm for GreedyMini. In all cases, proven density guarantees (like average-case density over the class) would be desirable, but presently do not exist. For miniception [23] and the mod-sampling scheme [10], *asymptotic* density estimates were proven. However, for practical (k, w) values, these estimates are very loose, and thus of little use[1].

As minimizers are intended for sampling, their sampling time is another very important parameter besides density. Namely, the methods can be ranked by the time required to the key retrieval, which is the method-specific part of sampling. For all methods with explicitly stored orders, every key is retrieved by one memory access. However, the key retrieval in constant-space methods varies a lot, and a method with low empirical density still can slow down an application due to a complex key computation. Previously, benchmarking

[1] For example, for $k = 15$ and $w = 10$ (the default parameters used in Minimap [13] and Minimap2 [14]), the measured density of the mod-sampling scheme with the default parameter $t = 4$ is ≈ 0.143 [6, Fig. 3B], while [10, Thm. 10] gives the estimate ≈ 0.177 plus an o-term, which means the error of 25%.

did not isolate k-mer key-retrieval time. Instead, evaluations focused on the total time required to find the minimizer - a metric heavily dependent on the specific implementation of the minimization algorithm (which, for instance, accounts for 80% of the runtime in SimdMinimizers [9]).

In this paper, we present 10-minimizers, a novel class of minimizers with the first provable guarantee of achieving density better than random in the non-asymptotic regime. We prove that random 10-minimizers have the expected density $\approx \frac{2}{w+2}$ compared to $\approx \frac{2}{w+1}$ achieved by random minimizers. We present a particular family of constant-space 10-minimizers, called *spacers*, that achieve densities competitive to those of the known advanced constant-space minimizers. Moreover, for some (k, w) pairs, spacers achieve the lowest density compared to all known (not necessarily constant-space) minimizers. Last, we formally define a performance-evaluation criterion for minimizers: k-mer key-retrieval time, and demonstrate that spacers, as well as random 10-minimizers, are highly competitive. In particular, spacers are faster than all constant-space methods that reach, for some (k, w) ranges, lower density. We expect 10-minimizers to improve the runtime and memory usage of many high-throughput sequencing methods by a simple replacement of their current k-mer key calculations.

2 Preliminaries

In what follows, Σ, σ, k, and w denote, respectively, the alphabet $\{0, \ldots, \sigma-1\}$, its size, the length of the sampled substrings (k-mers), and the number of k-mers in a window. We write $s[1..n]$ for a length-n string, denote the length of s by $|s|$, and use standard definitions of substring, prefix, and suffix of a string. The empty string is denoted by ε. We write n-string (n-prefix, n-suffix, n-window) to indicate length. We use the notation $[i..j]$ for the range of integers from i to j, and $s[i..j]$ for the substring of s covering the range $[i..j]$ of positions.

Let ρ be a permutation of Σ^k, which we view interchangeably as a linear order and as a string of distinct k-mers. Formally, we define ρ as a bijection $\rho : [1..\sigma^k] \to \Sigma^k$, binding all k-mers to their *ρ-ranks*. The *minimizer* (ρ, w) is a map $f : \Sigma^{w+k-1} \to [1..w]$ assigning to each $(w+k-1)$-window the starting position of its minimum-ρ-rank k-mer, with ties broken to the left. This map acts on strings (which we interchangeably call sequences) over Σ, selecting one position in each window so that in a window $v = S[i..i+w+k-2]$ the position $i + f(v) - 1$ in S is selected.

Minimizers are evaluated by the density of selected positions. Let $f(S)$ denote the set of positions selected in a string S by a minimizer f. The *density of f* is the limit $d_f = \lim_{n\to\infty} \frac{1}{\sigma^n} \sum_{S\in\Sigma^n} \frac{|f(S)|}{n}$ (also termed the *expected* density, to distinguish from the *particular* density over a specific sequence). Studying minimizer densities over a range of w values, it is convenient to "normalize" density to the *density factor* $D_f = (w+1)d_f$.

An *arrangement* π of Σ^k with the *domain* $\varnothing \neq U \subseteq \Sigma^k$ is a permutation of U. In particular, arrangements with the domain Σ^k are exactly (linear) orders. We view π as a bijection $\pi : [1..|U|] \to U$, and apply the notion of π-rank to arbitrary arrangement π. We also consider π as an ordered list of length $|\pi| = |U|$. We write U_π for the domain of π, and denote by $\pi_1 \cdot \pi_2$ the standard string concatenation of arrangements π_1 and π_2 with disjoint domains, i.e., $U_{\pi_1} \cap U_{\pi_2} = \varnothing$.

A subset $H \subseteq \Sigma^k$ is a *universal hitting set* (UHS) *for w* if every $(w+k-1)$-window contains at least one k-mer from H. For example, $\{00, 01, 11\}$ is a UHS for $\sigma = k = 2$ and any $w > 1$. We call an arrangement π a *UHS order for w* if U_π is a UHS for w. Then, any two orders $\pi \cdot \pi_1$ and $\pi \cdot \pi_2$ define the same minimizer for w, because the minimum-rank k-mer in every window is in π; we denote this minimizer by (π, w).

For a given minimizer $f = (\rho, w)$, a $(w+k)$-window v (which contains two consecutive $(w+k-1)$-windows and often termed *context*) is *charged* if its minimum-rank k-mer is either its prefix or its *unique suffix* (i.e., the k-suffix of v having no other occurrence in v); otherwise, v is *free*. Every string S has exactly $|f(S)| - 1$ charged $(w+k)$-windows [24, Lemma 6]. Since all possible n-strings have, in total, the same number of occurrences of each $(w+k)$-window, the density d_f of a minimizer equals the fraction of charged windows in Σ^{w+k} [15, 23].

For an arrangement π of Σ^k, a $(w+k)$-window v is *charged by* π *(due to* $\pi[i]$*)* if $\pi[i]$ is the k-mer of minimum π-rank in v, and is a prefix (*prefix-charged*) or a unique suffix (*suffix-charged*) of v. The number of windows over Σ^{w+k} charged by π is denoted by $\mathsf{ch}_w(\pi)$. A *live* window (w.r.t. a set $U \subset \Sigma^{w+k}$) is a window with no k-mer in U. If $U \subset \Sigma^k$ is disjoint from U_π, then $\mathsf{ch}_w(U, \pi)$ denotes the number of windows that are both live w.r.t. U and charged by π. A (UHS) order ρ is *initiated by* U if some prefix of ρ has the domain U.

Let (ρ, w) be a binary minimizer, $\sigma > 2$, and let $h : \Sigma \to \{0, 1\}$ be a surjective *projection* map, extended to all strings over Σ letter-wise. A minimizer (ρ', w) over Σ is a *σ-extension of (ρ, w) with the projection h* if for any k-mers $u, v \in \Sigma^k$ the condition $\mathsf{rank}_\rho(h(u)) < \mathsf{rank}_\rho(h(v))$ implies $\mathsf{rank}_{\rho'}(u) < \mathsf{rank}_{\rho'}(v)$. In other words, two k-mers u and v can be compared w.r.t. ρ' by comparing their projections $h(u)$ and $h(v)$ w.r.t. ρ, using ρ' only to break ties.

In a similar way, *k-extensions* of minimizers are defined. Let (ρ, w) and (ρ', w) be two minimizers over some alphabet Σ such that ρ is an order on Σ^k, ρ' is an order on $\Sigma^{k'}$, and $k' > k$. If for any k'-mers $u, v \in \Sigma^{k'}$ the condition $\mathsf{rank}_\rho(u[1..k]) < \mathsf{rank}_\rho(v[1..k])$ implies $\mathsf{rank}_{\rho'}(u) < \mathsf{rank}_{\rho'}(v)$, then (ρ', w) is a *k-extension* of (ρ, w).

Every string $s \in \Sigma^*$ is a base-σ representation of an integer, denoted by $\mathsf{num}(s)$. If $|s| = |u|$, then $\mathsf{num}(s) < \mathsf{num}(u)$ if and only if s precedes u lexicographically. We use Iverson notation: $[\mathsf{expression}]$ evaluates to 1/0 depending on $\mathsf{expression}$ being true/false.

3 Methods

3.1 10-minimizers

We start with the case $\Sigma = \{0, 1\}$. Let $\mathsf{IO}_k = \{10u \mid u \in \{0, 1\}^{k-2}\}$; the elements of IO_k are *10-k-mers* ("one-zero k-mers", a property that will make low-ranked k-mers well spaced).

We consider the binary UHS orders initiated by IO_k. First we solve the problem common to all such orders: to find an arrangement τ that hits all windows that are live w.r.t. IO_k and charges only few of them.

