



DeepMinimizer: A Differentiable Framework for Optimizing Sequence-Specific Minimizer Schemes

Minh Hoang¹(✉) , Hongyu Zheng² , and Carl Kingsford²

¹ Computer Science Department, Carnegie Mellon University,
Pittsburgh, PA 15213, USA
qhoang@andrew.cmu.edu

² Computational Biology Department, Carnegie Mellon University,
Pittsburgh, PA 15213, USA
hongyuz1@andrew.cmu.edu, carlk@cs.cmu.edu

Abstract. Minimizers are k -mer sampling schemes designed to generate sketches for large sequences that preserve sufficiently long matches between sequences. Despite their widespread application, learning an effective minimizer scheme with optimal sketch size is still an open question. Most work in this direction focuses on designing schemes that work well on expectation over random sequences, which have limited applicability to many practical tools. On the other hand, several methods have been proposed to construct minimizer schemes for a specific target sequence. These methods, however, require greedy approximations to solve an intractable discrete optimization problem on the permutation space of k -mer orderings. To address this challenge, we propose: (a) a reformulation of the combinatorial solution space using a deep neural network re-parameterization; and (b) a fully differentiable approximation of the discrete objective. We demonstrate that our framework, DEEP-MINIMIZER, discovers minimizer schemes that significantly outperform state-of-the-art constructions on genomic sequences.

Keywords: Sequence sketching · Optimization · Deep learning

1 Introduction

Minimizers [15, 16] are deterministic methods to sample k -mers from a sequence at approximately regular intervals such that sufficient information about the identity of the sequence is preserved. Sequence sketching with minimizers is widely used to reduce memory consumption and processing time in bioinformatics programs such as read mappers [7, 10], k -mer counters [3, 5] and genome assemblers [17]. Given a choice of k -mer length k and window length w , a minimizer selects the lowest priority k -mer from every overlapping window in the target sequence according to some total ordering π over all k -mers. Minimizer performance is measured by its density [12] on a target sequence, which is proportional to the induced sketch size.

Depending on the choice of π , the resulting density can significantly vary. The theoretical lower-bound of density achievable by any minimizer scheme is given by $1/w$ [12]. On the other hand, a random initialization of π will yield an expected density of approximately $2/w$ [16], which is frequently used as a baseline for comparing minimizer performance. This motivates the question: How do we effectively optimize π to improve the performance of minimizers?

An exhaustive search over the combinatorial space of π suffices for very small k , but quickly becomes intractable for values of k used in practice (i.e., $k \geq 7$) (Sect. 3.1). To work around this, many existing approaches focus on constructing minimizer schemes from mathematical objects with appealing properties such as universal hitting sets (UHS) [4, 11, 12, 14, 19]. While these schemes provide upper-bound guarantees for expected densities on random sequences, they only obtain modest improvements over a random minimizer when used to sketch a specific sequence [19].

The idea of learning minimizer schemes tailored towards a target sequence has been previously explored, although to a lesser extent. Current approaches include heuristic designs [1, 8], greedy pruning [2] and construction of k -mer sets that are well-spread on the target sequence [20]. However, these methods only learn crude approximations of π by dividing k -mers into disjoint subsets with different priorities to be selected. Within each subset, the relative ordering is arbitrarily assigned to recover a valid minimizer, hence they are not necessarily optimal. We give a detailed overview of these methods in Sect. 2.

This paper instead tackles the problem of directly learning a total order π . The hardness of solving such a task comes from two factors, which we will review in detail in Sect. 3.1: (1) the search space of k -mer orderings is very large; and (2) the density minimizing objective is discrete. To overcome the above challenges, we propose to reformulate the original problem as parameter optimization of a deep learning system. This results in the first fully-differentiable minimizer selection framework that can be efficiently optimized using gradient-based learning techniques. Our contributions are:

- We define a more well-behaved search space that is suitable for gradient-based optimization. This is achieved by implicitly representing k -mer orderings as continuous score assignments. The space of these assignments is parameterized by a neural network called **PRIORITYNET**, whose architecture guarantees that every output assignment is *consistent* (i.e., corresponding to valid minimizer schemes). The modelling capacity of **PRIORITYNET** can be controlled via increasing its architecture depth, which implies a mild restriction on the candidate space in practice (Sect. 3.2).
- We approximate the discrete learning objective by a pair of simpler tasks. First, we design a complementary neural network called **TEMPLATENET**, which outputs potentially *inconsistent* assignments (i.e., template) with guaranteed low densities on the target sequence (Sect. 3.4). We then search for *consistent* assignments (i.e., valid minimizers) around these templates, which potentially will yield similar densities. This is achieved via a fully differentiable proxy objective (Sect. 3.3) that minimizes a novel divergence (Sect. 3.5) between these networks.

- We compare our framework, DEEPMINIMIZER, against various state-of-the-art benchmarks and observe that DEEPMINIMIZER yields sketches with significantly lower densities on genomic sequences (Sect. 4).

2 Related Work

UHS-Based Methods. Most existing minimizer selection schemes with performance guarantees over random sequences are based on the theory of universal hitting sets (UHS) [11, 14]. Particularly, a (w, k) -UHS is defined as a set of k -mers such that every window of length w (from any possible sequence) contains at least one of its elements. Every UHS subsequently defines a family of corresponding minimizer schemes whose expected densities on random sequences can be upper-bounded in terms of the UHS size [12]. As such, to obtain minimizers with provably low density, it suffices to construct small UHS, which is often the common learning objective of many existing approaches [4, 12, 19].

In the context of *sequence-specific* minimizers, there are several concerns with this approach. First, the requirement of UHS to “hit” all windows of *every possible* sequence is often too strong with respect to the need of sketching a specific string and results in sub-optimal universal hitting sets [20]. Additionally, since real sequences rarely follow a uniform distribution [18], there tends to be little correspondence between the provable upper-bound on expected density and the actual density measured on a target sequence. In practice, the latter is usually more pessimistic on sequences of interest, such as the human reference genome [19, 20], which drives the development of various *sequence-specific* minimizer selection methods.

Heuristic Methods. Several minimizer construction schemes rank k -mers based on their frequencies in the target sequence [1, 8], such that rare k -mers are more likely to be chosen as minimizers. These constructions nonetheless rely on the assumption that rare k -mers are spread apart and ideally correspond to a sparse sampling. Another greedy approach is to sequentially remove k -mers from an arbitrarily constructed UHS, as long as the resulting set still hits every w -long window on the target sequence [2]. Though this helps to fine-tune a given UHS with respect to the sequence of interest, there is no guarantee that such an initial set will yield the optimal solution after pruning.

