Fast Alignment-Free Similarity Estimation By Tensor Sketching

Amir Joudaki[1,2,3], Gunnar Rätsch[1,2,3,*], and André Kahles[1,2,3,*]

[1] Department of Computer Science, ETH Zurich, Zurich 8092, Switzerland
{ajoudaki,raetsch,andre.kahles}@inf.ethz.ch
[2] University Hospital Zurich, Biomedical Informatics Research, Zurich 8091, Switzerland
[3] SIB Swiss Institute of Bioinformatics, Lausanne 1015, Switzerland
[*] to whom correspondence should be addressed

Abstract. The sharp increase in next-generation sequencing technologies' capacity has created a demand for algorithms capable of quickly searching a large corpus of biological sequences. The complexity of biological variability and the magnitude of existing data sets have impeded finding algorithms with guaranteed accuracy that efficiently run in practice. Our main contribution is the Tensor Sketch method that efficiently and accurately estimates edit distances. In our experiments, Tensor Sketch had 0.88 Spearman's rank correlation with the exact edit distance, almost doubling the 0.466 correlation of the closest competitor while running 8.8 times faster. Finally, all sketches can be updated dynamically if the input is a sequence stream, making it appealing for large-scale applications where data cannot fit into memory.

Conceptually, our approach has three steps: 1) represent sequences as tensors over their sub-sequences, 2) apply tensor sketching that preserves tensor inner products, 3) implicitly compute the sketch. The sub-sequences, which are not necessarily contiguous pieces of the sequence, allow us to outperform k-mer-based methods, such as min-hash sketching over a set of k-mers. Typically, the number of sub-sequences grows exponentially with the sub-sequence length, introducing both memory and time overheads. We directly address this problem in steps 2 and 3 of our method. While the sketching of rank-1 or super-symmetric tensors is known to admit efficient sketching, the sub-sequence tensor does not satisfy either of these properties. Hence, we propose a new sketching scheme that completely avoids the need for constructing the ambient space.

Our tensor-sketching technique's main advantages are three-fold: 1) Tensor Sketch has higher accuracy than any of the other assessed sketching methods used in practice. 2) All sketches can be computed in a streaming fashion, leading to significant time and memory savings when there is overlap between input sequences. 3) It is straightforward to extend tensor sketching to different settings leading to efficient methods for related sequence analysis tasks. We view tensor sketching as a framework to tackle a wide range of relevant bioinformatics problems, and we are confident that it can bring significant improvements for applications based on edit distance estimation.

Keywords: sequence distance metrics, tensor sketching, edit distance estimation

1 Introduction

The emergence of next-generation sequencing technologies and a dramatic decrease in cost have led to an exponential increase in biological sequence data, frequently stored in exceptionally large databases. While alignment scores are considered the gold-standard of sequence distance in many bioinformatics applications, the growing number of sequences to be analyzed poses serious challenges to exact distance computation via alignment. This problem has motivated research on time- and space-efficient alignment-free methods that try to estimate rather than compute sequence similarity. Especially applications relying on comparing millions of sequences have turned towards choosing approximate methods over exact alignments [2]. Out of many possibilities, we have selected the task of phylogeny reconstruction to further motivate distance estimation. In this task, the estimated distances between sequences are used to reconstruct the evolutionary history in form of a phylogenetic tree. Instead of using exact alignment scores, many alignment-free methods instead rely on k-mer statistics as a proxy. (Where the term *k-mer* refers to all length k substrings of a given string.) The multiplicity, frequency, mode, and reappearance statistics of k-mers have been utilised to directly estimate evolutionary distance [17,16,1]. Other approaches include variable lengths matches in their statistics [18].

To break the stringent dependency of adjacent letters in a k-mer, *spaced k-mers* have introduced "match" and "ignore" positions. For example, if a match-pattern "11011" is used, then "CTGAC"

versus "CTTAC" constitutes a match. Spaced k-mers have been shown to improve mapping sensitivity, the accuracy of phylogenies, and the performance of sequence classification [5,7,13,15]. Analogously, substring-based methods can be relaxed to allow for some mismatches [6]. Since the quality of the estimations will greatly depend on the selected match-pattern, several works have focused on optimizing these patterns [3,11,8,10]. However, finding combinatorially optimal patterns becomes intractable as the number of ignore-positions increases. Furthermore, any optimization is prone to be task-dependent, which would require the optimization to be repeated for each task separately.

More recently, hashing based methods have become increasingly popular. MinHash sketch methods have been primarily used for set similarity estimation [2], that have shown promise in fast genomic and metagenomic distance estimation by representing sequences as collections of informative substrings [12]. From locality sensitive hashing literature we know that any sketch over the ℓ^p-norm ($p \in (0, 2]$) will automatically lead to sub-quadratic nearest neighbor search by locality sensitive hashing [4]. This provides further motivation for focusing on sketching the edit distance, leaving behind the inherent problems of seed-based methods. To our knowledge, the most accurate sketching method currently available is *Ordered MinHash (OMH)* [9], which dramatically improves accuracy of k-mer MinHash, by considering k-mers that are non-adjacent. Therefore, we will compare our approach against the classical k-mer MinHash, and Ordered MinHash as the current state-of-the-art.

An especially difficult task for most current methods is the estimation of edit distance for longer, less closely related sequences. To illustrate the main limitations of using k-mer statistics in this context, we look at two examples that act as opposing forces in deciding the size of k.

Example 1. Consider two sequences $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{A}^N$ drawn randomly over the alphabet of size $\#\mathcal{A} = 4$, and let $v_{\boldsymbol{x}}, v_{\boldsymbol{y}} \in \mathbb{R}^{4^k}$ denote the k-mer frequency profile for each sequence. For long sequences with $N \gg 4^k$, k-mer frequencies will converge to their mean, that is 4^{-k} for all their components, which implies that $\|v_x - v_y\| \to 0$. Therefore, any k-mer profile-based method will severely *underestimate* distance between these two random sequences. In order to avoid this, k has to be restricted to larger values $k \gtrsim \log_4(N)$.

Example 2. Now, consider a second pair of sequences, where $\boldsymbol{y}$ is generated by substituting every character of $\boldsymbol{x}$ with 8% probability. Plugging this value back into our lower bound for k, the probability that a k-mer would not overlap any substitution equals $(1/8)^{\log_4(N)} = N^{-1.5}$. Therefore, the total number of k-mers not overlapping any mutation will be $N \cdot N^{-1.5} = (\frac{1}{N})^{1/2}$ which converges to 0 as N grows. Since the sequences share a 92% similarity in global alignment by design, the frequency vectors are guaranteed to *overestimate* the distance.

By combining the insights from these examples, we conclude that any value of k will fail with high probability in one of these scenarios for distantly related sequences. Since both results hold for k-mer frequency vectors in general, any such statistic on the k-mer profile is also bound to fail. This inherent problem with k-mers is our main motivation for designing sketches that are more resilient to mutations.