▶ **Lemma 1.** *Suppose that $\sigma = 2$, $w \geq k - 2$, $\tau = (1^{k-1}0, 01^{k-2}0, \ldots, 0^{k-2}10, 0^{k-1}1, 1^k, 0^k)$, and π is an arrangement of IO_k. Then $\pi \cdot \tau$ is a UHS order for w and $\mathsf{ch}_w(\mathsf{IO}_k, \tau) = w + k + 3 + [w = k - 2]$.*

Proof. Since a window is live w.r.t. IO_k if and only if it contains no k-mers beginning with 10, the set X of such live windows is given by $X = \{0^i 1^{w+2-i} u \mid i \in [0..w+2], u \in \{0, 1\}^{k-2}\}$. For $1 \leq i \leq k-1$, any window with the prefix $\tau[i] = 0^{i-1}1^{k-i}0$ does not belong to X: the longest prefix of the form $0^i 1^j$ in a window beginning with $\tau[i]$ has length $k-1$, while the windows in X have such a prefix of length at least $w + 2 > k - 1$. Hence, $\pi \cdot \tau$ prefix-charges 0 windows due to $\tau[1], \ldots, \tau[k-1]$. As a unique suffix, $1^{k-1}0$ occurs in all live windows of the form $0^i 1^{w-i} 1^{k-1} 0$, $i \in [0..w]$, so it suffix-charges $w + 1$ windows. Each of the k-mers $0^i 1^{k-i-1} 0$, $i \in [1..k-2]$, is a unique suffix of exactly one window from X, namely, $0^{w+i-1}1^{k-i}0$. Since a window from X cannot contain both $\tau[i]$ and $\tau[j]$, where $1 \leq i < j \leq k - 1$, each $\tau[i]$, $i \in [2..k - 1]$, suffix-charges one window.

Suppose that an n-window v is live w.r.t. $\pi \cdot \tau[1..k{-}1]$ and contains 10. We can write $v = v_1 10 v_2$ for some $v_1, v_2 \in \Sigma^*$ such that v_1 does not contain 10. Then, $|v_1| \le k-3$ (otherwise, v contains some $\tau[i]$) and $|v_2| \le k-3$ (otherwise, v contains a k-mer from π). Therefore, $n \le 2k-4 \le w+k-2$. Hence, the live $(w+k)$-windows w.r.t. $\pi \cdot \tau[1..k{-}1]$ form the set $X' = \{0^i 1^{w+k-i} \mid i \in [0..w{+}k]\}$ (and for $(w+k-1)$-windows, the structure of the set is the same). Every window from X' contains at least one of the k-mers $\tau[k] = 0^{k-1}1$, $\tau[k{+}1] = 1^k$, $\tau[k{+}2] = 0^k$, implying that $\pi \cdot \tau$ is a UHS for w. It remains to note that $\tau[k] = 0^{k-1}1$ charges the windows $0^{k-1}1^{w+1}$ and $0^{w+k-1}1$, $\tau[k{+}1] = 1^k$ charges 1^{w+k} and $0^w 1^k$ (the latter one only if it does not contain $0^{k-1}1$, i.e., for $w = k-2$), and $\tau[k{+}2] = 0^k$ charges 0^{w+k}. Summing up all windows charged due to τ, we get the amount stated in the lemma. ◀

▶ Remark 2. The only k-mers from τ that can occur more than once in a live window w.r.t. IO_k are 1^k and 0^k.

▶ **Definition 3** (10-orders and 10-minimizers). *An order on $\{0,1\}^k$ is a* 10-order *("one-zero-order") if it has the prefix $\pi \cdot \tau$, where π is an arbitrary arrangement of IO_k and τ is the arrangement defined in Lemma 1. An order ρ on $\{0,\dots,\sigma-1\}^k$ is a* 10-order *if it is a σ-extension of a binary 10-order with a projection $h : \{0,\dots,\sigma-1\} \to \{0,1\}$. If, moreover, $|h^{-1}(0)| = |h^{-1}(1)|$, then ρ is a* balanced *10-order. A minimizer defined by a 10-order is called a* 10-minimizer *("one-zero-minimizer").*

3.2 Expected density of random 10-minimizers

▶ **Definition 4** (random minimizers). *A* random minimizer *is a random variable equidistributed over all minimizers with given parameters σ, k, and w. A* random 10-minimizer *is a random variable equidistributed over all 10-minimizers with given parameters σ, k, and w, and with a given projection h in the case $\sigma > 2$.*

The expected density of a random minimizer equals $\sigma^{-w-k} \sum_{v \in \Sigma^{w+k}} P(v)$, where $P(v)$ is the probability of the window v being charged [5]. To estimate this expected density, a simplifying assumption was used [15], yielding an error that is small for wide ranges of parameters. Let us call a window *repetitive* if it contains some k-mer at least twice, and *repetition-free* if it is not repetitive. Informally, the assumption says "ignore all repetitive $(w{+}k)$-windows". As such windows are rare unless $w \gg k$, the error introduced by this assumption can be proved small. Since any repetition-free $(w{+}k)$-window v contains $(w{+}1)$ distinct k-mers, a random minimizer prefix-charges v with probability $\frac{1}{w+1}$ and suffix-charges v with the same probability. Hence, the simplifying assumption can be formalized directly as follows:

(A) the random minimizer with the parameters (σ, k, w) prefix-charges any $(w{+}k)$-window with probability $\frac{1}{w+1}$ and suffix-charges it with the same probability $\frac{1}{w+1}$.

Assuming (A), one estimates the expectation of the density of a random (σ, k, w)-minimizer to $\frac{2}{w+1}$. Note that for a $(w{+}k)$-window with t distinct k-mers, the probability of being prefix-charged by a random minimizer is $\frac{1}{t}$, while the probability of being suffix-charged is either $\frac{1}{t}$ or 0, depending on the uniqueness of the k-suffix. Thus, for repetitive $(w{+}k)$-windows (A) underestimates the probability of being prefix-charged, while the probability of being suffix-charged can be affected in any direction.

In what follows, we formulate similar assumptions for random 10-minimizers and estimate their expected density based on these assumptions. We restrict ourselves to the case $w \ge k-2$. Therefore, $\pi \cdot \tau$ is a UHS order for any arrangement π of IO_k.

3.2.1 Case $\sigma = 2$

We introduce the simplifying assumption for random binary 10-minimizers, aiming to ignore only a fraction of all repetitive windows. Namely, a $(w{+}k)$-window v is *essentially repetitive* if there exists a k-mer u that occurs in v at least twice and is minimal in v in at least one 10-order but not in all 10-orders. Informally, the assumption says "ignore all essentially repetitive $(w{+}k)$-windows".

The status (charged or free) of any window without 10-k-mers is the same for any 10-minimizer, and the number of charged windows of this type is computed in Lemma 1. Hence, essentially repetitive windows are those containing some 10-k-mer at least twice. If a $(w{+}k)$-window v contains m 10-k-mers, each occurring in v exactly once, then v is prefix-charged (respectively, suffix-charged) with probability $\frac{1}{m}$ if its k-prefix (respectively, k-suffix) is a 10-k-mer. Accordingly, we assign the same probabilities to all $(w{+}k)$-windows with m occurrences of 10-k-mers, formalizing our assumption as follows:

(A†) the random 10-minimizer with the parameters $(2, k, w)$, where $w \geq k{-}2$, prefix-charges (respectively, suffix-charges) any $(w{+}k)$-window v with $m \geq 1$ occurrences of 10-k-mers with probability $\frac{1}{m}$ if v has a 10-k-mer as a prefix (respectively, suffix).

Using (A†), we get the following estimate, very close to $\frac{2}{w+2}$, for the expected density $\mathcal{R}(2, k, w)$ of a random binary 10-minimizer.

▶ **Theorem 5.** *Under the assumption* (A†)*, the expected density $\mathcal{R}(2, k, w)$ of a random 10-minimizer with the parameters $(2, k, w)$, where $w \geq k{-}2$, approximates to*

$$\mathcal{R}^*(2, k, w) = \frac{2}{w+2} - \frac{1}{2^w(w+2)} + \frac{k+w+3+[w=k-2]}{2^{k+w}}. \tag{1}$$

Proof. Let k and w be fixed. For $m \geq 1$, we denote by $\mathsf{IOP}(m)$ (resp., $\mathsf{IOS}(m)$) the set of $(w{+}k)$-windows with m occurrences of 10-k-mers, one of which is a prefix (resp., suffix) of the window. The function that deletes the first two letters of the k-suffix of a window and appends them to the beginning of the window is an obvious bijection from $\mathsf{IOS}(m)$ to $\mathsf{IOP}(m)$ for every $m \geq 1$. Therefore, $|\mathsf{IOP}(m)| = |\mathsf{IOS}(m)|$. Assuming (A†) and taking Lemma 1 into account, we obtain

$$\mathcal{R}^*(2, k, w) = 2^{-(k+w)}\Big(2 \cdot \sum\nolimits_m \frac{|\mathsf{IOP}(m)|}{m} + k + w + 3 + [w = k-2]\Big). \tag{2}$$

To compute $|\mathsf{IOP}(m)|$, we note that a $(w+k)$-window v belongs to $\mathsf{IOP}(m)$ if and only if v begins with 10 and contains m occurrences of 10 in the prefix $v[1..w{+}2]$. Equivalently,

$$v = 10^{i_1}1^{j_1}0^{i_2}1^{j_2}\cdots 0^{i_m}1^{j_m}x, \text{ where } i_1, \ldots, i_m, j_1, \ldots, j_{m-1} > 0,\ j_m \geq 0,\ |x| = k-2.$$

Respectively, the number of options for v is 2^{k-2} times the number of positive integer solutions to the inequality $i_1 + j_1 + \cdots + j_{m-1} + i_m \leq w+1$. This number of solutions is the number of possibilities to choose distinct values for the $(2m-1)$ partial sums

$$i_1,\ i_1 + j_1,\ i_1 + j_1 + i_2,\ \ldots,\ i_1 + j_1 + \cdots + j_{m-1} + i_m,$$

which is $\binom{w+1}{2m-1}$. Hence, $|\mathsf{IOP}(m)| = 2^{k-2}\binom{w+1}{2m-1}$. To obtain (1), we prove a technical claim.