Polar Set Construction. Recently, a novel class of minimizer constructions was proposed based on polar sets of k -mers, whose elements are sufficiently far apart on the target sequence [20]. The sketch size induced by such a polar set is shown to be tightly bounded with respect to its cardinality. This reveals an alternate route to low-density minimizer schemes through searching for the minimal polar set. Unfortunately, this proxy objective is NP-hard and currently approximated by a greedy construction [20].

Remark. In all of the above methods, the common objective to be optimized can be seen as a partition of the set of all k -mers into disjoint subsets. For example, frequency values are used to denote different buckets of k -mers [1, 8]. Others [2, 4, 19, 20] employ a more fine-grained partitioning scheme defined by the constructed UHS/polar set. Each subset has an assigned priority value, such that k -mers from higher priority subsets are always chosen over k -mers from lower priority subsets. However, it remains inconclusive how k -mers from within the same subset can be optimally selected to recover a total ordering π . Practically, these methods resort to using a pre-determined arbitrary ordering to resolve such situations. In contrast, our work investigates a novel approach to directly learn this ordering.

3 Methods

3.1 Background

Let Σ be an alphabet of size $|\Sigma| = \sigma$ and S be a sequence containing exactly l overlapping k -mers defined on this alphabet, i.e., $S \in \Sigma^{l+k-1}$. For some $w \in \mathbb{N}^+$ such that $l \geq w$, we define a (w, k) -window as a substring in S of length $w+k-1$, which contains exactly w overlapping k -mers. For ease of notation, we further let $l_w \triangleq l - w + 1$ denote the number of (w, k) -windows in S . For the rest of this paper, we assume that w and k are fixed and given as application-specific parameters.

Definition 1 (Minimizer). A minimizer scheme $m : \Sigma^{w+k-1} \rightarrow [1..w]$ is uniquely specified by a total ordering π on Σ^k . Here, we encode π as a function $\rho : \Sigma^k \rightarrow \mathbb{N}^+$ that maps k -mers to its position in π . Given a (w, k) -window ω , m then returns the smallest k -mer in ω according to ρ :

$$m(\omega; \pi) \triangleq \operatorname{argmin}_{i \in [1..w]} \rho(\omega[i]; \pi) \equiv \operatorname{argmin}_{i \in [1..w]} \sum_{s \in \Sigma^k} \mathbb{I}(s <_{\pi} \omega[i]) , \quad (1)$$

where $\mathbb{I}$ denotes the indicator function, $\omega[i]$ denotes the i -th k -mer in ω , and $s <_{\pi} \omega[i]$ implies s precedes $\omega[i]$ in π . We break ties by prioritizing k -mers that occur earlier in (i.e., to the left of) the window.

When applied to a sequence S , the above scheme selects one k -mer position from every overlapping window to construct the sequence sketch $\mathcal{L}(S; m) = \{t + m(\omega_t) \mid t \in [1, l_w]\}$, with ω_t denoting the t^{th} window in S . Naturally, a smaller sketch leads to more space and cost savings. As such, we measure minimizer performance by the density factor metric $\mathcal{D}(S; m) \triangleq |\mathcal{L}(S; m)| \times (w+1)/l_w$, which approximates the number of k -mers selected per window [12]. The minimizer selection problem is then formalized as density minimization with respect to π :

$$\pi_* = \operatorname{argmin}_{\pi} \mathcal{D}(S; m(\cdot; \pi)) \equiv \operatorname{argmin}_{\pi} |\mathcal{L}(S; m(\cdot; \pi))|. \quad (2)$$

This objective, however, is intractable to optimize for two reasons. First, the number of all k -mer permutations scales super-exponentially with k and σ (i.e.,

$\sigma^k!$), thus renders any form of exhaustive search on this space impossible under most practical settings. Furthermore, the set counting operation $|\mathcal{L}(S; m(\cdot; \pi))|$ is non-differentiable even if the solution space is continuous, which makes efficient gradient-based optimizers inaccessible. The remainder of this section therefore proposes a deep-learning strategy to address both these challenges, and is organized as follows.

Section 3.2 describes a unifying view of existing methods as reparameterizations of ρ (Definition 1). We then propose a novel deep parameterization called **PRIORITYNET**, which relaxes the permutation search space of Eq. 2 into a well-behaved weight space of a neural network.

Section 3.3 shows that density optimization with respect to **PRIORITYNET** can be approximated by two sub-tasks via introducing another complementary network, called **TEMPLATENET**. This approximation can be formalized as a fully-differentiable proxy objective that minimizes divergence between **TEMPLATENET** and **PRIORITYNET**.

Section 3.4 and Sect. 3.5 then respectively discuss the parameterization of **TEMPLATENET** and the divergence measure in our proxy objective, thus completing the specification of our framework, **DEEPMINIMIZER**. An overview of our framework is given in Fig. 1.

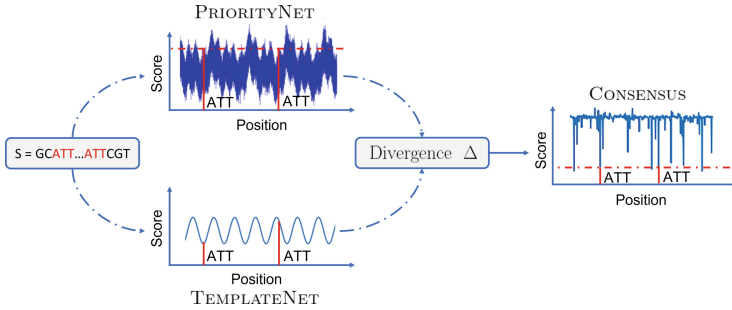


Fig. 1. Our **DEEPMINIMIZER** framework employs a twin network architecture. **PRIORITYNET** generates valid minimizers, but has no guarantee on density. In contrast, **TEMPLATENET** generates low-density templates that might not correspond to valid minimizers. We minimize the divergence between these networks to arrive at consensus minimizers with low densities on the target sequence.