Conceptually, our method can be seen as expectation over all spaced seeds, without any limits on the number of *ignore* positions. In other words, Tensor Sketching resolves the inherent limitations of any seed-based method by taking an average of all possible seeds, and thereby reducing the risk of selecting a seed that reduces the sensitivity. In statistical terms, this implies that the resulting sketch is a Bayesian admissible estimator.

We will begin by introducing our notation and terminology in Section 2.1 and then proceed to presenting the Tensor Sketch and Tensor Slide sketch methods in Sections 2.2 and 2.3, respectively. In subsequent Section 3 we first summarize our data generation scheme and then compare our approach to the available state-of-the-art methods. We then discuss applications such as phylogeny reconstruction and graph distance estimation. In the final Section 4 we put our work into context and give an outlook into future work.

2 Methods

2.1 Preliminaries and Notation

Sets We use $\mathbb{1}\{\cdot\}$ to denote the indicator function, taking a value of 1 when logical expression between the brackets is true, and zero otherwise. For integer N, $[N]$ denotes the integer set $[N] := \{1, \ldots, N\}$. For finite set S, let $|S|$ and $\#S$ interchangeably denote its cardinality, and define $\mathcal{H}_S$ to be all bijective maps from S to $[|S|]$

$$\mathcal{H}_S := \Big\{ h : S \mapsto [|S|] \mid \{h(a) : a \in S\} \text{ is a permutation} \Big\}. \tag{1}$$

Finally, we use $X \sim S$ to imply drawing X uniformly from set S.

Vectors Throughout the manuscript bold face fonts are used to distinguish vectors, tensors, and sequences, from scalar variables $\mathbf{a} = (a_1, a_2)$. $\mathbf{1}^N$ and $\mathbf{0}^N$ are all-ones and all-zeros vectors of length N, respectively. We use $i \overset{N}{\equiv} j$ when integers i and j are equal modulus N. The circular r-shift $\text{shift}_r(\mathbf{a})$, will shift the elements of $\mathbf{a}$ to the right $\text{shift}_r(\mathbf{a}) = (a_{N-r+1}, \ldots a_N, a_1, \ldots, a_{N-r})$, defined formally as

$$\text{shift}_r(\mathbf{a}) := (a_{\sigma(i)})_{i \in [N]}, \qquad \sigma(i) \overset{N}{\equiv} i - r,\ \sigma\colon [N] \mapsto [N] \tag{2}$$

where the mapping σ, circularly shifts indices to the right.

Strings Let $\mathcal{A}, \mathcal{A}^N$, and $\mathcal{A}^\star$ denote the alphabet, strings of length N, and all strings respectively. Let $\boldsymbol{x} \circ \boldsymbol{y}$ denote concatenation of $\boldsymbol{x}$ and $\boldsymbol{y}$, and $|\boldsymbol{x}|$ be the length of string $\boldsymbol{x}$. We use $\boldsymbol{x}_i$ to denote the i-th character. Define $\boldsymbol{x}_{[i:j]}$ to be a slice from ith to jth index $\boldsymbol{x}_{[i:j]} := \boldsymbol{x}_i \ldots \boldsymbol{x}_j$, referred to as a *substring* of $\boldsymbol{x}$, and define $\boldsymbol{x}_{[\mathbf{i}]} := \boldsymbol{x}_{i_1} \ldots \boldsymbol{x}_{i_k}$ to be a *subsequence* of $\boldsymbol{x}$, when $1 \le i_1 < \cdots < i_k \le |\boldsymbol{x}|$ are strictly increasing indices of $\boldsymbol{x}$. For two strings $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{A}^\star$, the edit distance $\mathrm{d_{ed}}(\boldsymbol{x}, \boldsymbol{y})$ denotes the minimum number of edit operations needed to transform one string to the other. It can be defined recursively using $\boldsymbol{x}' = \boldsymbol{x}_{[2:|\boldsymbol{x}|]}, \boldsymbol{y}' = \boldsymbol{y}_{[2:|\boldsymbol{y}|]}$ as follows

$$\mathrm{d_{ed}}(\boldsymbol{x}, \boldsymbol{y}) := \min\big\{\mathrm{d_{ed}}(\boldsymbol{x}', \boldsymbol{y}) + 1, \mathrm{d_{ed}}(\boldsymbol{x}, \boldsymbol{y}') + 1, \mathrm{d_{ed}}(\boldsymbol{x}', \boldsymbol{y}') + \mathbb{1}\{\boldsymbol{x}_1 \neq \boldsymbol{y}_1\}\big\}, \tag{3}$$

with $\mathrm{d_{ed}}(\varepsilon, \varepsilon) := 0$ as the recursion basis.

Minimum hash sketching For sequence $\boldsymbol{x} \in \mathcal{A}^N$, let $X := (X_i)_{i \in [N-k+1]}$ denote its k-mer sequence, with X_i denoting the ith k-mer $X_i := \boldsymbol{x}_{[i:i+k-1]}$, and $\#X_i$ the occurrence number of this k-mer $\#X_i := \#\{j \le i : X_j = X_i\}$. Furthermore, the pairs $(X_i, \#X_i)$ will be referred to as *unified k-mers*.

In *MinHash (MH)*, a random hash is used to sort the k-mers, the k-mer with the lowest index, with respect to hash functions $h_1, \ldots, h_D \sim \mathcal{H}_{\mathcal{A}^k}$

$$\boldsymbol{\varphi}_{mh}(\boldsymbol{x}) := (X_{j_1}, \ldots, X_{j_D}) \qquad j_r := \operatorname*{argmin}_i \{h_r(X_i)\} \tag{4}$$

In *Weighted MinHash (WMH)*, hash functions are drawn from a hash family over unified k-mers $h_1, \ldots, h_D \sim \mathcal{H}_{\mathcal{A}^k \times [N]}$

$$\boldsymbol{\varphi}_{wmh}(\boldsymbol{x}) := (X_{j_1}, \ldots, X_{j_D}) \qquad j_r := \operatorname*{argmin}_i \{h_r(X_i, \#X_i)\} \tag{5}$$

Finally, *Ordered MinHash (OMH)* [9], generalizes WMH by sampling from multiple k-mers that appear in the same order in sequence. The sampling uses hashes over the k-mers space $h_1, \ldots, h_D \sim \mathcal{H}_{\mathcal{A}^k \times [N]}$, and selects indices that are mapped to the t lowest hash values. Formally, for each hash

function h_r, we sketch $(X_{\sigma_1}, \dots, X_{\sigma_t})$, when monotone indices $1 \le \sigma_1 < \dots < \sigma_t \le |X|$ are mapped to the lowest hash values

$$(\boldsymbol{\varphi}_{omh}(\boldsymbol{x}))_r := X_{\sigma_1} \circ \dots \circ X_{\sigma_t} \qquad \left|\left\{i : h_r(X_i, \#X_i) \le h_r(X_{\sigma_j}, \#X_{\sigma_j})\right\}\right| \le t, \text{ for all } j \in [t] \tag{6}$$