▷ Claim 6. $\sum_{t=1,\, t \text{ odd}}^{n} \frac{2}{t+1}\binom{n}{t} = \frac{2^{n+1}-2}{n+1}$.

Proof. We first observe that (i) $\frac{1}{t+1} = \int_0^1 x^t dx$. Next, $\sum_{t=1}^{n} \binom{n}{t} x^t = (1+x)^n$, $\sum_{t=1}^{n} \binom{n}{t}(-x)^t = (1-x)^n$, and thus we obtain (ii) $2 \cdot \sum_{t=1,\, t \text{ odd}}^{n} \binom{n}{t} x^t = (1+x)^n - (1-x)^n$. As a result, we have

$$\sum_{\substack{t=1 \\ t \text{ odd}}}^{n} \frac{2}{t+1}\binom{n}{t} =_{\text{(i)}} \sum_{\substack{t=1 \\ t \text{ odd}}}^{n} 2\binom{n}{t}\int_0^1 x^t dx = \int_0^1 2 \cdot \sum_{\substack{t=1 \\ t \text{ odd}}}^{n} \binom{n}{t} x^t dx$$
$$=_{\text{(ii)}} \int_0^1 ((1+x)^n - (1-x)^n)dx = \frac{2^{n+1}-1}{n+1} - \frac{1}{n+1} = \frac{2^{n+1}-2}{n+1}.$$

The claim is proved. ◁

To finish the proof, we substitute the formula for $|\mathsf{IOP}(m)|$ into (2), using Claim 6 with $n = w+1$, $t = 2m-1$ to compute the sum (note that $\frac{2}{t+1} = \frac{1}{m}$) and finally obtain (1). ◀

3.2.2 Case $\sigma \geq 3$

Let a 10-order ρ on Σ^k be the σ-extension of a binary 10-order ρ^* by a projection h, and let $v \in \Sigma^{k+w}$. If the binary window $h(v)$ has a unique minimum k-mer in the order ρ^*, then v is charged by ρ if and only if $h(v)$ is charged by ρ^*. Hence, the random 10-order with the parameters (σ, k, w, h) charges v with the same probability as the random 10-order with the parameters $(2, k, w)$ charges $h(v)$, except for two special cases of repetitive windows. If $h(v)$ contains 10-k-mers, the special case arises if $h(v)$ contains a repeated 10-k-mer; this case is exactly the one covered by the assumption ($\mathrm{A}^\dagger$). If $h(v)$ contains no 10-k-mers, its minimum-rank k-mer is from τ (Lemma 1), and hence is the same for all binary 10-orders. By Remark 2, 0^k and 1^k are the only k-mers in τ that can repeat in $h(v)$, and one of these k-mers has the minimum rank in $h(v)$ only if $h(v) = 0^{w+k}$ or $h(v) = 0^i 1^{w+k-i}$, where $i \in [0..k-2]$. This is the second special case where v and $h(v)$ have different probabilities of being charged by the random 10-orders, but it affects a negligibly small set of windows (k distinct projections out of 2^{w+k} possible).

Thus, to be consistent with ($\mathrm{A}^\dagger$), we state our assumption for the case $\sigma \geq 3$ as follows:

($\mathrm{A}^\ddagger$) the random 10-minimizer with the parameters (σ, k, w, h), where $\sigma \geq 3$ and $w \geq k-2$, charges any $(w+k)$-window v with the same probability as the random 10-minimizer with the parameters $(2, k, w)$ charges the window $h(v)$ under the assumption ($\mathrm{A}^\dagger$).

Using ($\mathrm{A}^\ddagger$), we extend Theorem 5 to *balanced* random 10-minimizers.

▶ **Theorem 7.** *Under the assumption* ($\mathrm{A}^\ddagger$)*, the expected density* $\mathcal{R}(\sigma, k, w, h)$ *of a balanced random 10-minimizer with the parameters* (σ, k, w, h)*, where* $\sigma \geq 3$ *and* $w \geq k-2$*, approximates to* $\mathcal{R}^*(\sigma, k, w, h) = \mathcal{R}^*(2, k, w)$*.*

Proof. For $v \in \Sigma^{w+k}$, let $P(v)$ be the probability of v being charged by the random 10-minimizer with the parameters (σ, k, w, h), assigned according to ($\mathrm{A}^\ddagger$). Similarly, $P'(z)$ denotes the probability that $z \in \{0,1\}^{w+k}$ is charged by the random 10-minimizer with the parameters $(2, k, w)$, assigned according to ($\mathrm{A}^\dagger$). Then, $\mathcal{R}^*(\sigma, k, w, h) = \sigma^{-k-w} \sum_{v \in \Sigma^{w+k}} P(v)$ and $\mathcal{R}^*(2, k, w) = 2^{-k-w} \sum_{z \in \{0,1\}^{w+k}} P'(z)$. Since $|h^{-1}(0)| = |h^{-1}(1)| = \frac{\sigma}{2}$, every window $z \in \{0,1\}^{w+k}$ satisfies $|h^{-1}(z)| = \left(\frac{\sigma}{2}\right)^{w+k}$. By ($\mathrm{A}^\ddagger$), all windows with the projection z have the same probability of being charged, equal to $P'(z)$. Then, $\sum_{v \in \Sigma^{w+k}} P(v) = \left(\frac{\sigma}{2}\right)^{w+k} \sum_{z \in \{0,1\}^{w+k}} P'(z)$, implying $\mathcal{R}^*(\sigma, k, w, h) = \mathcal{R}^*(2, k, w)$. ◀

▶ Remark 8. The error of the approximations computed in Theorems 5, 7 can be roughly bounded by the fraction of $(w+k)$-windows with repeated k-mers among all $(w+k)$-windows. To evaluate the actual error, we computed the exact expected density in the case where $\sigma = 2$ and $k \in \{7, 12\}$; see Section 4.1.

3.3 Low-density 10-minimizers

3.3.1 Existing 10-minimizers

In [1], the *ABB order* was proposed in the context of studying codes. This order compares any two k-mers over Σ using the alphabetic order for their first letters and the rule $0 > 1 = 2 = \cdots = \sigma - 1$ for subsequent letters. Neither [1] nor the later paper [4] (which applied this order as part of a minimizer) specified the tie-break used to make ABB a linear order. In [7, Ch. 8], lexicographic tie-breaking was assumed and the resulting order, termed ABB+, was considered from the density point of view.

For every $k > 1$, the ABB+ order is an "almost" 10-order: it is obtained by σ-extension of a binary order ρ with the projection $h(0) = 1, h(a) = 0$ for $a = 1, \ldots, \sigma-1$, and ρ is initiated by the set IO_k. The sorting used for the rest of k-mers is inefficient compared to the arrangement τ employed in 10-orders (Lemma 1), but this makes a difference only for small w (for large w, the windows with no 10-k-mers in their projections are extremely rare). Since ABB+ in most cases has smaller density than the expected density of a random 10-minimizer, it serves as an important baseline for our minimizers.

3.3.2 Spacers

For a k-mer $u \in \{0,1\}^k$, its *tail* $\mathsf{tail}(u)$ is the longest proper suffix of u that is a prefix of a 10-k-mer. For example, $\mathsf{tail}(1001010) = 1010$, $\mathsf{tail}(1001111) = 1$, $\mathsf{tail}(1000000) = \varepsilon$. To obtain binary 10-minimizers of lower density than expected by Theorem 5, we assign lower ranks to the 10-k-mers that ensure larger distance to the next 10-k-mer in the input sequence. In other words, we prioritize 10-k-mers with short tails.

▶ **Definition 9** (binary spacer)**.** Binary spacer (ρ, w) *is a binary 10-minimizer such that ρ ranks all 10-k-mers by assigning to each 10-k-mer u the triple* $(|\mathsf{tail}(u)|, -\mathsf{num}(\mathsf{tail}(u)), \mathsf{num}(u))$ *of numeric keys, and comparing these triples lexicographically.*

▶ Remark 10.