3.2 Search Space Reparameterization

We first remark that many existing methods can be seen as different reparameterizations of ρ in Definition 1. For example, ρ can be parameterized with frequency information from the target sequence [1, 8], i.e., $\rho(\omega_i; S) \propto \sum_{j=1}^{l_w} \mathbb{I}(\omega_j = \omega_i)$; or instantiated with a UHS v [4, 19], i.e., $\rho(\omega_i; v) = \mathbb{I}(\omega_i \notin v)$. Similar set-ups have been explored in the context of sequence-specific minimizers using a pruned UHS $v(S)$ [2] and a polar set $\zeta(S)$ [20] constructed for the target

sequence. We note that there are fewer discrete values potentially assigned by ρ than the total number of k -mers in all these re-parameterizations. As such, these methods still rely on a pre-determined arbitrary ordering to break ties in windows with two or more similarly scored k -mers. When collisions occur frequently, this could have unexpected impact on the final density.

DEEPMINIMIZER instead employs a continuous parameterization of ρ using a feed-forward neural network parameterized by weights α , which takes as input the multi-hot encoding of a k -mer (i.e., a concatenation of its character one-hot encodings) and returns a real-valued score in $[0, 1]$. This continuous scheme practically eliminates the chance for scoring collisions. Furthermore, the solution space of this re-parameterization is only restricted by the modelling capacity encoded by our architecture weight space. This limitation quickly diminishes as we employ sufficiently large number of hidden layers in the network. We can subsequently rewrite Eq. 2 as optimizing a neural network with density as its loss function:

$$\alpha_* = \underset{\alpha}{\operatorname{argmin}} \mathcal{D}(S; \rho(\cdot; \alpha)) . \quad (3)$$

Applying this network on every k -mer along S can be compactly written as a convolutional neural network, denoted by f , which maps the entire sequence S to a *score assignment* vector. We require this score assignment to be *consistent* across different windows in order to recover a valid ordering π from such implicitly encoded ρ . Specifically, one k -mer can not be assigned different scores at different locations in S . To enforce this, we let the first convolution layer of our architecture, PRIORITYNET, have kernel size k , and all subsequent layers to have kernel size 1. An illustration for PRIORITYNET when $k = 2$ is given in Fig. 2.

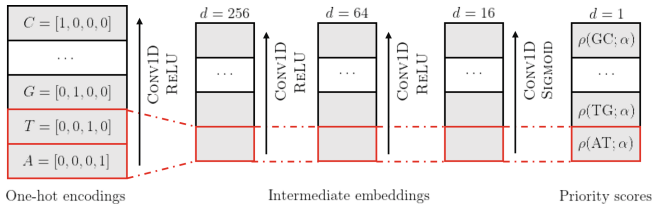


Fig. 2. Our PRIORITYNET architecture for $k = 2$, parameterized by weights α , maps sequence multi-hot encoding to priority scores through a series of 3 convolution layers with kernel size $[k, 1, 1]$ and $[256, 64, 16]$ embedding channels respectively. Fixing network weights α , the computation of assigned priority score to any k -mer is deterministic given its character one-hot encodings.

3.3 Proxy Objective

The density computation in Eq. 3, however, is not differentiable with respect to the network weights. As such, α cannot be readily optimized with established

gradient back-propagation techniques used in most deep learning methods. To work around this, we introduce a proxy optimization objective that approximates Eq. 3 via coupling PRIORITYNET with another function called TEMPLATENET. Unlike the former, TEMPLATENET relaxes the *consistency* requirement and generates *template* score assignments that might not correspond to valid minimizer schemes. In exchange, such *templates* are guaranteed to yield low densities by design.

Intuitively, the goals of these networks are complementary: PRIORITYNET generates valid minimizer schemes in the form of *consistent* priority score assignments, whereas TEMPLATENET pinpoints neighborhoods of low-density score assignments situated around its output templates. This reveals an alternative optimization route where these networks negotiate towards a consensus solution that (a) satisfies the constraint enforced by PRIORITYNET; and (b) resembles a template in the output space of TEMPLATENET, thus potentially yielding low density. Let f and g denote our proposed PRIORITYNET and TEMPLATENET, respectively parameterized by weights α and β , we formalize this objective as minimizing some divergence measure Δ between their outputs:

$$(\alpha_*, \beta_*) = \underset{\alpha, \beta}{\operatorname{argmin}} \Delta(f(S; \alpha), g(S; \beta)) \quad (4)$$

In the remainder of this paper, we detail the full specification of our proxy objective, which requires two other ingredients. First, Sect. 3.4 discusses the parameterization of our TEMPLATENET g to consistently generate templates that achieve the theoretical lower-bound density [12] on the target sequence. Furthermore, we note that the proxy objective in Eq. 4 will perform best when the divergence measure Δ reflects the difference in densities of two score assignments. Section 3.5 then discusses a practical choice of Δ to accurately capture high-performing neighborhoods of minimizers. These specifications have strong implications on the expressiveness of the solution space and directly influences the performance of our framework, as shown in Sect. 4.

3.4 Specification of TEMPLATENET

The well-known theoretical lower bound $1 + 1/w$ for density factor [12] implies that the optimal minimizer, if it exists, samples k -mers exactly w positions apart. As a result, we want to guarantee that the output of TEMPLATENET approximates this scenario given any weights initialization. Without loss of generality, we impose that TEMPLATENET is given by a continuous function $g : \mathbb{R} \rightarrow [0, 1]$, such that its output template $\mathbf{v} = [g(i)]_{i \in [U]}$ consists of evaluations of g restricted to integer inputs (i.e., k -mer positions). Then, Proposition 1 below shows a sufficient construction for g that approximately yields the optimal density.

Proposition 1. *Let $g : \mathbb{R} \rightarrow [0, 1]$ be a periodic function with minimal period w , such that g has a unique minimum value on every w -long interval. Formally, g satisfies:*

$$(1) : \forall t \in \mathbb{R} : g(t) = g(t + w)$$

$$(2) : \forall i, j \in \underset{t}{\operatorname{arginf}} g(t), \exists u \in \mathbb{N} : |i - j| = uw .$$

Then, the template generated by g induces a sketch with density factor $1 + 1/w + o(1)$ on S when S is sufficiently long (i.e., $l_w \gg w^2$).

Proof. We give a detailed proof of Proposition 1 in Appendix A.

Note that the resulting sketch induced by g does not necessarily correspond to a valid minimizer. While this sketch has low density, it does not preserve the sequence identity like a minimizer sketch, hence is not useful for downstream applications. However, it is sufficient as a guiding template to help PRIORITYNET navigate the space of orderings.