We can compute distances via the *Hamming distance* $\mathrm{d_H}(a, b)$, defined as the number of indices that the input sequences differ $\mathrm{d_H}(a, b) = \#\{i \colon a_i \neq b_i\}$, where a and b have equal length. When φ is a MinHash sketch, $\mathrm{d_H}(\varphi(\boldsymbol{x}), \varphi(\boldsymbol{y}))$ is closely related to the Jaccard set similarity index between k-mer sets

$$\frac{1}{D}\mathbb{E}\,\mathrm{d_H}(\boldsymbol{\varphi}_{mh}(\boldsymbol{x}), \boldsymbol{\varphi}_{mh}(\boldsymbol{y})) = 1 - \frac{|S_X \cap S_Y|}{|S_X \cup S_Y|}, \quad S_X := \{X_i\}_{i \le |\boldsymbol{x}|-k+1}, \quad S_Y := \{Y_i\}_{i \le |\boldsymbol{y}|-k+1} \tag{7}$$

Note, that MH, WMH, and OMH are defined as elements with lowest hash over a set of k-mers, unified k-mers, and ordered tuples of k-mers, respectively. For example, in WMH, we can change the definition of S_X to a set of unified k-mers $S_X := \{(X_i, \#X_i) : i \le |\boldsymbol{x}| - k + 1\}$ (analogous for S_Y). It is worth mentioning that OMH becomes equivalent to WMH if we set the tuple length to one $t = 1$.

2.2 Tensor Sketch

First, define $\mathcal{I}_N^t$ to be set of all increasing subsequences of $[N]^t$

$$\mathcal{I}_N^t := \{(\sigma_1, \dots, \sigma_t) \in [N]^t \colon \text{ strictly increasing } \sigma_1 < \dots < \sigma_t\}, \tag{8}$$

Without loss of generality let the alphabet be relabeled to $\mathcal{A} := \{1, \dots, m\}$, and $m := \#\mathcal{A}$ be the size. The main idea behind sequence tensor sketch is counting subsequences as opposed to k-mers. Define $T_{\boldsymbol{x}}^{(t)} \colon \mathcal{A}^t \to [0, 1]$ to be the probability mass function of a random t-ary subsequence of x

$$T_{\boldsymbol{x}}^{(t)}[\boldsymbol{a}] = \mathbb{P}(\boldsymbol{x}_{[\mathbf{i}]} = \boldsymbol{a} \mid \mathbf{i} \sim \mathcal{I}_{|\boldsymbol{x}|}^t). \tag{9}$$

With a slight abuse of notation, we treat $T_{\boldsymbol{x}}^{(t)}$ both as a probability mass function, and an t-dimensional tensor with size m^t. The inner product equals the probability of getting a match, if we draw two t-ary tuples from $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. Therefore, OMH and Tensor Sketch are closely related from a theoretical perspective. Then, for two arbitrary strings $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{A}^\star$, define order t similarity $s^{(t)}(\boldsymbol{x}, \boldsymbol{y})$ and distance $d^{(t)}(\boldsymbol{x}, \boldsymbol{y})$ as

$$s^{(t)}(\boldsymbol{x}, \boldsymbol{y}) = \langle T_{\boldsymbol{x}}^{(t)}, T_{\boldsymbol{y}}^{(t)} \rangle, \qquad d^{(t)}(\boldsymbol{x}, \boldsymbol{y}) := \|T_{\boldsymbol{x}}^{(t)} - T_{\boldsymbol{y}}^{(t)}\|_2^2. \tag{10}$$

However, naive computation of equations (9) and (10) requires $\binom{N}{t}$ time and m^t memory, which becomes prohibitive for even moderate values of these variables. Our tensor sketching scheme, provides a $(1 + \epsilon)$-approximation for tensor distance, but only requires $\mathcal{O}(\frac{1}{\epsilon^2} N t)$ time to compute. Furthermore, any downstream analysis will operate on sketches of size $\frac{1}{\epsilon^2}$, regardless of the original sequence length.

Since we are interested in preserving the Euclidean norm after sketching, we follow the definition of tensor sketch by Pham and Pagh [14]. For integer D, tensor sketch $\Phi : \mathbb{R}^{m^t} \to \mathbb{R}^D$ sketches an m^t-dimensional tensor into $\mathbb{R}^D$. Let us assume a series of pairwise independent hash functions $h_1, \dots, h_t \colon \mathcal{A} \to [D]$, and sign hash functions $s_1, \dots, s_t : \mathcal{A} \to \{-1, 1\}$. Moreover, define hash sum $H \colon \mathcal{A}^t \to [D]$ and hash sign product $S \colon \mathcal{A}^t \to \{-1, 1\}$ as follows

$$H(\boldsymbol{a}) := \sum_i^t h_i(a_i) \bmod D, \qquad S(\boldsymbol{a}) := \prod_i^t s_i(a_i), \qquad \boldsymbol{a} \in \mathcal{A}^t \tag{11}$$

Finally, tensor sketch $\Phi(T) := (\phi_r(T))_{r \in [D]}$ is defined as

$$\phi_r(T) := \sum_{\mathbf{a} \in \mathcal{A}^t : H(\mathbf{a}) = r} S(\mathbf{a}) T[\mathbf{a}], \tag{12}$$

where $T \in \mathbb{R}^{m^t}$ is an arbitrary tensor. Crucially, tensor sketch preserves the Euclidean norm in expectation $\mathbb{E}\|\Phi(T)\|_2^2 = \|T\|_2^2$, and its accuracy increases with sketch size $\operatorname{Var}\|\Phi(T)\|_2^2 \leq \frac{1}{D}\|T\|_2^2$ which leads to the second moment bound (for proof, see Lemma 7 of Pham and Pagh [14]):

$$\mathbb{P}\left(\left|\|\Phi(T)\|_2^2 - \|T\|_2^2\right| \geq \epsilon\|T\|_2^2\right) \leq \frac{2}{\epsilon^2 D}. \tag{13}$$

Therefore, a sketch size of $D = 4\epsilon^{-2}$ suffices to bound the multiplicative error by ϵ with probability $\frac{1}{2}$, and sketch size of $D = \mathcal{O}(\epsilon^{-2}\log(1/\delta))$ suffices to boost the probability of success to $1-\delta$, for arbitrary $\delta \in (0,1)$, by taking the median of $\log(1/\delta)$ independent $\mathcal{O}(\epsilon^{-2})$-size sketches.