(1) The primary key $|\mathsf{tail}(u)|$ distinguishes binary spacers from binary ABB+ orders. For example, a binary spacer assigns the k-mer 101110 lower rank compared to 100100, while binary ABB+ (assuming $1 < 0$) ranks these k-mers in reverse order.

(2) Using the inverse lexicographic order on tails as the secondary key showed the best empirical density results among the "simplest" tie-breakers.

(3) The choice of tertiary key has a near-zero impact on the density (e.g., for $k = 12$ and $10 \le w \le 36$ choosing the inverse lexicographic order for the tertiary key instead of the lexicographic order changes the density by $< 0.04\%$ in all cases); the only purpose of this key is to create a linear order.

▶ **Proposition 11.**

(1) *Any binary spacer (ρ, w) is a constant-space minimizer.*

(2) *Assuming that a k-mer fits into $O(1)$ machine words, any two k-mers can be compared w.r.t. the order ρ in $O(1)$ time.*

Proof.

(1) Definition 9 provides the required $O(1)$-size description for all binary spacers.

(2) It suffices to show that $|\mathsf{tail}(u)|$ can be found from the 10-k-mer u in $O(1)$ operations. We compute the "indicator" $\mathsf{ind}(u) = \mathsf{num}(u)\,\&\,{\sim}(\mathsf{num}(u) \ll 1)$ (here $\&$, $\sim$, and $\ll$ are the bitwise and, bitwise not, and left shift, respectively). The positions of 1's in $\mathsf{ind}(u)$ are exactly the positions of 10's in u. Then the second, from the left, occurrence of 1 in $\mathsf{ind}(u)$ is at the position of $\mathsf{tail}(u)$ in u by the definition of tail. Two more operations are needed to locate this position: a xor operation with the constant 2^{k-1} removes the leftmost 1, and the number of leading zeroes in the fixed-length representation of the obtained number is computed by the instruction lzcnt, which is standard for modern CPUs. ◀

Binary spacers have significantly lower density than binary ABB+. For example, for $w = 24$ and $k \in [8..26]$, binary spacers have the density factors ≈ 1.74, compared to ≈ 1.86 for ABB+ and ≈ 1.92 for the random 10-minimizer according to Theorem 5 (Supplementary Table A2).

3.3.3 DNA spacers

As DNA minimizers are our primary interest, we consider σ-extensions of binary spacers. Such a σ-extension is defined by adding a tie-breaker for k-mers with the same projection, to create a linear order. This tie-breaker makes the tertiary key of the binary spacer redundant, as the only purpose of the latter is to create a linear order; see Remark 10(3). Taking this observation into account, we define spacers over larger alphabets as follows.

▶ **Definition 12** (spacer). *Let an alphabet Σ of size $\sigma > 2$ and a projection map $h : \Sigma \to \{0, 1\}$ be given. A k-mer $u \in \Sigma^k$ is* 10-projected *if $h(u)$ is a 10-k-mer.* Spacer (ρ, w) *over Σ with the projection h is a 10-minimizer such that ρ ranks 10-projected k-mers by assigning to each such k-mer u the triple $(|\mathsf{tail}(h(u))|, -\mathsf{num}(\mathsf{tail}(h(u))), \mathsf{num}(u))$ of numeric keys, and comparing these triples lexicographically.*

If a spacer is balanced (i.e., if h is balanced), then it inherits the density of the corresponding binary spacer almost exactly (for the details on the density guarantees for balanced σ-extensions see [6, Sect. 3.3]). On the other hand, σ-ary ABB+ minimizers, being *unbalanced* σ-extensions of binary minimizers, show much lower densities than in the binary case (density factor ≈ 1.72–1.73 for the above-mentioned range $w = 24$, $k \in [8..26]$, and the DNA alphabet). To reach even lower density, we focus on unbalanced spacers. For the 4-ary alphabet, we define the *DNA spacer* by the projection $h(0) = h(1) = h(2) = 0$, $h(3) = 1$.

▶ Remark 13. Given a sequence x of n bits, the subsequence of bits in even positions can be extracted from x in $O(\log n)$ bitwise operations by a standard FFT-like algorithm. As a DNA k-mer u is a sequence of $2k$ bits, its projection $h(u)$ can be found in $O(\log k)$ operations by extracting the bits in even positions from the sequence $(u\,\&\,(u \ll 1)) \gg 1$.

Proposition 11, Remark 13, and Definition 12 together imply

▶ **Proposition 14.**

(1) *Any DNA spacer (ρ, w) is a constant-space minimizer.*

(2) *Assuming that a k-mer fits into $O(1)$ machine words, any two k-mers can be compared w.r.t. the order ρ in $O(\log k)$ time.*

As we demonstrate in Section 4.2, DNA spacers reach lower density than ABB+ DNA minimizers for all (w, k) regimes.

3.4 k-mer key retrieval

3.4.1 Key retrieval problem

Fast sequence sampling with a minimizer (ρ, w) requires efficient solutions to two problems. First, the input sequence S should be converted into a sequence of fast-sortable keys, preferably numeric, representing all k-mers in S. Second, the minimum key in every set of w consecutive keys should be computed; this key represents the minimum k-mer w.r.t. ρ in the corresponding $(w{+}k{-}1)$-window. The second problem is independent of ρ, so we do not consider it here. Meanwhile, the solution to the first problem is ρ-specific unless ρ is explicitly stored. To be practically useful, a constant-space minimizer should be endowed with a fast key-retrieval algorithm. Let us formally define the problem.

▶ **Definition 15** (k-mer key-retrieval problem for DNA minimizers).
Parameters: *$\sigma = 4$, $k \geq 2$, $w \geq 2$, an order ρ on Σ^k.*
Input: *an arbitrary sequence $S = S[1..n] \in \Sigma^*$.*
Output: *a sequence $K = K[1..n{-}k{+}1]$ of k-mer keys, satisfying the following property:*
($*$) *for every $(w{+}k{-}1)$-window $v = S[i..i{+}w{+}k{-}2]$ in S and every $j \in [i..i{+}w{-}1]$, the k-mer $S[j..j{+}k{-}1]$ has the minimum ρ-rank among the k-mers in v if and only if $K[j]$ is the minimum key among $K[i], \ldots, K[i{+}w{-}1]$.*

▶ Remark 16. The condition ($*$) ensures that K suffices for a correct sampling of S with the minimizer (ρ, w). At the same time, ($*$) gives a lot of freedom in assigning keys to k-mers. In particular, no bijection between k-mers and keys is required. For example, if $S[j..j{+}k{-}1]$ provably cannot be a minimum k-mer in any window it occurs in, then $K[j]$ can be safely set to be the "infinite" key.

We suggest the key-retrieval time as an important metric for local sampling schemes, in particular, minimizers. For minimizers, we assume standard conditions for all methods:
(1) Input sequence S is processed left to right, one symbol at a time
(2) Each iteration consists of
- reading the next symbol and updating the k-suffix of the input read so far
- computing and reporting the key of the k-suffix

Under such conditions, the difference of key-retrieval time between methods reflects the complexity of computing the key, which is exactly the method-specific parameter we suggest to measure. In addition, we get a natural lower bound for all methods: the time needed by the minimizer with **no** key computation, which is exactly the lexicographic minimizer (it reports the k-suffix as the key).

The key assignment and key retrieval for DNA spacers are described below, and the results of benchmarking of all competitive methods are given in Section 4.3.

3.4.2 Assigning and retrieving k-mer keys for DNA spacers

▶ **Definition 17** (k-mer keys assignment for DNA spacers). *The key of a k-mer u is a pair $(\mathsf{pkey}(u), \mathsf{num}(u))$ of integers, where the* primary key $\mathsf{pkey}(u)$ *is assigned as follows.*
(1) *If u is 10-projected, then $\mathsf{pkey}(u)$ represents the pair $(|x|, -\mathsf{num}(x))$, where $x = \mathsf{tail}(h(u))$; we assign $\mathsf{pkey}(u) = |x| \cdot 2^{k-2} + 2^{k-2} - \mathsf{num}(x)$;*
(2) *If u is not 10-projected, then $\mathsf{pkey}(u)$ is one of five predefined constants $X_1 < X_2 < X_3 < X_4 < X_\infty$, where $X_1 > \mathsf{pkey}(u)$ for any 10-projected k-mer u; specifically,*
- *X_1 is assigned if $h(u) \in \tau[1..k{-}1]$;*
- *X_2 is assigned if $h(u) = \tau[k] = 0^{k-1}1$;*
- *X_3 is assigned if $h(u) = \tau[k{+}1] = 1^k$;*

- X_4 *is assigned if* $h(u) = \tau[k{+}2] = 0^k$*;*
- X_∞ *is assigned if* $h(u) \notin \tau$*;*

($\star$) *in addition,* X_∞ *is assigned instead of* X_1–X_4 *if at the moment of assignment* u *is known to be* ***non-minimum*** *in every window it appears in.*

▶ Remark 18.