Proposition 1 leaves us with infinitely many candidate functions to choose from. In fact, TEMPLATENET can be as simple as $g(t) = \sin(2\pi t/w)$ to generate a near-optimal score assignment. This naïve specification, however, encodes exactly one template (i.e., one that picks k -mers from the set of interval positions $\{w, 2w, \dots\}$), whose proximal neighborhood might not contain any valid minimizer scheme. For example, consider a sequence S in which some particular k -mer occurs exactly at positions $t \in \{\frac{1}{2}w, \frac{3}{2}w, \dots\}$. Ideally, we would want to align the *template minima* with these locations, which is not possible given the above choice of g . As such, it is necessary that the specification of TEMPLATENET is sufficiently expressive for Eq. 4 to find an optimal solution.

In particular, we want to construct a parameterized function such that every k -mer position can be sampled by at least one sketch encoded in its parameter space. Furthermore, we note that the periodic property is only a sufficient condition to obtain low-density sketches. In practice, we only want the template minima to periodically occur at fixed intervals. Enforcing the scores assigned at all positions to exactly follow a sinusoidal pattern is restrictive and might lead to overlooking good templates. To address these design goals, we propose the following ensemble parameterization:

$$g(t) = \sigma \left(\sum_{\phi=0}^{w-1} \beta_{\phi} \sin \left(\frac{2\pi}{w} (t + \phi) \right) \right) , \quad (5)$$

where σ denotes a sigmoid activation function, which ensures that $g(t)$ appropriately maps to $[0, 1]$; $\beta = \{\beta_{\phi}\}_{\phi=0}^{w-1}$ are optimizable parameters such that $\beta_{\phi} \geq 0$ and $\sum_{\phi=1}^w \beta_{\phi} = 1$.

Optimizing β has two implications. First, by adjusting the dominant phase shift $\phi_{\max} = \operatorname{argmax}_{\phi} \beta_{\phi}$, we can control the offset of the periodic template minima, which leads to good coverage on the target sequence. Second, by adjusting the magnitudes of the remaining phase shifts $\{\beta_{\phi}\}_{\phi \neq \phi_{\max}}$, we can have more degrees of freedom to assign scores outside the template minima. Lastly, the non-negative and sum-to-one constraints help to avoid the trivial assignment of squashing all magnitudes to 0 and are easily guaranteed by letting β be the output of a softmax layer.

3.5 Specification of the Divergence Measure Δ

As standard practice, we first consider instantiating Δ with the squared ℓ^2 -distance. Specifically, let $\mathbf{v}_f = f(S; \alpha)$ and $\mathbf{v}_g = g(S; \beta)$ denote the score assignments respectively output by PRIORITYNET and TEMPLATENET given S , then $\Delta_{\ell^2}(\mathbf{v}_f, \mathbf{v}_g) \triangleq \sum_{i=1}^l (\mathbf{v}_f[i] - \mathbf{v}_g[i])^2$. This divergence measure, however, places an excessively strict matching objective at all locations along $\mathbf{v}_f$ and $\mathbf{v}_g$. Such a perfect match is unnecessary as long as the k -mers outside sampled locations are assigned higher scores, and will take away the degrees of freedom needed for the proxy objective to satisfy the constraints implied by PRIORITYNET.

Consequently, we are interested in constructing a divergence that: (a) strategically prioritizes matching $\mathbf{v}_f$ to the minima of the template $\mathbf{v}_g$; and (b) enables flexible assignment at other positions to admit more solutions that meet the consistency requirement. To accomplish these design goals, we propose the following asymmetrical divergence:

$$\Delta(\mathbf{v}_f, \mathbf{v}_g) \triangleq \sum_{i=1}^l \left[(1 - \mathbf{v}_g[i]) \cdot (\mathbf{v}_f[i] - \mathbf{v}_g[i])^2 + \lambda \cdot \mathbf{v}_g[i] \cdot (1 - \mathbf{v}_f[i])^2 \right]. \quad (6)$$

Specifically, the idea behind the first component $(1 - \mathbf{v}_g[i]) \cdot (\mathbf{v}_f[i] - \mathbf{v}_g[i])^2$ in the summation is to weight each position-wise matching term $(\mathbf{v}_f[i] - \mathbf{v}_g[i])^2$ by its corresponding template score: the weight term $(1 - \mathbf{v}_g[i])$ implies stronger matching preference around the minima of $\mathbf{v}_g$ where the template scores $\mathbf{v}_g[i]$ are low, and vice versa weaker preference at other locations. Furthermore, to ensure that f properly assigns higher scores to the locations outside the minima of $\mathbf{v}_g$, the second component $\mathbf{v}_g[i] \cdot (1 - \mathbf{v}_f[i])^2$ subsequently encourages f to maximize its assigned scores wherever possible, again weighted by the relative relevance of each location. The trade-off between these components is controlled by the hyper-parameter λ . Finally, we confirm that this divergence measure is fully differentiable with respect to α and β , hence can be efficiently optimized using gradient-based techniques. Particularly, the parameter gradients of both networks are given by:

$$\begin{aligned} \frac{\partial}{\partial \alpha} \Delta(\mathbf{v}_f, \mathbf{v}_g) &= \sum_{i=1}^l a_i \cdot \frac{\partial}{\partial \alpha} \mathbf{v}_f[i] \\ \frac{\partial}{\partial \beta} \Delta(\mathbf{v}_f, \mathbf{v}_g) &= \sum_{i=1}^l b_i \cdot \frac{\partial}{\partial \beta} \mathbf{v}_g[i], \end{aligned} \quad (7)$$

where the gradients of network outputs are obtained via back-propagation and their respective constants are given by $a_i = 2 \cdot (1 - \mathbf{v}_g[i]) \cdot (\mathbf{v}_f[i] - \mathbf{v}_g[i]) + 2\lambda \cdot \mathbf{v}_g[i] \cdot (\mathbf{v}_f[i] - 1)$ and $b_i = 2 \cdot (\mathbf{v}_g[i] - 1) \cdot (\mathbf{v}_f[i] - \mathbf{v}_g[i]) - (\mathbf{v}_f[i] - \mathbf{v}_g[i])^2 + \lambda \cdot \mathbf{v}_g[i] \cdot (\mathbf{v}_f[i] - 1)$.