If we could compute $\Phi(T_{\boldsymbol{x}}^{(t)})$ and $\Phi(T_{\boldsymbol{y}}^{(t)})$ efficiently, distance computation $\|\Phi(T_{\boldsymbol{x}}^{(t)}) - \Phi(T_{\boldsymbol{y}}^{(t)})\|_2^2$ merely requires $\mathcal{O}(D)$ time, as opposed to the exponential cost of constructing the ambient space. However, since the tensors $T_{\boldsymbol{x}}^{(t)}$ are never rank-1, the conventional efficient tensor sketching schemes cannot be applied here. Therefore, we have designed an efficient algorithm to compute (12), and provide several extensions of the algorithm in the next section. In summary, for sequence $\boldsymbol{x}$, the tensor sketch $\boldsymbol{\varphi}_{ts}(\boldsymbol{x})$ is computed by applying the tensor sketch on tuple tensor $T_{\boldsymbol{x}}^{(t)}$

$$\boldsymbol{\varphi}_{ts}(\boldsymbol{x}) := \Phi(T_{\boldsymbol{x}}^{(t)}). \tag{14}$$

Implicit sketching by recursive computation First, observe that the rth component of the sketch $\phi_r(T_{\boldsymbol{x}}^{(t)})$, can be rewritten based on the probability mass of $(H(\boldsymbol{x}_{[\mathbf{i}]}), S(\boldsymbol{x}_{[\mathbf{i}]}))$, when $\mathbf{i}$ is a uniformly drawn ordered t-tuple $\mathbf{i} \sim \mathcal{I}_N^t$

$$\phi_r(T_{\boldsymbol{x}}^{(t)}) = \sum_{b\in\{1,-1\}} (b) \sum_{\boldsymbol{a}\in\mathcal{A}^t : H(\boldsymbol{a}) \overset{D}{\equiv} r, S(\boldsymbol{a})=b} T_{\boldsymbol{x}}^{(t)}[\boldsymbol{a}] \tag{15}$$

$$= \sum_{b\in\{-1,1\}} (b)\, \mathbb{P}\big(H(x_{[\mathbf{i}]}) = r, S(x_{[\mathbf{i}]}) = b \,\big|\, \mathbf{i} \sim \mathcal{I}_{|\boldsymbol{x}|}^t\big) \tag{16}$$

We set out to recursively compute the probability mass for t-ary tuples, based on probability mass of shorter ordered tuples. Therefore, for $p \in \{1,\dots,t\}$ let $T_{\boldsymbol{x}}^{(p)}(r,b)$ be the probability mass function of $(H_p(\boldsymbol{x}_{[\mathbf{i}]}), S_p(\boldsymbol{x}_{[\mathbf{i}]}))$, when $\mathbf{i} \sim \mathcal{I}_p^N$ is a random ordered p-tuple over $[N]$, while $H_p\colon \mathcal{A}^p \to [D]$ and $S_p\colon \mathcal{A}^p \to \{-1,1\}$ are partial hash sum and products up to p, respectively

$$H_p(\boldsymbol{a}) := \sum_{j=1}^{p} h_j(a_j) \pmod D, \quad S_p(\boldsymbol{a}) := \prod_{j=1}^{p} s_j(a_j), \qquad \boldsymbol{a} \in \mathcal{A}^p \tag{17}$$

$$T_{\boldsymbol{x}}^{(p)}(r,b) := \mathbb{P}\big(H_p(\boldsymbol{x}_{[\mathbf{i}]}) = r, S_p(\boldsymbol{x}_{[\mathbf{i}]}) = b \,\big|\, \mathbf{i} \sim \mathcal{I}_{|\boldsymbol{x}|}^p\big), \qquad b \in \{-1,1\} \tag{18}$$

To recursively compute $T_{\boldsymbol{x}_{[1:N]}}^{(p)}(r,b)$ for $p \geq 2$, we separate it into two cases depending on whether the last index is part of the tuple ($i_p = N$) or not ($i_p < N$)

$$T_{\boldsymbol{x}_{[1:N]}}^{(p)}(r,b) = \mathbb{P}(H_p(\boldsymbol{x}_{[\mathbf{i}]}) = r, S_p(\boldsymbol{x}_{[\mathbf{i}]}) = b \mid \mathbf{i} \sim \mathcal{I}_N^p) \tag{19}$$

$$= \overbrace{\mathbb{P}_{\mathbf{i}}(i_p = N)}^{z:=} \mathbb{P}(H_p(\boldsymbol{x}_{[\mathbf{i}]}) = r, S_p(\boldsymbol{x}_{[\mathbf{i}]}) = b \mid \mathbf{i} \sim \mathcal{I}_N^p, i_p = N) + \mathbb{P}_{\mathbf{i}}(i_p < N)\mathbb{P}(H_p(\boldsymbol{x}_{[\mathbf{i}]}) = r, S_p(\boldsymbol{x}_{[\mathbf{i}]}) = b \mid \mathbf{i} \sim \mathcal{I}_N^p, i_p < N) \tag{20}$$

$$= z\, \mathbb{P}\Big(\sum_{j=1}^{p} h_j(\boldsymbol{x}_{\mathbf{i}_j}) + \overbrace{h_p(\boldsymbol{x}_N)}^{r':=} = r, \prod_{j=1}^{p-1} s_j(\boldsymbol{x}_{\mathbf{i}_j}) \overbrace{s_p(\boldsymbol{x}_N)}^{b':=} = b \,\Big|\, \mathbf{i} \sim \mathcal{I}_{N-1}^{p-1}\Big) + (1-z)\mathbb{P}\Big(H_p(\boldsymbol{x}_{[\mathbf{i}]}) = r, S_p(\boldsymbol{x}_{[\mathbf{i}]}) = b \,\Big|\, \mathbf{i} \sim \mathcal{I}_{N-1}^{p-1}\Big) \tag{21}$$

$$= z\, \mathbb{P}\Big(H_{p-1}(\boldsymbol{x}_{[\mathbf{i}]}) \overset{D}{\equiv} r - r', S_{p-1}(\boldsymbol{x}_{[\mathbf{i}]}) = bb' \,\Big|\, \mathbf{i} \sim \mathcal{I}_{N-1}^{p-1}\Big) + (1-z)\mathbb{P}\Big(H_p(\boldsymbol{x}_{[\mathbf{i}]}) = r, S_p(\boldsymbol{x}_{[\mathbf{i}]}) = b \,\Big|\, \mathbf{i} \sim \mathcal{I}_{N-1}^{p}\Big) \tag{22}$$

$$= z\, T_{\boldsymbol{x}_{[1:N-1]}}^{(p-1)}(r - r' \pmod D, bb') + (1-z) T_{\boldsymbol{x}_{[1:N-1]}}^{(p)}(r,b), \tag{23}$$

where $z = \frac{p}{N}$, since $\binom{N-1}{p-1}/\binom{N}{p} = \frac{p}{N}$. Defining vector $\mathbf{T}_{\boldsymbol{x}}^{(p)}(b) := (T_{\boldsymbol{x}}^{(p)}(r,b))_{r\in[D]}$, the index shift $r - r' \pmod D$, amounts to a circular r'-shift of $\mathbf{T}_{\boldsymbol{x}_{[1:N-1]}}^{(p)}(b)$. Hence, the recursion in vector form can be written as