(i) The k-mers with $\mathsf{pkey} = X_1$ have projections ending in 10. Hence, any two k-mers with $\mathsf{pkey} = X_1$ are separated by at least one 10-projected k-mer (which has a smaller pkey by the choice of X_1). Then, two k-mers with $\mathsf{pkey} = X_1$ are never compared for the minimum in a window. This property justifies the use of the same pkey for the k-mers with the projections from $\tau[1..k{-}1]$.

(ii) If $w \geq k-2$, Lemma 1 ensures that every $(w{+}k{-}1)$-window contains a k-mer with the projection in IO_k or in τ. Hence, tie breaks for the k-mers with $\mathsf{pkey} = X_\infty$ never affect the computation of the minimum k-mer in a window.

▶ **Proposition 19.** *Any algorithm assigning the k-mer keys according to Definition 17 solves the key-retrieval problem for DNA spacers.*

Proof. From Definition 12 and Remark 18, it follows that, for any input sequence S, the sequence K of keys assigned according to Definition 17 satisfies Definition 15($*$) for (ρ, w) being the DNA spacer. Since ($*$) ensures the correct sampling of S, the proposition holds. ◀

Now we describe KRspacer, a fast key-retrieval algorithm for DNA spacers. It obeys conditions (1,2) (Section 3.4.1) so that each iteration follows the scheme

$$\text{read a symbol } a \to \text{update the } k\text{-suffix } u \text{ and its projection } h(u) \text{ using } a \to$$
$$\to \text{compute } \mathsf{pkey} \text{ from } h(u) \to \text{report } \mathsf{pkey}, \mathsf{num}(u)\,.$$

A *trivial* iteration skips the pkey computation and reports $\mathsf{pkey} = X_\infty$ immediately after the update step. All other iterations are *nontrivial*. KRspacer uses Definition 17($\star$) to make most of the iterations trivial.

(1) KRspacer has three *states* and works in *phases*; each phase consists of a nontrivial iteration followed by zero or more trivial iterations and the state update;

(2) state10 indicates that the k-mer u processed in the nontrivial iteration is 10-projected;
- in the nontrivial iteration, pkey is computed by Definition 17(1);
- after that, the algorithm branches on $t = |\mathsf{tail}(h(u))|$;
- if $t \geq 2$: perform $k-t-1$ trivial iterations, remain in state10;
- if $t = 0$: switch to state0;
- if $t = 1$: set the counter i to be the number of trailing 1's in $h(u)$, switch to state1;

(3) state0 indicates $h(u) = 0^{k-1}h(a)$ for the k-mer u processed in the nontrivial iteration;
- after finishing the nontrivial iteration in accordance with Definition 17(2), the algorithm remains in state0 if $h(u) = 0^k$ and switches to state1 with $i = 1$ if $h(u) = 0^{k-1}1$;

(4) state1 indicates $h(u) = 0^{k-i-1}1^i h(a)$ for the k-mer u processed in the nontrivial iteration;
- if $h(a) = 0$, the algorithm assigns $\mathsf{pkey} = X_1$ by Definition 17(2), performs $k-3$ trivial iterations and switches to state10;
- if $h(a) = 1$ and $i < k-1$, the algorithm assigns $\mathsf{pkey} = X_\infty$ by Definition 17(2), increments i and remains in state1;
- if $h(a) = 1$ and $i = k-1$, the algorithm assigns $\mathsf{pkey} = X_3$ by Definition 17(2) and remains in state1;

(5) Initialization: read $S[1..k{-}3]$ and start the first iteration with reading $S[k{-}2]$ in state10; throw the first two computed keys as they do not represent k-mers.

▶ **Proposition 20.** KRspacer *solves the key-retrieval problem for DNA spacers.*

Proof. By Proposition 19, it suffices to prove that KRspacer assigns keys to all k-mers in S according to Definition 17. First we show that all transitions between states are correct. The transitions from state10 follow the definition of tail. If $\mathsf{tail}(h(u))$ begins with 10 (i.e., $t = |\mathsf{tail}(h(u))| \geq 2$), it determines the position of the next 10-projected k-mer to be $k-t$ symbols to the right of the currently processed k-mer u. Hence, it is correct to be in state10 after $k-t-1$ trivial iterations. If $\mathsf{tail}(h(u)) = \varepsilon$, then $h(u) = 10^{k-1}$, so KRspacer correctly switches to state0 for the next phase. Similarly, $\mathsf{tail}(h(u)) = 1$ implies $h(u) = 10^{k-i-1}1^i$, and KRspacer respectively switches to state1 with the correct value of i. The transitions from state0 and state1 are checked in the same way.

In nontrivial iterations, the keys are assigned by Definition 17. It remains to check that the assignment of infinite keys in trivial iterations satisfies the condition $(\star)$ of Definition 17. Let I be the set of consecutive k-mers that obtained infinite keys during trivial iterations of one phase. Note that the elements of I are not 10-projected. If the phase corresponds to state10, then $|I| = k - t - 1 \leq k - 3 < w$. The k-mers in I are preceded and followed in S by 10-projected k-mers. By definition, 10-projected k-mers have lower ranks than the k-mers in I. Hence, the assignment of infinite keys respects $(\star)$. Similarly, if the phase corresponds to state1, then $|I| = k - 3 < w$. The k-mers in I are preceded by the k-mer u with $\mathsf{pkey}(u) = X_1$, and followed by a 10-projected k-mer. By Remark 18(i), no k-mer in I has $\mathsf{pkey} = X_1$ according to Definition 17(2). Once again, the elements of I are surrounded in S by k-mers of lower ranks, and thus the assignment of infinite keys respects $(\star)$. ◀

4 Results

4.1 Expected density of random 10-minimizers: estimated vs. exact

To evaluate the actual error introduced by assumption $(\mathrm{A}^\dagger)$ and the closely related assumption $(\mathrm{A}^\ddagger)$ to the expected density of a random 10-minimizer, we designed an optimized enumeration algorithm (Supplementary Section A.1) that computes the *exact* expected density of a random 10-minimizer with the parameters $(2, k, w)$ via the sum, over all $(w+k)$-windows containing 10-k-mers, of probabilities of being charged by such a random 10-minimizer. Recall that the number of charged windows among those containing no 10-k-mers is given by Lemma 1. For $k \in \{7, 12\}$, $w \in [k-2..2k]$, we compared the estimate (1) to the exact expected density (Supplementary Table A1). The error of the estimate is very small: $< 0.11\%$ for $k = 7$ and $< 0.0022\%$ for $k = 12$, which is less than the error of the estimate $\frac{2}{w+1}$ for the density of random minimizers over the same ranges of parameters.

Thus, the estimate $\frac{2}{w+2}$ for the expected density of a random binary 10-minimizer is quite precise not only asymptotically but also for practical (w, k) ranges. This conclusion also extends to random balanced 10-minimizers over arbitrary even-sized alphabets as the simplifying assumption $(\mathrm{A}^\ddagger)$ is based on $(\mathrm{A}^\dagger)$.

4.2 Density results for DNA spacers

We compared the density achieved by DNA spacers to densities achieved by other constant-space minimizers, specifically: miniception [23], double-decycling [19], open-closed syncmer [18], and ABB+ [1, 7]. We included two baselines: one is the KGMLT lower bound [11] on the density of all *forward* schemes, including minimizers. The second baseline presents the lowest density achieved by non-constant-space minimizers generated by the state-of-the-art

GreedyMini method [6]. We do not consider mod-sampling-based schemes [10] as they are not minimizers in general[2] and, additionally, they are designed for $k > w$ (even $k \geq w + 4$ in the default setting), while DNA spacers are designed primarily for $w \geq k - 2$.

We measured the density of constant-space minimizers using a random DNA sequence of 2×10^6 nucleotides. We averaged the results of 5 runs over different sequences, ensuring to use different seeds for the minimizers that rely on randomness, i.e., miniception and open-closed-syncmer. For both miniception and open-closed-syncmer, which rank s-mers for some $s < k$, we used $s = 4$. The density measurement for GreedyMini minimizers is detailed in Supplementary Section A.4.

We measured the density in the modes "fixed k, varying w" for $k \in \{12, 24\}$, $3 \leq w \leq 100$ (Fig. 1 AB), and "fixed w, varying k" for $w \in \{12, 24\}$ and $3 \leq k \leq 32$ (Fig. 1 CD). For the DNA spacer and $w < k - 2$, we show the density of the k-extension (with lexicographic tie-breaking) of the DNA spacer for $(w{+}2, w)$.

For $k = 12$, DNA spacer beats all other constant-space minimizers for $w \geq 23$ and all minimizers, including GreedyMini-generated, for $w \geq 40$. For $k = 24$, double-decycling wins for $w \geq k$, but DNA spacer catches it up as w grows to 100. In the fixed w mode, DNA spacer is still competitive, especially for k small enough. In particular, it beats all constant-space minimizers for $w = 24$, $5 \leq k \leq 12$, and is better than GreedyMini for $k = 5, 6, 7$.