4 Results

Implementation Details. We implement our method using PyTorch and deploy all experiments on a RTX-2060 GPU. Due to limited GPU memory, each training epoch only computes a batch divergence which averages over $N = 10$ randomly sampled subsequences of length $l = 500 \times (w + k)$. We set $\lambda = 1$ and use architectures of PRIORITYNET and TEMPLATENET as given in Fig. 2 and Sect. 3.4 respectively. Network weights are optimized using the ADAM optimizer [9] with learning rate $\eta = 5 \times 10^{-3}$. Our implementation is available at <https://github.com/Kingsford-Group/deepminimizer>.

Comparison Baselines. We compare DEEPMINIMIZER with the following benchmarks: (a) random minimizer baseline; (b) Miniception [19]; (c) PASHA [4]; and (d) PolarSet Minimizer [20]. Among these methods, (d) is a sequence-specific minimizer scheme. For each method, we measure the density factor $\mathcal{D}$ obtained on different segments of the human reference genome: (a) chromosome 1 (CHR1); (b) chromosome X (CHRX); (c) the centromere region of chromosome X [13] (which we denote by CHRXC); and (d) the full genome (HG38). We used lexicographic ordering for PASHA as suggested by [19]. Random ordering is used to rank k -mers within the UHS for Miniception, and outside the layered sets for PolarSet.

Visualizing the Mechanism of DEEPMINIMIZER. First, we show the transformation of the priority scores assigned by SCORENET and TEMPLATENET over 600 training epochs. Figure 3 plots the outputs of these networks evaluated on positions 500 to 1000 of CHRXC, and their corresponding locations of sampled k -mers. Initially, the PRIORITYNET assignment resembles that of a random minimizer and expectedly yields $\mathcal{D} = 2.05$. After training, the final

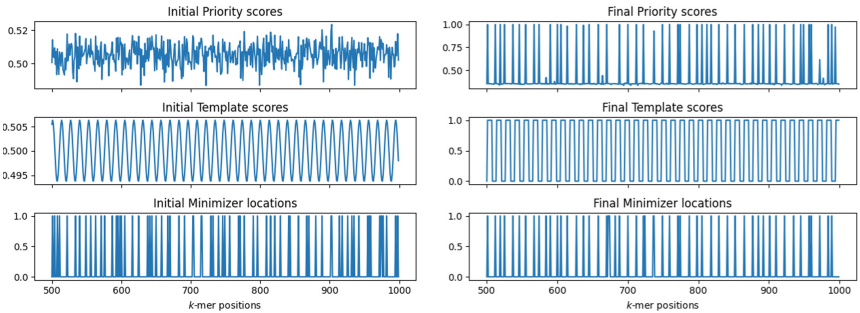


Fig. 3. Visualization of PRIORITYNET and TEMPLATENET score assignments on positions 500–1000 of CHRXC with $w = 13$, $k = 8$. Left: Initial assignments ($\mathcal{D} = 2.05$); Right: Final assignments after 600 training epochs ($\mathcal{D} = 1.39$). The bottom plots show corresponding locations of sampled k -mers: a value of 1 means selected, and 0 otherwise.

TEMPLATENET assignment converges with a different phase shift than its initial assignment, but its period remains the same. Simultaneously, the PRIORITYNET assignment learns to match this template, hence induces a visibly sparser sketch with $\mathcal{D} = 1.39$. This result clearly demonstrates the negotiating behaviour of our twin architecture to find optimal neighborhood of score assignments.

Convergence of Our Proxy Objective. We further demonstrate that our proxy objective meaningfully improves minimizer performance as it is optimized. The first two columns of Fig. 4 show the best density factors achieved by our method over 600 epochs on two scenarios: (a) varying k with fixed w ; and (b) varying w with fixed k . The experiment is repeated on CHRXC and HG38. In every scenario, DEEPMINIMIZER starts with $\mathcal{D} \simeq 2.0$, which is only comparable to a random minimizer. We observe steady decrease of $\mathcal{D}$ over the first 300 epochs before reaching convergence, where total reduction ranges from 11–23%.

Generally, larger k values lead to better performance improvement at convergence. This is expected since longer k -mers are more likely to occur uniquely in the target sequence, which makes it easier for a minimizer to achieve sparse sampling. In fact, previous results have shown that when k is much smaller than $\log w$, no minimizer will be able to achieve the theoretical lower-bound $\mathcal{D}$ [19]. On the other hand, larger w values lead to smaller improvements and generally slower convergence. This is because our ensemble parameterization of TEMPLATENET scales with the window size w and becomes more complicated to optimize as w increases.

Evaluating Our Proposed Divergence Measure. The last column of Fig. 4 shows the density factors achieved by our DEEPMINIMIZER method, respectively specified by the proposed divergence function in Eq. 6 and ℓ^2 -divergence. Here, we fix $w = 14$ and vary $k \in \{6, 8, 10, 12, 14\}$ and observe that with the ℓ^2 -divergence, we only obtain performance similar to a random minimizer. On the other hand, with our divergence function, DEEPMINIMIZER obtains much lower densities on all settings, thus confirming the intuition in Sect. 3.5.

Comparing Against Other Minimizer Selection Benchmarks. We show the performance of DEEPMINIMIZER compared to other benchmark methods. DEEPMINIMIZER is trained for 600 epochs to ensure convergence, as shown above. Figure 5 shows the final density factors achieved by all methods, again on two comparison scenarios: (a) fix $w = 13$, and vary $k \in \{6, 8, 10, 12, 14\}$; and (b) fix $k = 14$, and vary $w \in \{10, 25, 40, 55, 70, 85\}$.