◊ *Recursive insert* ($p \in [t]$):

$$\begin{cases} \mathbf{T}_{\boldsymbol{x}_{[1:N]}}^{(p)}(\pm 1) = (1-z)\mathbf{T}_{\boldsymbol{x}_{[1:N-1]}}^{(p)}(\pm 1) + z \text{ shift}_{r'}\left(\mathbf{T}_{\boldsymbol{x}_{[1:N-1]}}^{(p-1)}(\pm b')\right) \\ r' := h_p(\boldsymbol{x}_N), \quad b' := s_p(\boldsymbol{x}_N), \quad z := \frac{p}{N} \end{cases} \tag{24}$$

◊ *Basis* ($p = 0$):

$$\mathbf{T}_{\boldsymbol{x}}^{(0)}(\pm 1) := \left(\mathbb{1}\left\{r \overset{D}{\equiv} 0\right\}\right)_{r\in[D]} \tag{25}$$

Interestingly, the recursion relation for variables in $T_{\boldsymbol{x}_{[1:N]}}^{(p)}(\cdot,\cdot)$ depends only on variables in the previous layer $T_{\boldsymbol{x}_{[1:N-1]}}^{(p)}(\cdot,\cdot)$. Therefore, it is sufficient to store the variables corresponding to layer i, which implies a memory complexity of $\mathcal{O}(tD)$. This fact is exploited in Algorithm 1, which is why the subscript $\boldsymbol{x}_{[\mathbf{i}]}$ is dropped from $\mathbf{T}$, and uses ± 1 for brevity[4]. The number of sub-problems needed to compute $\Phi(T_{\boldsymbol{x}}^{(t)})$ is exactly $2NtD$, and the cost of each recursion is $\mathcal{O}(1)$. Therefore, we can compute the sketch in $\mathcal{O}(DtN)$ time.

Algorithm 1: Sequence Tensor Sketching

Output : Sequence Tensor Sketch $\Phi \in \mathbb{R}^D$
Input : string $\boldsymbol{x} \in \mathcal{A}^N$,
hash functions $h_1, \ldots, h_t : \mathcal{A} \to [D]$,
sign hashes $s_1, \ldots, s_t : \mathcal{A} \to \{-1, 1\}$
Parameter: D: sketch size, t: subsequence length
$\mathbf{T}_{\pm 1}^{(p)} \leftarrow \mathbf{0}^D$ for all $p \in [t]$
$\mathbf{T}_{\pm 1}^{(0)} \leftarrow \left(\mathbb{1}\left\{r \overset{D}{\equiv} 0\right\}\right)_{r\in[D]}$
for $i \leftarrow 1$ *to* N **do**
  for $p \leftarrow t$ *to* 1 **do**
    $z \leftarrow \frac{p}{i}$
    $\mathbf{T}_{\pm 1}^{(p)} \leftarrow (1-z)\mathbf{T}_{\pm 1}^{(p)} + z \text{ shift}_{h_p(x_i)}\left(\mathbf{T}_{\pm s_p(x_i)}^{(p-1)}\right)$
  end
end
$\Phi \leftarrow \mathbf{T}_{+1}^{(t)} - \mathbf{T}_{-1}^{(t)}$

2.3 Tensor Slide Sketch

In this section, we improve precision of tensor sketch, by concatenating sketches of overlapping subsequences of the original sequence. However, instead of applying Algorithm 1 to compute each sketch separately, we can exploit the fact that many sub-problems in the recursion (19) are shared. The main idea is to consider sub-problems $T_{\boldsymbol{x}}^{(p:q)}(\cdot,\cdot)$, that correspond to hash functions $s_p, \ldots, s_q$ and $h_p, \ldots, h_q$, for all $1 \le p \le q \le t$, so that we can efficiently remove characters from the initial position. Formally, $T_{\boldsymbol{x}}^{(p:q)}(\cdot,\cdot)$ is the probability mass of $\left(\sum_{j=p}^{q} h_j(\boldsymbol{x}_{i_j}) (\bmod m), \prod_{j=p}^{q} s_j(\boldsymbol{x}_{i_j})\right)$,

[4] The plus-minus signs are shorthand for two equations: one by taking all to be $+$, and another taking all to be $-$, but not any other combination, for example $\sin(a \pm b) = \sin(a)\cos(b) \pm \sin(b)\cos(a)$. If there are $\mp$ in the equation, their order for execution must the reversed, for example $\cos(a \pm b) = \cos(a)\cos(b) \mp \sin(a)\sin(b)$.

when $\mathbf{i}$ is a randomly drawn $(q-p+1)$-ary increasing subsequence $\mathbf{i} \sim \mathcal{I}_{|\boldsymbol{x}|}^{q-p+1}$ (See **Supplementary Section A** for more details).

In Algorithm 2, $\mathbf{T}_b^{(p:q)}$ denotes the probability vector $\mathbf{T}_b^{(p:q)} := (T_{\boldsymbol{x}}^{(p:q)}(r,b))_{r\in D}$. The time and memory complexity of computing sketches for all w-mers of a reference sequence with length N are $\mathcal{O}(NDt^2)$ and $\mathcal{O}(t)$, respectively. The length of the sliding window w does not appear in any complexity term. In our experience, a value of $t \in \{2,3,4,5\}$ suffices for most practical purposes, which could lead to significant savings, when window length is larger $w \gg 1$.

The tensor slide sketch, is defined as

$$\boldsymbol{\varphi}_{tss}(\boldsymbol{x}) := (\boldsymbol{\varphi}_{ts}(\boldsymbol{x}_{[iw+1:(i+1)w]}))_{i\in\{0,1,\ldots,\lfloor N/w\rfloor\}}. \tag{26}$$

where $\boldsymbol{\varphi}_{ts}(\boldsymbol{x})$ is Tensor Sketch(14), the only difference being that successive sketches are computed more efficiently. It is important to note that Tensor Slide Sketch (TSS) only stores $\frac{1}{w}$ of the indices. Because sliding windows that have a large overlap cannot improve our estimate, we can down-sample the slide sketch to keep the sketch condensed.