4.3 Key-retrieval times benchmarking

The key-retrieval time of a minimizer heavily depends on the hardware, so the absolute values may be insufficiently informative. However, as mentioned in Section 3.4.1, the lexicographical minimizer presents a natural baseline to refer to, as it does not compute k-mer keys.

We included in comparison all minimizers from the density benchmarking (DNA spacer, ABB+, miniception, open-closed syncmer, double-decycling, GreedyMini), random 10-minimizer (using a straightforward adaptation of the KRspacer algorithm), and random minimizer. We adapted all sampling algorithms to C++, implementing tricks used in their respective papers and repositories and improving upon them. As several minimizers rely on random orders, we emulated it by first XORing each k-mer (or s-mer in the case of miniception and open-closed syncmer) with the random seed and then hashing it. We experimented with two hash functions: C++ standard `std::hash`, and `PCG RXS-M-XS` 64-bit mixer [17]; the latter is very fast while satisfying the rigorous TestU01 BigCrush statistical tests [12].

We measured the processing time for a random DNA sequence of 1.5×10^8 nucleotides, averaging the results over a hundred runs to ensure a robust estimate. All experiments were conducted concurrently and shuffled using 6 cores (out of 8) on an AMD Ryzen 7800X3D system with 32 GB of DDR5 RAM. We measured GreedyMini separately, as it heavily relies on cache space. The results of measurement for $5 \leq k \leq 32$ are shown in Fig. 2. The main conclusions are as follows.

(1) Double-decycling, which performs well in terms of density, is very slow. In the key assignment, it relies on precise computations over complex numbers. Apparently, any known method that spends $O(1)$ time per key amplifies the precision error, eventually leading to incorrect key assignment. Otherwise, computing each key requires $\Theta(k)$ operations. In our experiment, all key-retrieval times for double-decycling exceeded $(400k + 200)$ms. Already at $k = 15$, this is four times slower than the second slowest minimizer; hence, we excluded double-decycling from Fig. 2.

[2] In [10, Lemma 13], it is claimed that mod-sampling schemes with a specific choice of the internal parameter are minimizers, but the proof is flawed. See Supplementary Section A.3 for details.

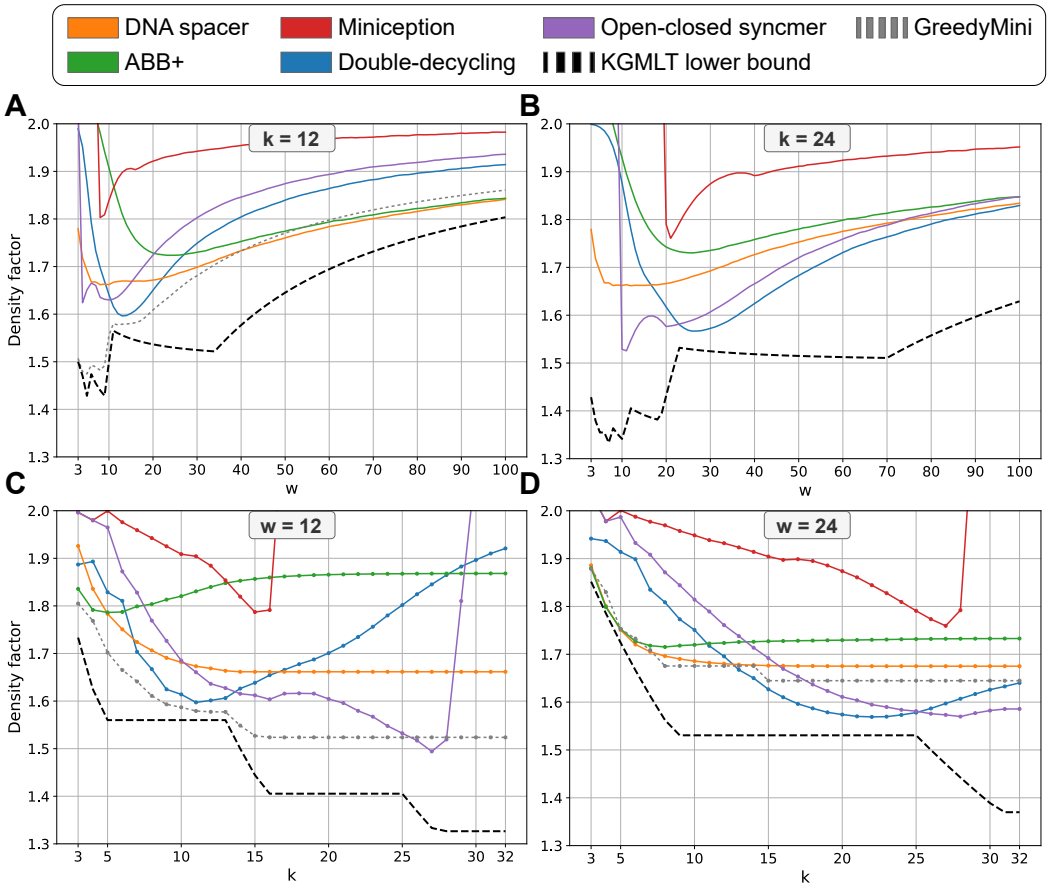

Figure 1 Performance of various DNA minimizers over certain ranges of (k, w) values. (A-B) Density comparisons over fixed $k \in \{12, 24\}$ and $3 \leq w \leq 100$. GreedyMini is not shown in B since $k = 24$ is beyond its recommended range in terms of both runtime and sampling time. (C-D) Density comparisons over fixed $w \in \{12, 24\}$ and $3 \leq k \leq 32$.

(2) The hashing speed has a large effect on key-retrieval times of randomized minimizers. Using `std::hash` instead of the `PCG` mixer increases the time to compute k-mer keys (i.e., the distance to the baseline) up to the factor of 4, reached by a random minimizer. The slowing effect is modest for a random 10-minimizer, as it hashes only 10-projected k-mers.

(3) The fastest constant-space minimizers are the `PCG`-mixer-based random minimizer and ABB+. They are followed by DNA spacer and random 10-minimizer (5–6 seconds per a 2Gbp-long sequence). Miniception and open-closed syncmer are significantly slower[3].

(4) For most constant-space minimizers, the key-retrieval time is independent of k as long a k-mer fits into a 64-bit machine word. The only exclusions are small k values for DNA-spacer and random 10-minimizer, where the key computation is a bit slower due to a smaller fraction of trivial iterations.

(5) Predictably, the time required by non-constant-space minimizers is determined by the cache size and behavior. In our experiment, GreedyMini was the fastest up to $k = 16$ and good up to $k = 21$, but fell out of competition since $k = 23$.

[3] While the pseudocode of the open-closed syncmer suggests that it is slower than miniception, our benchmarking demonstrated variable performance, likely due to the memory layout. See Supplementary Section A.5 for details.

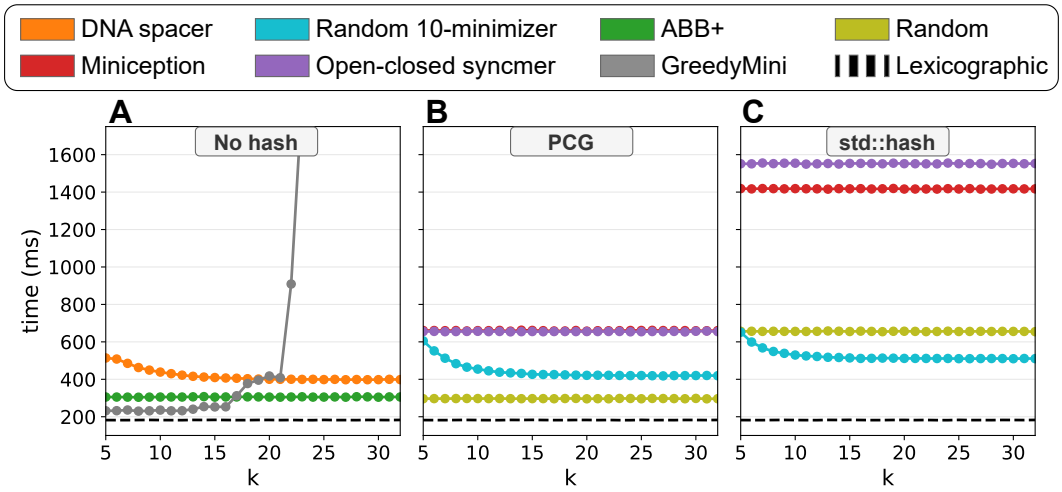

Figure 2 Key-retrieval times for k-mers over a 1.5×10^8 random DNA sequence, averaged over 100 runs, for $5 \leq k \leq 32$. (A) Minimizers requiring no hash function. (B) Minimizers using the `PCG RXS-M-XS` 64-bit mixer hash function [17]. (C) Minimizers using `std::hash`.