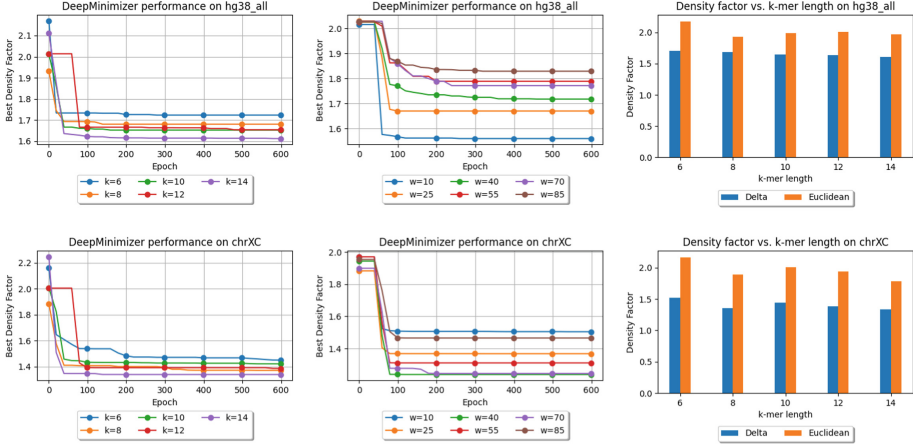


Fig. 4. Best density factors obtained by DEEPMINIMIZER on HG38 (above) and CHRXC (below) over 600 training epochs. Left: fix $w = 13$, and vary $k \in \{6, 8, 10, 12, 14\}$; Center: fix $k = 14$, and vary $w \in \{10, 25, 40, 55, 70, 85\}$; Right: Comparing proposed Δ -divergence and ℓ^2 -divergence.

DEEPMINIMIZER consistently achieves better performance compared to *non-sequence-specific* minimizers (i.e., PASHA, Miniception) on all settings. We observe up to 40% reduction of density factor (e.g., on CHRXC, $w = 70$, $k = 14$), which clearly demonstrates the ability of DEEPMINIMIZER to exploit *sequence-specific* information. Furthermore, we also observe that DEEPMINIMIZER outperforms our *sequence-specific* competitor, PolarSet, in a majority of settings. The improvements over PolarSet are especially pronounced for smaller k values, which are known harder tasks for minimizers [19]. On larger w values, our method performs slightly worse than PolarSet in some settings. This is likely due to the added complexity of optimizing TEMPLATENET, as described in convergence ablation study of our method.

In addition, we also conduct investigation on the centromere region of chromosome X (i.e., CHRXC), which contains highly repetitive subsequences [6] and has been shown to hamper performance of PolarSet [20]. Figure 5 shows that PolarSet and the UHS-based methods perform similarly to a random minimizer, whereas our method is consistently better. Moreover, we observe that DEEPMINIMIZER obtains near-optimal densities with CHRXC on several settings. For example, we achieved $\mathcal{D} = 1.22$ when $k = 14$, $w \in \{40, 70\}$, which is significantly better than the results on CHR1 and CHRX. This suggests that CHRXC is not necessarily more difficult to sketch, but rather good sketches have been excluded by the UHS and polar set reparameterizations, which is not the case with our framework.

Runtime Performance. DEEPMINIMIZER runs efficiently with GPU computing. In all of our experiments, each training epoch takes approximately 30 seconds

to 2 minutes, depending on the choice of k and w , which controls the batch size. Performance evaluation takes between several minutes (CHRX) to 1 hour (HG38), depending on the length of the target sequence. Generally, our method is cost-efficient without frequent evaluations. Our most cost-intensive experiment (i.e., convergence ablation study on HG38) requires a full-sequence evaluation every 20 epochs over 600 epochs, thus takes approximately 2 days to complete. This is faster than PolarSet, which has a theoretical runtime of $\mathcal{O}(n^2)$ and takes several days to run with HG38. A more detailed runtime ablation study on CHR1 is provided in Appendix B.

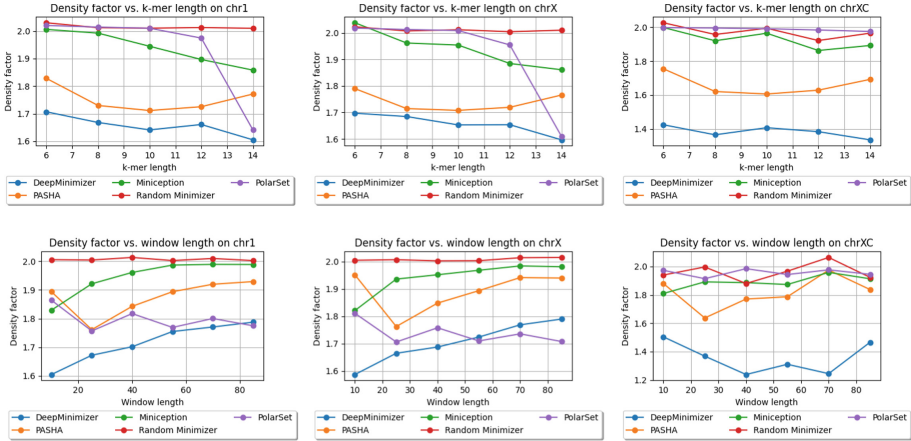


Fig. 5. Density factors obtained by DEEPMINIMIZER (600 training epochs), Random Minimizer, PASHA, Miniception and PolarSet on CHR1, CHRX and CHRX. Above: fix $w = 13$, and vary $k \in \{6, 8, 10, 12, 14\}$; Below: fix $k = 14$, and vary $w \in \{10, 25, 40, 55, 70, 85\}$.

5 Conclusion

We introduce a novel framework called DEEPMINIMIZER for learning *sequence-specific* minimizers. This is achieved via casting minimizer selection as optimizing a k -mer scoring function ρ . We propose a more well-behaved search space for minimizers, given by a neural network parameterization of ρ , called PRIORITYNET. Then, we introduce a complementary network, called TEMPLATENET which pinpoints optimal scoring templates and guides PRIORITYNET to the neighborhood of low-density assignments around them. Coupling these networks leads to a fully differentiable proxy objective that can effectively leverage gradient-based learning techniques. DEEPMINIMIZER obtains better performance than state-of-the-art sequence-agnostic and sequence-aware minimizer selection schemes, especially on known hard tasks such as sketching the repetitive centromere region of Chromosome X. However, we also observe mild limitations

in several settings with large window length w , which hampers the performance of DEEPMINIMIZER. This is likely due to the heuristic construction of our TEMPLATENET component, which we will investigate in our future work.

Acknowledgements. This work was supported in part by the Gordon and Betty Moore Foundation’s Data-Driven Discovery Initiative [GBMF4554 to C.K.], by the US National Institutes of Health [R01GM122935], and the US National Science Foundation [DBI-1937540]. Conflict of Interest: C.K. is a co-founder of Ocean Genomics, Inc.