Algorithm 2: Tensor Slide Sketch

Output : Sketches $\Phi_1 := \Phi(T^{(t)}_{\boldsymbol{x}[\boldsymbol{1:w}]}), \ldots, \Phi_{N-w+1} := \Phi(T^{(t)}_{\boldsymbol{x}[\boldsymbol{N-w+1:N+}]})$
Input : reference sequence $\boldsymbol{x} \in \mathcal{A}^N$,
hash functions $h_1, \ldots, h_t : \mathcal{A} \to [D]$,
sign hashes $s_1, \ldots, s_t : \mathcal{A} \to \{-1, 1\}$
Parameter: D: sketch size,
w: sliding window,
t: tuple length

$\mathbf{T}_{\pm 1}^{(p:q)} \leftarrow \mathbf{0}^D$ for all $1 \le p \le q \le t$
$\mathbf{T}_{\pm 1}^{(p:q)} \leftarrow \left(\mathbb{1}\left\{r \overset{D}{\equiv} 0\right\}\right)_{r\in[D]}$ for all $1 \le q < p \le t$
for $i \leftarrow 1$ *to* N **do**
  $x_{ins} \leftarrow x_i$
  for *all* $1 \le p \le q \le t$ *in descending* $(q-p)$ **do**
    $z \leftarrow \frac{q-p+1}{w}$
    $\mathbf{T}_{\pm 1}^{(p:q)} \leftarrow (1-z)\mathbf{T}_{\pm 1}^{(p:q)} + z \operatorname{shift}_{h_q(x_{ins})}\left(\mathbf{T}^{(p:q-1)}_{\pm s_q(x_{ins})}\right)$
  end
  if $i \le w$ **then**
    `Continue`;
  end
  $x_{del} \leftarrow x_{i-w}$
  for $1 \le p \le q \le t$ *in ascending* $(q-p)$ **do**
    $z \leftarrow \frac{q-p+1}{w-q+p}$
    $\mathbf{T}_{\pm 1}^{(p:q)} \leftarrow (1+z)\mathbf{T}_{\pm 1}^{(p:q)} - z \operatorname{shift}_{h_q(x_{del})}\left(\mathbf{T}^{(p:q-1)}_{\pm s_q(x_{del})}\right)$
  end
  $\Phi_{(i-w+1)} \leftarrow \mathbf{T}_{+1}^{(1:t)} - \mathbf{T}_{-1}^{(1:t)}$
end

3 Experimental Results

The primary objective of our tensor sketching framework is to efficiently sketch global alignment scores between sequences. In order to put our method into context, we compare its performance to MH, WMH, and OMH [9], three state-of-the-art sketching techniques (see **Section 2** for formal definitions).

3.1 Synthetic data generation

As a basis for evaluation, we generated a set of synthetic sequence pairs over the full spectrum of edit distances using Monte Carlo sampling. We draw a reference sequence at random $\boldsymbol{x} \sim \mathcal{A}^N$, and for the desired edit distance $d \in [N]$, we generate $\boldsymbol{y}$ by mutating with d edit operations at random $\boldsymbol{x} =: \boldsymbol{x}_0 \to \boldsymbol{x}_1 \to \cdots \to x_d =: \boldsymbol{y}$, where $\boldsymbol{x}_{i+1}$ is created by selecting a random index from $[|\boldsymbol{x}_i|]$, and either delete, substitute, or insert a random character at that index. We accept the pair $\boldsymbol{x}, \boldsymbol{y}$, if we have $\mathrm{d_{ed}}(\boldsymbol{x}, \boldsymbol{y}) = d$, and compute the sketches and sketch-based distances for each method. This process is repeated to create the full data set. In total, we generated 1000 sequence pairs of length $N = 20000$ over an alphabet of size $\#\mathcal{A} = 4$. (See **Supplemental Figure S1** for a raw presentation of approximate vs. exact distances as scatter plot).

3.2 Tensor slide sketch achieves high rank correlation with edit distance

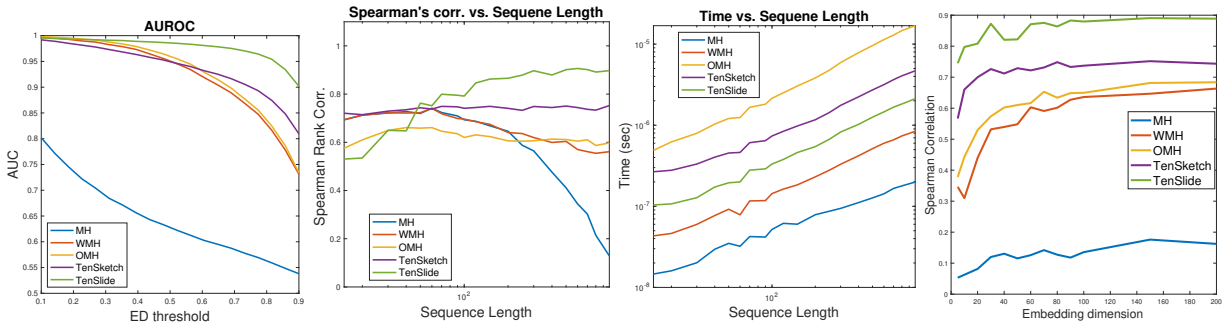
For each sketching method, Spearman's rank correlation with the edit distance, the execution time, and the execution time relative to the edit distance is determined (see Table 1). In particular, Tensor Slide Sketch achieves a correlation of 0.881 while reducing the computation time by 98.6%. The 0 rank correlation for MinHash indicates the fact that very long sequences have 0 distance, as they contain every possible k-mer. Furthermore, the AUC are calculated for classification of sequences based on a threshold for their exact edit distance (normalized by length). For all thresholds, TSS clearly outperforms other sketches.

Table 1: 1000 sequence pairs of length $N = 20000$ were generated over an alphabet of size $\#\mathcal{A} = 4$. with the number of random edit operations, uniformly drawn from $\{0, 1, \ldots, N\}$. The time column shows normalized time in microseconds, i.e., total time divided by number of sequences, while the relative time shows the ratio of sketch-based time to the time for computing exact edit distance. As for the the model parameters, embedding dimension is set to $D = 50$, and model parameters are (a) MinHash $k = 3$, (b) Weighted MinHash $k = 3$, (c) Ordered MinHash $k = 3, t = 3$, (d) Tensor Sketch $t = 3$, (e) Tensor Slide Sketch $w = 32, t = 3$.