5 Discussion

10-minimizers represent a significant advance in k-mer sampling schemes for high-throughput sequencing analysis, introducing the first class of constant-space minimizers with provable density guarantees superior to random minimizers in the non-asymptotic regime. By proving that random 10-minimizers achieve an expected density of approximately $\frac{2}{w+2}$ (vs. $\frac{2}{w+1}$ for random minimizers), we established a theoretical foundation for their superiority across practical k, w ranges, validated by exact density computations showing approximation errors $< 0.0022\%$ for $k = 12$, $10 \leq w \leq 24$. While the difference between $\frac{2}{w+2}$ and $\frac{2}{w+1}$ seems small, it is not negligible for practical w ranges: e.g., for $k = 12$ the random 10-minimizer achieves lower density than miniception over the whole range of w values. Given their competitive key-retrieval time even for "heavy" hash functions, random 10-minimizers can replace random minimizers in applications that require minimizers randomly chosen from a large set.

DNA spacers, our flagship 10-minimizers, combine three desirable properties: (1) constant space (O(1) description, no explicit order storage); (2) low density, competitive with state-of-the-art schemes, and even achieving the lowest known density for certain (k, w) regimes (e.g., for $40 \leq w \leq 100$ and $k = 12$, or $5 \leq k \leq 7$ and $w = 24$); and (3) fast k-mer key retrieval, clearly outperforming constant-space minimizers of lower density. Two key observations that lead to the notion of DNA spacer are: (1) maximization of inter-minimizer distances via short tails prioritization reduces the density; (2) 10-minimizers using unbalanced projections achieve lower density compared to those using balanced projections.

The benchmarking on the new and apparently important metric of key-retrieval time revealed that none of existing methods produces "universally best" minimizers. When both k and w take small/medium values, GreedyMini is a clear winner in both density and key-retrieval time. However, its key-retrieval time for large k is prohibitive, while for large w it loses in terms of density to DNA spacers and ABB+. Double-decycling method is attractive in terms of density but lacks a fast key-retrieval algorithm, and it is unclear whether such an algorithm can exist. Hence, in the "large k" and "large w" regimes, the main competitors are DNA spacer, ABB+, and open-closed syncmer, with the DNA spacer being arguably the most attractive option for large w values.

There are several open questions and future steps that arise from our study. First, can we push the density of constant-space 10-minimizers further down, while keeping the key-retrieval time on the same level? Second, is it possible to obtain further theoretic density estimates, for example, on the expected density of a random unbalanced 10-minimizer or on the density of a DNA spacer? Third, can we improve current high-throughput sequencing analysis methods and advanced sampling schemes by plugging in the best constant-space minimizer for their (k, w) values?

To conclude, 10-minimizers, particularly spacers, offer a theoretically-grounded, practically-efficient alternative to existing schemes, poised to enhance high-throughput sequencing pipelines through superior density and competitive key-retrieval times.

References

1 Simon R Blackburn. Non-overlapping codes. *IEEE Transactions on Information Theory*, 61(9):4890–4894, 2015. doi:10.1109/TIT.2015.2456634.

2 Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, February 2021. doi:10.7717/peerj.10805.

3 Barış Ekim, Bonnie Berger, and Yaron Orenstein. A randomized parallel algorithm for efficiently finding near-optimal universal hitting sets. In *International Conference on Research in Computational Molecular Biology*, pages 37–53. Springer, 2020.

4 Martin C Frith, Laurent Noé, and Gregory Kucherov. Minimally overlapping words for sequence similarity search. *Bioinformatics*, 36(22-23):5344–5350, 2020. doi:10.1093/BIOINFORMATICS/BTAA1054.

5 Shay Golan and Arseny M. Shur. Expected density of random minimizers. In Rastislav Královic and Vera Kurková, editors, *SOFSEM 2025: Theory and Practice of Computer Science - 50th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2025, Proceedings, Part I*, volume 15538 of *Lecture Notes in Computer Science*, pages 347–360. Springer, 2025. doi:10.1007/978-3-031-82670-2_25.

6 Shay Golan, Ido Tziony, Matan Kraus, Yaron Orenstein, and Arseny Shur. GreedyMini: generating low-density DNA minimizers. *Bioinformatics*, 41(Supplement_1):i275–i284, 2025. doi:10.1093/BIOINFORMATICS/BTAF251.

7 Ragnar Groot Koerkamp. Optimal throughput bioinformatics (doctoral thesis), 2025. doi:10.3929/ethz-c-000783091.

8 Ragnar Groot Koerkamp, Daniel Liu, and Giulio Ermanno Pibiri. The open-closed mod-minimizer algorithm. *Algorithms Mol. Biol.*, 20(1):4, 2025. doi:10.1186/s13015-025-00270-0.

9 Ragnar Groot Koerkamp and Igor Martayan. SimdMinimizers: Computing Random Minimizers, fast. In Petra Mutzel and Nicola Prezza, editors, *23rd International Symposium on Experimental Algorithms (SEA 2025)*, volume 338 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:19, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SEA.2025.20.

10 Ragnar Groot Koerkamp and Giulio Ermanno Pibiri. The mod-minimizer: A simple and efficient sampling algorithm for long k-mers. In Solon P. Pissis and Wing-Kin Sung, editors, *24th International Workshop on Algorithms in Bioinformatics, WABI 2024*, volume 312 of *LIPIcs*, pages 11:1–11:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.WABI.2024.11.

11 Bryce Kille, Ragnar Groot Koerkamp, Drake McAdams, Alan Liu, and Todd J Treangen. A near-tight lower bound on the density of forward sampling schemes. *Bioinformatics*, 41(1):btae736, December 2024. doi:10.1093/bioinformatics/btae736.

12 Pierre L'Ecuyer and Richard Simard. Testu01: A c library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, 33(4):1–40, 2007. doi:10.1145/1268776.1268777.

13 Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, March 2016. doi:10.1093/bioinformatics/btw152.

14 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018. doi:10.1093/BIOINFORMATICS/BTY191.

15 Guillaume Marçais, David Pellow, Daniel Bork, Yaron Orenstein, Ron Shamir, and Carl Kingsford. Improving the performance of minimizers and winnowing schemes. *Bioinformatics*, 33(14):i110–i117, 2017. doi:10.1093/BIOINFORMATICS/BTX235.

16 Malick Ndiaye, Silvia Prieto-Baños, Lucy M Fitzgerald, Ali Yazdizadeh Kharrazi, Sergey Oreshkov, Christophe Dessimoz, Fritz J Sedlazeck, Natasha Glover, and Sina Majidian. When less is more: sketching with minimizers in genomics. *Genome Biology*, 25(1):270, 2024.

17 Melissa E. O'Neill. Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation. Technical Report HMC-CS-2014-0905, Harvey Mudd College, Claremont, CA, September 2014.

18 Yaron Orenstein, David Pellow, Guillaume Marçais, Ron Shamir, and Car Kingsford. Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing. *PLoS Computational Biology*, 13(10):e1005777, 2017. doi:10.1371/JOURNAL.PCBI.1005777.

19 David Pellow, Lianrong Pu, Bariş Ekim, Lior Kotlar, Bonnie Berger, Ron Shamir, and Yaron Orenstein. Efficient minimizer orders for large values of k using minimum decycling sets. *Genome Research*, 33(7):1154–1161, 2023.

20 Michael Roberts, Wayne Hayes, Brian R. Hunt, Stephen M. Mount, and James A. Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, July 2004. doi:10.1093/bioinformatics/bth408.

21 Kristoffer Sahlin. Effective sequence similarity detection with strobemers. *Genome Research*, 31(11):2080–2094, 2021. doi:10.1101/gr.275648.121.

22 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, pages 76–85, New York, NY, USA, 2003. Association for Computing Machinery. doi:10.1145/872757.872770.

23 Hongyu Zheng, Carl Kingsford, and Guillaume Marçais. Improved design and analysis of practical minimizers. *Bioinformatics*, 36(Supplement_1):i119–i127, 2020. doi:10.1093/BIOINFORMATICS/BTAA472.

24 Hongyu Zheng, Guillaume Marçais, and Carl Kingsford. Creating and using minimizer sketches in computational genomics. *Journal of Computational Biology*, 30(12):1251–1276, 2023. doi:10.1089/CMB.2023.0094.

A Appendix

A.1 Computing the exact density of random binary 10-minimizers

To compute the density of a random binary 10-minimizer with parameters k, w, it suffices to find, for each $(w+k)$-window containing at least one 10-k-mer, the probability of being charged (the case of windows with no 10-k-mers is covered by Lemma 1). For a $(w+k)$-window v with m distinct k-mers, this probability can be expressed as

$$\frac{[k\text{-prefix of } v \text{ is a 10-}k\text{-mer}] + [k\text{-suffix of } v \text{ is unique and is a 10-}k\text{-mer}]}{m}$$

The efficient enumeration of all binary $(w+k)$-windows is organized as follows.

(1) The set of all $(w+k)$-windows is viewed as a binary *prefix tree*; its nodes are all binary strings of length $\leq w+k$, and its edges are all pairs (u, ua), where $a \in \{0,1\}$. In particular, the leaves of this tree are exactly all the $(w+k)$-windows.