A Proof of Proposition 1

We first re-express the density factor of S in terms of a priority score assignment $\mathbf{v} \in [0, 1]^l$. Note that this expression will hold regardless of whether $\mathbf{v}$ satisfies the consistency constraint (Sect. 3.2). Particularly, let $\gamma_1 = 1$ and $\gamma_t = \mathbb{I}\left(\underset{j \in \omega_t}{\operatorname{argmin}} \mathbf{v}[j] \neq \underset{j' \in \omega_{t-1}}{\operatorname{argmin}} \mathbf{v}[j']\right)$ indicate the event the t -th window picks a different k -mer than the $(t-1)$ -th window, we have:

$$\mathcal{D}(S; \mathbf{v}) = \frac{w+1}{l_w} \times |\mathcal{L}(S; \mathbf{v})| = \frac{w+1}{l_w} \sum_{t=1}^{l_w} \gamma_t. \quad (8)$$

Without loss of generality, we assume $0 \in \underset{t}{\operatorname{arginf}} g(t)$ since this can always be achieved via adding a constant phase shift to g . As g has a fundamental period of w , this implies $\{uw \mid u \in \mathbb{N}\} \subseteq \underset{t}{\operatorname{arginf}} g(t)$, which further reduces to $\{uw \mid k \in \mathbb{N}\} = \underset{t}{\operatorname{arginf}} g(t)$ when condition (2) holds.

Let us now derive the values of γ_t for $t \in \mathcal{I}_u \triangleq [(u-1)w+1, uw]$, $u \in \mathbb{N}^+$. We have:

- $uw \in \underset{t}{\operatorname{arginf}} g(t)$,
- $\forall t \in \mathcal{I}_u$ such that $t \neq uw$, we have $t \notin \underset{t}{\operatorname{arginf}} g(t)$, and
- $\forall t \in \mathcal{I}(u) : uw \in \omega_t$, which follows from the above argument and the definition of window ω_t .

Together, these observations imply that $\forall t \in \mathcal{I}_u : \underset{j \in \omega_t}{\operatorname{argmin}} \mathbf{v}[j] = uw$ and consequently $\gamma_t = 0$ for all values of $t \in \mathcal{I}_u$ except $t = (u-1)w+1$. For $u = 1$, we trivially have $\gamma_{(u-1)w+1} = 1$ by definition of γ_1 . For $u > 1$, we have $\underset{j \in \omega_{(u-1)w}}{\operatorname{argmin}} \mathbf{v}[j] = (u-1)w \neq \underset{j \in \omega_{(u-1)w+1}}{\operatorname{argmin}} \mathbf{v}[j]$, which also implies $\gamma_{(u-1)w+1} = 1$. Following the above derivations, we have:

$$D(S; \mathbf{v}) = \frac{w+1}{l_w} \sum_{t=1}^{l_w} \gamma_t = \frac{w+1}{l_w} \left(c + \sum_{u=1}^{\lfloor \frac{l_w}{w} \rfloor} \sum_{t \in \mathcal{I}_u} \gamma_t \right) = \frac{w+1}{l_w} \left(c + \left\lfloor \frac{l_w}{w} \right\rfloor \right), \quad (9)$$

where the third equality follows from the derived values γ_t for $t \in \mathcal{I}_u$. Finally, using the fact that $c = \sum_{t=\lfloor \frac{l_w}{w} \rfloor w+1}^{l_w} \gamma_t < w$ and the sufficient length assumption

$l_w \gg w^2$, we have:

$$\frac{w+1}{l_w} \left(c + \left\lfloor \frac{l_w}{w} \right\rfloor \right) \leq \frac{w^2}{l_w} + \frac{w+1}{w} = 1 + \frac{1}{w} + o(1), \quad (10)$$

which concludes our proof. $\square$

B Other Empirical Results

This section contains extra experiments that showcase various aspects of our DEEPMINIMIZER framework. For all experiments, we use the same implementation, benchmarks and settings as detailed in Sect. 4.

Density Performance of DEEPMINIMIZER on More Sequence Baselines.

We deploy DEEPMINIMIZER on CHR1 and CHR X. For both sequences, we observe the best density factor obtained over 600 training epochs for various values of k and w . Figure 6 shows that DEEPMINIMIZER consistently improves density factors until convergence, which tends to happen between 200–300 training epochs for all experiments.

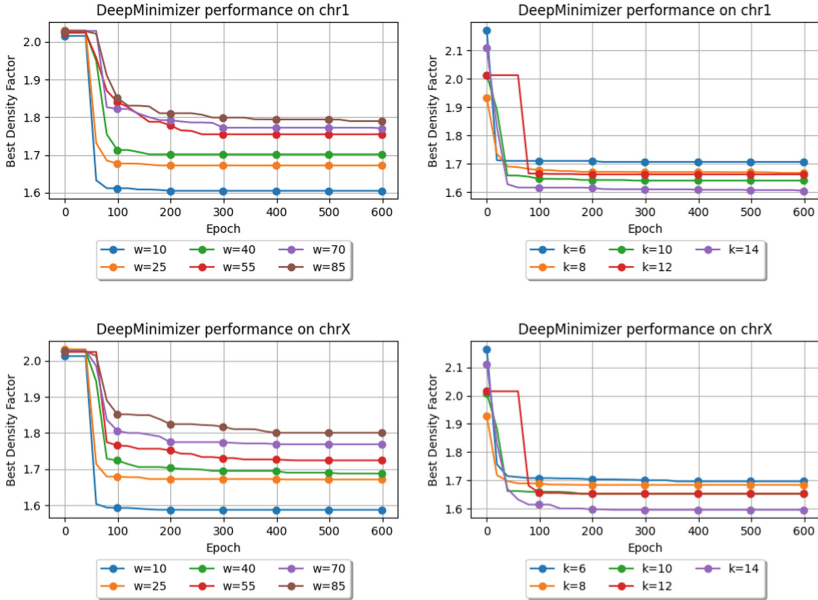


Fig. 6. Demonstrating convergence of DEEPMINIMIZER on CHR1 (left) and CHR X (right) with different w, k values.

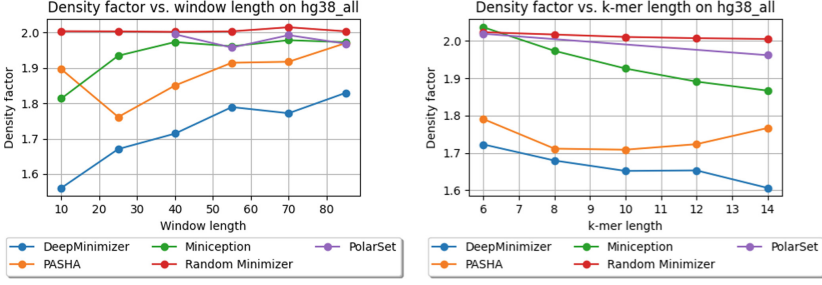


Fig. 7. Comparing performance of DEEPMINIMIZER with other benchmarks on HG38 for different values of w, k .