	Correlation	**AUROC** ($\mathrm{d_{ed}} \leq \cdot$)			**Time**	
Method	Spearman	0.10	0.20	0.5	Abs. (10^{-6} sec)	Rel.(1/ED)
MH	0	0.500	0.500	0.500	**1.398**	0.001
WMH	0.361	0.795	0.746	0.670	6.497	0.004
OMH	0.466	0.917	0.817	0.700	194.195	0.120
TS	0.687	0.968	0.890	0.837	45.591	0.028
TSS	**0.881**	**0.980**	**0.965**	**0.939**	22.198	0.014
ED	-	-	-	-	1614.040	1.000

We then assessed the performance and execution time of all considered methods utilizing the synthetically generated data set (Figure 1). We observe that the quality of both MinHash (MH) and Weighted MinHash (WMH) greatly diminishes as the sequences become more distant from one another (**Figure 1a**) and the quality of MH and WMH drops with increasing sequence length (**Figure 1b**). In both plots, the accuracy of OMH and the tensor sketch methods maintains a much higher level. Notably, the increased accuracy of TS and TSS comes only at a very small additional cost in time when compared with MH (**Figure 1c**), while both methods run much faster than OMH. Lastly, we assessed the relationship of sketching accuracy to the number of embedding dimensions for each method (**Figure 1d**). Remarkably, TSS achieves a rank correlation of 0.74% even when using only 10 embedding dimensions.

In summary, TSS produces the most accurate sketches across the board, while introducing a small time and memory overhead when compared with min-hashing. Furthermore, TS and TSS can be computed in a streaming fashion. For example, in order to sketch all 100-mers of a reference sequence, the time complexity grows only linearly with the length of the reference sequence, while



(a) (b) (c) (d)

Fig. 1: The dataset for these experiments consisted of 10000 sequence pairs independently generated over an alphabet of size $\#\mathcal{A} = 4$, and models parameters: MH: $k = 3$, WMH: $k = 3$, OMH: $k = 3, t = 3$, TS: $t = 3$, TSS: $w = 32, t = 3$ (1a) Area Under the ROC Curve (AUC) distribution, for detection of distance pairs below a threshold sketch-based distances. The x-axis, shows which edit distance rank (normalized) is used, and the y axis shows AUC for various sketch-based distances. (1b) For the same dataset as in (1c), the Spearman's rank correlation is plotted against the sequence length (logarithmic scale), (1c) Execution time of each sketching method (y-axis, log-scale) as a function of sequence length one (x-axis, log scale). The reported times are normalized, i.e., total sketching time divided by the number of sequences. (1d) Accuracy of various sketching methods as a function of number of embedding dimensions D. All methods reach a plateau at some point, after which quality of estimation does improve with sketch size.

WMH and MH need to be computed separately for each sequence of length m, rendering them m times slower for this task.

While sorting creates an iterative bottleneck for all hash-based schemes, the tensor sketch operation is based on a vector shift and sum, which explains the faster execution in practice. Although we have only reported times for single-thread execution, we can expect tensor sketch to achieve a greater boost from multi-threading, as vector shift and sum can be easily vectorized for parallel execution. Moreover, vector operations can be executed in a single step on some CPU or GPU architectures to achieve an even greater boost. Finally, the hash-based methods store k-mers, or tuples of k-mers as the sketch, which grows with alphabet size and model parameters. In contrast, the tensor sketches can be stored as small integers, namely in a single byte.

3.3 Tensor Sketching helps estimate phylogenies

We further explored the task of estimating all pairwise distances of a given set of evolutionary related sequences, resulting in a well structured distance matrix. This task is reminiscent of important bioinformatics applications, such as the binning of metagenomic reads or phylogeny reconstruction. In both cases, the reconstruction of the good distance matrix as a whole, and not only individual components, forms the basis for a good approximation. While there are several ways to formulate this problem, we can simply visualize the exact and approximate matrices and compare them with an overall metric such as Spearman's rank correlation. Figure 2, shows such a distance matrix. The sequences were generated to emulate a phylogeny tree.

3.4 Tensor Sketching supports Discretization

While the result of sketching is a rational number, in our applications, these numbers were discretized to take less space. In fact, one can store the sign bit, plus its logarithm with $(1 + \epsilon)$ base rounded to the nearest integer i, to ensure an ϵ-bound on the error, as the original number must lie in the interval between $(1 + \epsilon)^i$ and $(1 + \epsilon)^{i+1}$. We used 256 bins, which implies one byte per dimension. In contrast, MH and WMH sketch k-mers for each embedding dimension, which will

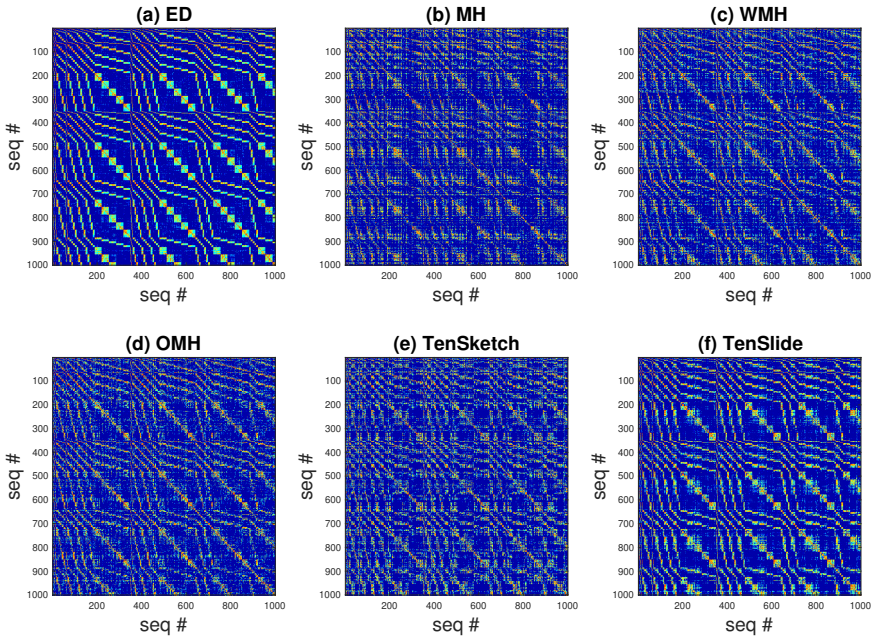


Fig. 2: 1000 sequences of length $n = 100$ were generated over an alphabet of size 4. The phylogeny constructed as a perfectly balanced binary tree, by starting from a single sequence, and at each generation mutate every sequence in the pool and add it to the pool. Children were generated with a point-mutation rate of 0.1. Subplot (a) shows the pairwise edit distances normalized by length n, while subplots (b)-(f) are approximate sketch-based distances normalized by embedding dimension $m = 128$

take $k \log(\#\mathcal{A})$ bits per dimension. As OMH sketches t separate k-mers, this complexity will even increase to $tk \log(\#\mathcal{A})$ bits.