(2) The prefix tree is processed by a recursive DFS; each node za gets from its parent z the set of 10-k-mers occurring in z and the flag indicating whether the k-prefix of z is a 10-k-mer. Hence, if za is a leaf, the probability of za being charged is computed in $O(1)$ operations.

(3) To speed up the DFS traversal, the recursive routine is called for a node z only if the k-suffix of z is a 10-k-mer (and thus may change the set of occurred 10-k-mers inherited from the calling ancestor). If no descendants of a node z have 10-k-mers as suffixes, then all leaves in the subtree of z have the same probability of being charged; this probability is computed at z and multiplied by the number of those leaves.

We computed the densities of random binary 10-minimizers with $k = 7$ and $k = 12$ over ranges of w values and compared these results to the estimate $\mathcal{R}^*(2, k, w)$ by Theorem 5. They are presented in Table A1, normalized to density factors. In all cases, the error of the estimate $\mathcal{R}^*(2, k, w)$ is less than 0.11% for $k = 7$ and less than 0.0022% for $k = 12$. For the same ranges, the maximum error of the density estimate for *random minimizers* is higher: greater than 0.25% for $k = 7$ and greater than 0.0036% for $k = 12$. This confirms that our density estimates for random 10-minimizers are good for the practical ranges of parameters.

Table A1 The exact vs. estimated (by Theorem 5) values of the expected density factor for random 10-minimizers. For comparison, the exact and estimated values of the expected density factor of random minimizers are provided for the same ranges.

(a) $\sigma = 2$, $k = 7$.

	Random 10-minim.		Random minim.	
w	exact	estim.	exact	estim.
5	1.711426	1.710938	2.005127	2.000000
6	1.749573	1.750000	2.005025	2.000000
7	1.780111	1.779134	2.003988	2.000000
8	1.801437	1.801428	2.003471	2.000000
9	1.820292	1.819305	2.002756	2.000000
10	1.834556	1.834117	2.002331	2.000000
11	1.848158	1.846664	2.001902	2.000000
12	1.858888	1.857462	2.001677	2.000000
13	1.868592	1.866860	2.001520	2.000000
14	1.877106	1.875114	2.001489	2.000000

(b) $\sigma = 2$, $k = 12$.

	Random 10-minim.		Random minim.	
w	exact	estim.	exact	estim.
10	1.832525	1.832504	2.000072	2.000000
11	1.845776	1.845739	2.000054	2.000000
12	1.856962	1.856936	2.000044	2.000000
13	1.866605	1.866564	2.000031	2.000000
14	1.874966	1.874949	2.000022	2.000000
15	1.882349	1.882328	2.000013	2.000000
16	1.888892	1.888876	2.000006	2.000000
17	1.894742	1.894731	1.999999	2.000000
18	1.900005	1.899997	1.999994	2.000000
19	1.904763	1.904760	1.999989	2.000000
20	1.909094	1.909090	1.999985	2.000000
21	1.913039	1.913043	1.999981	2.000000
22	1.916660	1.916666	1.999978	2.000000
23	1.919991	1.920000	1.999975	2.000000
24	1.923064	1.923077	1.999972	2.000000

A.2 Density of binary spacers vs. binary ABB+ minimizers

We compared the densities of binary spacers and binary ABB+ orders for $w = 24$ and $k = 8, \ldots, 26$, estimating them on a random sample of 10^8 $(w+k)$-windows for each pair (k, w) (Supplementary Table A2). We note that the estimate of the expected density factor of a random 10-minimizer (Theorem 5), rounded to 6 decimal places, equals 1.923077 for all points in the analysed range.

Table A2 Comparing the density factors of binary spacer and binary ABB+ minimizer for $w = 24$. The densities are estimated from a sample of 10^8 random $(w + k)$-windows for each pair (k, w).

k	8	9	10	11	12	13	14	15	16
$D_{\mathsf{spacer}}(2,k,24)$	1.744185	1.741076	1.741098	1.742237	1.742177	1.742321	1.742793	1.743476	1.742896
$D_{\mathsf{ABB+}}(2,k,24)$	1.854068	1.857927	1.858477	1.860609	1.858185	1.860894	1.860402	1.861672	1.861706

$\cdots$	17	18	19	20	21	22	23	24	25	26
$\cdots$	1.743550	1.743088	1.742920	1.742048	1.744163	1.741836	1.742611	1.742745	1.742599	1.743465
$\cdots$	1.862441	1.861741	1.861548	1.860859	1.860694	1.859866	1.861647	1.860313	1.858494	1.859848

A.3 A comment on mod-minimizers

[10, Def. 7] defines *mod-sampling scheme* with the parameters (σ, k, w) as follows: an internal parameter $t \leq k$ and a (pseudo)random linear order ρ on Σ^t are chosen; in each window v, the leftmost position x of the ρ-minimal t-mer is computed, and the k-mer at position x mod w is sampled. (The positions in any string are numerated starting with 0.) We note that the case $t = k$ is trivial, as it implies x mod $w = x$, and thus the scheme coincides with the "internal" minimizer (ρ, w). So it may be safely assumed $t < k$.

[10, Lemma 13] claims that the mod-sampling scheme is a minimizer (called there *mod-minimizer*) if the parameter t satisfies $t \equiv k \pmod{w}$. Let us give a counterexample.

Suppose there exist two k-mers u_1 and u_2, sharing a common t-prefix z, and two windows: v_1 begins with u_1 and contains u_2, while v_2 begins with u_2 and contains u_1. As the order ρ of t-mers is random, we can assume that z is the ρ-minimal t-mer in both v_1 and v_2. Then the mod-sampling scheme samples the k-prefix in both v_1 and v_2. However, no minimizer can sample this way: if u_1 has smaller (respectively, larger) rank than u_2, then the sampling for v_2 (respectively, for v_1) would be invalid. A particular example follows.

Let $\sigma = 2, w = 8, t = 7, k = 15$. The positions of u_2 in v_1 and of u_1 in v_2 are indicated in boldface. The example can be easily generalized to $k = 2i + 7$ for any $i \geq 1$.

$$
\begin{aligned}
u_1 &= 010101010\ 010101\\
u_2 &= 0101010\ 01010101\\
v_1 &= u_1 0101010 = 01\mathbf{0101010\ 0101010}1010\\
v_2 &= u_2 0010101 = 0101010\mathbf{10101010\ 010101}
\end{aligned}
$$

The conclusion is that (i) the condition $t \equiv k \pmod{w}$ does not guarantee that the mod-sampling scheme is a minimizer, and (ii) currently there is no class of mod-sampling schemes with $t < k$ that is **proved** to consist of minimizers.

A.4 Calculating density of GreedyMini orders

To measure the density of GreedyMini orders, we conducted measurements using a random DNA sequence of 10^7 nucleotides, or incorporated densities reported in the GreedyMini study [6]. The density values were measured as follows:

(1) **For $k = 12$ and $w \in [3, 100]$:** We utilized the density values reported in the GreedyMini study. Additionally, we evaluated published GreedyMini orders for $(k, w) \in \{(12, 20), (12, 30)\}$ across smaller w values. For each parameter set, we reported the minimum density observed between the original published results and our supplementary measurements.

(2) **For $k = 24$ and $w \in [3, 100]$:** GreedyMini densities were omitted for this range, as this k value exceeds the recommended k range for GreedyMini.

(3) **For $w = 12$ and $k \in [3, 32]$:** We utilized the densities reported in the GreedyMini study. For k values not explicitly covered in the original publication, we estimated the density by applying the k-extension with lexicographic tie-breaking to the published GreedyMini order constructed for the largest available k and $w = 12$.

(4) **For $w = 24$ and $k \in [3, 32]$:** In the absence of published GreedyMini orders for these specific parameters, we calculated the density by applying the following orders (using k-extensions as needed) to measure the density for the setting of $(k, w) = (k', 24)$:

- All $(k', 15)$ orders for $3 \leq k' \leq 14$.
- The $(15, 17)$ orders for $15 \leq k' \leq 32$.
- The $(15, 30)$ orders for $15 \leq k' \leq 32$.
- The $(8, 19)$ orders for $8 \leq k' \leq 32$.

For every (k, w) configuration, we report the lowest density observed across all tested orders.

A.5 Key-retrieval times for the open-closed syncmer vs. miniception

The pseudocode of open-closed syncmer suggests it is theoretically slower than miniception due to an additional `if`. However, our benchmarking demonstrated fluctuating performance. Specifically, for $k = 16$, miniception was faster in half of the experimental runs for `std::hash`, while open-closed syncmer outperformed it in the other half. These differences, though alternating in direction, remained statistically significant per each run ($p < 10^{-9}$, Wilcoxon rank-sum test). After aligning the s-mer rank structures to memory addresses divisible by 64, both schemes were slower, yet miniception consistently outperformed open-closed syncmer, hinting at variability due to memory layout.