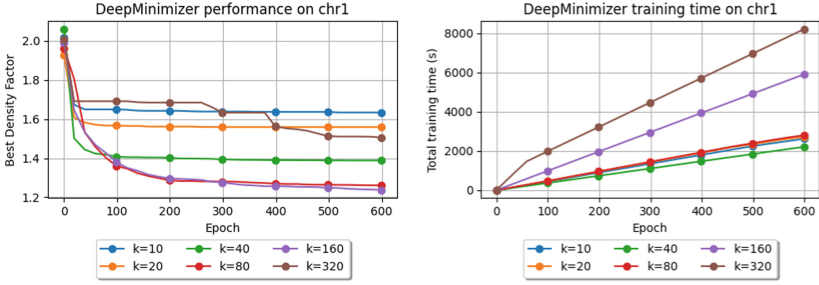


Fig. 8. Best density obtained (left) and runtime (right) of DEEPMINIMIZER for $k \in \{10, 20, 40, 80, 160, 320\}$ on CHR1.

DEEPMINIMIZER Outperforms Other Baselines on Large Sequences.

Figure 7 compares the performance of DEEPMINIMIZER and various comparison baselines on the entire human genome HG38. We measure the best density factor obtained over 600 training epochs for various values of k and w and observe that DEEPMINIMIZER consistently achieves the best performance among comparison baselines.

Density Performance of DEEPMINIMIZER on Large Values of k . Figure 8 (left) showcases the performance of DEEPMINIMIZER on CHR1 with large values of k . We fix $w = 13$ and observe the best density factor obtained over 600 training epochs for various values of k up to 320. We show that DEEPMINIMIZER behaves similarly for large k , and achieves the best density $\mathcal{D} = 1.22$ with $k = 160$.

Runtime Performance of DEEPMINIMIZER on Large Values of k . Figure 8 (right) measures runtime (in seconds) of DEEPMINIMIZER on CHR1 over 600 epochs. Larger k values require PRIORITYNET to have more parameters. Expectedly, we observe runtime for $k = 40, 80, 160, 320$ to increase in the same order. For $k = 10$ and 20, however, the runtimes are approximately the same as $k = 80$. We note that a smaller k value means there are more k -mers in the same sequence.

As such, even though PRIORITYNET is more compact for these values of k , we will incur some overhead from querying it more often.

References

1. Chikhi, R., Limasset, A., Medvedev, P.: Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics* **32**(12), i201–i208 (2016)
2. DeBlasio, D., Gbosibo, F., Kingsford, C., Marçais, G.: Practical universal k-mer sets for minimizer schemes. In: *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, BCB 2019*, pp. 167–176. Association for Computing Machinery, New York (2019)
3. Deorowicz, S., Kokot, M., Grabowski, S., Debudaj-Grabysz, A.: KMC 2: fast and resource-frugal k-mer counting. *Bioinformatics* **31**(10), 1569–1576 (2015)
4. Ekim, B., Berger, B., Orenstein, Y.: A randomized parallel algorithm for efficiently finding near-optimal universal hitting sets. *bioRxiv* (2020)
5. Erbert, M., Rechner, S., Müller-Hannemann, M.: Gerbil: a fast and memory-efficient k-mer counter with GPU-support. *Algorithms Mol. Biol.* **12**(1), 1–12 (2017)
6. Fukagawa, T., Earnshaw, W.C.: The centromere: chromatin foundation for the kinetochore machinery. *Dev. Cell* **30**(5), 496–508 (2014)
7. Jain, C., Rhie, A., Hansen, N., Koren, S., Phillippy, A.M.: A long read mapping method for highly repetitive reference sequences. *bioRxiv* (2020)
8. Jain, C., Rhie, A., Zhang, H., Chu, C., Walenz, B.P., Koren, S., Phillippy, A.M.: Weighted minimizer sampling improves long read mapping. *Bioinformatics* **36**(Suppl._1), i111–i118 (2020)
9. Kingma, D.P., Ba, J.: ADAM: a method for stochastic optimization. *arXiv preprint [arXiv:1412.6980](https://arxiv.org/abs/1412.6980)* (2014)
10. Li, H.: Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* **34**(18), 3094–3100 (2018)
11. Marçais, G., DeBlasio, D., Kingsford, C.: Asymptotically optimal minimizers schemes. *Bioinformatics* **34**(13), i13–i22 (2018)
12. Marçais, G., Pellow, D., Bork, D., Orenstein, Y., Shamir, R., Kingsford, C.: Improving the performance of minimizers and winnowing schemes. *Bioinformatics* **33**(14), i110–i117 (2017)
13. Miga, K.H., et al.: Telomere-to-telomere assembly of a complete human X chromosome. *Nature* **585**(7823), 79–84 (2020)
14. Orenstein, Y., Pellow, D., Marçais, G., Shamir, R., Kingsford, C.: Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing. *PLOS Comput. Biol.* **13**, e1005777 (2017)
15. Roberts, M., Hayes, W., Hunt, B., Mount, S., Yorke, J.: Reducing storage requirements for biological sequence comparison. *Bioinformatics* **20**, 3363–9 (2005)
16. Schleimer, S., Wilkerson, D., Aiken, A.: Winnowing: local algorithms for document fingerprinting. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data 10* (2003)
17. Ye, C., Ma, Z.S., Cannon, C.H., Pop, M., Douglas, W.Y.: Exploiting sparseness in de novo genome assembly. In: *BMC Bioinformatics*, vol. 13, pp. 1–8. BioMed Central (2012)

18. Zhang, Z.D., et al.: Statistical analysis of the genomic distribution and correlation of regulatory elements in the encode regions. *Genome Res.* **17**(6), 787–797 (2007)
19. Zheng, H., Kingsford, C., Marçais, G.: Improved design and analysis of practical minimizers. *Bioinformatics* **36**(Suppl._1), i119–i127 (2020)
20. Zheng, H., Kingsford, C., Marçais, G.: Sequence-specific minimizers via polar sets. *Bioinformatics* **37**, i187–i195 (2021)