3.5 Tensor Sketch opens up a wide range of applications

Sketching de-Bruijn graph distances The slide-sketch algorithm presented in 2, assumes only having access to a stream of characters coming in and that the task involves sketching substrings of the entire sequence. While the task description may sound to be restricted to a linear reference sequence, the same assumptions actually apply to any Euler path on a string graph. For example, if the input is a de-Bruijn graph and the task is to sketch all vertices of the graph, any traversal of the graph that walks through adjacent vertices can be mapped to the streaming of incoming and outgoing characters, which will fit the assumptions of the slide sketch algorithm. Consequently, the time complexity of sketching vertices of a de-Bruijn graph will grow only as a function of its size, i.e., the number of its vertices, as opposed to the size of each vertex.

Robustness to Global Mutations Tensor Sketch is sensitive to global shifts and transpositions of characters, while k-mer statistics are entirely local. To illustrate this, consider case when sequence $\boldsymbol{y}$ is constructed from $\boldsymbol{x}$, by swapping the first and second half of the sequence $\boldsymbol{y} := \boldsymbol{x}[1 : \frac{N}{2}] \circ \boldsymbol{x}[\frac{N}{2} + 1 : N]$. It is evident that $(N - k + 1)$ out of N k-mers are intact under this block permutation, implying that for sufficiently large $N \gg k$ the k-mer profile of $\boldsymbol{y}$ will converge to $\boldsymbol{x}$. This may severely overestimate their alignment score. In contrast, only $2\binom{N/2}{t}/\binom{N}{t} \simeq 2^{-(t-1)}$ of t-ary subsequences remain intact. This greater sensitivity makes tensor sketch an appealing method for applications that such block permutations are likely to occur.

4 Discussion

We presented tensor sketching, a method for estimating sequence similarity for biological sequences. We demonstrated that Tensor Slide Sketch achieves a high Spearman's rank correlation, but runs an order of magnitude faster than computing the exact alignment. When compared with other sketching methods, Tensor Sketch and Tensor Slide Sketch both achieve a more accurate preservation of the order between distances than MinHash, Weighted MinHash, and Ordered MinHash.

Hyper-parameters The results presented in Section 3 have focused on a fixed set of parameters for the models. However, our results are fairly robust to the choice of parameters. Namely, varying the size of k-mers will be beneficial to hash-based sketches for small edit distances, while it will hurt the performance for more distant pairs. Hence, Spearman's rank correlation will not substantially improve. While tensor sketch achieves the best accuracy in our set of experiments, a comprehensive study of different parameter regimes and settings under which they will be beneficial merits further investigation. In particular, min-hash methods typically show good performance at detecting small distances, as it is evident in our AUROC plot in Figure 1. This suggests the possibility of having a hybrid-version of tensor sketching and ordered MinHash to combine the best features of the two. Moreover, such a study could shed more light on the connections between min-hashing and tensor sketching.

Bayesian estimation with tensor sketching Spaced k-mers were introduced motivated by applications involving distantly related sequences. If we are to allow for i *ignore* positions, there will be $\binom{k}{i} \approx (k/i)^i$ possible patterns. Considering all such patterns is clearly not an option if i is anything larger but a small constant. Not only is it highly non-trivial to search for the optimal pattern in this large combinatorial space, but any incremental step for pattern optimization also has to be repeated if different tasks require a different set of patterns. Seen from this angle, Tensor Sketch can be interpreted as an average of all possible spaced t-mers patterns with $(N-t)$ ignore positions, while the sketching step avoids the exponential cost of explicitly representing all combinatorial options.

The core idea of tensor sketching, that is to average over all possible seed patterns, can be alternatively viewed as Bayesian estimation. This implies that our estimate is admissible, i.e., no other estimator can outperform it across the board. This provides some intuition into why our approach achieves an acceptable or better accuracy over the entire range of edit distances. In other words, the risk of selecting a bad seed or size of k-mer is minimized by taking the mean over all possible seeds. While this corresponds to a uniform prior on our seeds, one can introduce a non-uniform prior by setting different weights for vertices on our tensors, namely to penalize more number of gaps in the pattern. Therefore, we can strengthen or weaken the contribution of individual patterns. Weighted average can help to reduce background statistics into the distance metric, as well as introduce nonuniform mismatch penalties. These variations and extensions come naturally within the tensor sketching framework, in contrast with hash-based approaches that require heuristic steps.

Conclusion This work's main contributions are the introduction of two tensor-based sketching methods to sequence similarity estimation: Tensor Sketch, providing an efficient algorithm to compute it, and Tensor Slide Sketch, a streaming version of the algorithm. Our results indicate that the tensor sketching method and its applications open up exciting research directions to explore for the bioinformatics community. The main advantages of tensor sketching compared with hash-based methods are that it 1) can run in a streaming fashion, 2) achieves much higher accuracy, and 3) runs much faster in practice.

Acknowledgements

We would like to thank Ximena Bonilla, Daniel Danciu, and Kjong-Van Lehmann for their constructive feedback on the manuscript. AJ was funded by ETH Zurich core funding to GR. AK was partially funded by PHRT Project #106 and the Swiss National Science Foundation Grant No. 407540_167331 "Scalable Genome Graph Data Structures for Metagenomics and Genome Annotation" as part of Swiss National Research Programme (NRP) 75 "Big Data".

References

1. Apostolico, A., Denas, O.: Efficient tools for comparative substring analysis. Journal of Biotechnology **149**(3), 120–126 (2010). DOI 10.1016/J.JBIOTEC.2010.05.006. URL https://www.sciencedirect.com/science/article/pii/S0168165610002403?via%3Dihub
2. Broder, A.Z.: On the resemblance and containment of documents. Proceedings of the International Conference on Compression and Complexity of Sequences pp. 21–29 (1997)
3. Hahn, L., Leimeister, C.A., Ounit, R., Lonardi, S., Morgenstern, B.: rasbhari: Optimizing Spaced Seeds for Database Searching, Read Mapping and Alignment-Free Sequence Comparison. PLoS computational biology **12**(10), e1005,107 (2016). DOI 10.1371/journal.pcbi.1005107. URL http://www.ncbi.nlm.nih.gov/pubmed/27760124http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC5070788
4. Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing for scalable image search. In: 2009 IEEE 12th international conference on computer vision, pp. 2130–2137. IEEE (2009)
5. Leimeister, C.A., Boden, M., Horwege, S., Lindner, S., Morgenstern, B.: Fast alignment-free sequence comparison using spaced-word frequencies. Bioinformatics (Oxford, England) **30**(14), 1991–9 (2014). DOI 10.1093/bioinformatics/btu177. URL http://www.ncbi.nlm.nih.gov/pubmed/24700317http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4080745
6. Leimeister, C.A., Morgenstern, B.: Kmacs: the k-mismatch average common substring approach to alignment-free sequence comparison. Bioinformatics (Oxford, England) **30**(14), 2000–8 (2014). DOI 10.1093/bioinformatics/btu331. URL http://www.ncbi.nlm.nih.gov/pubmed/24828656http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4080746
7. Ma, B., Tromp, J., Li, M.: PatternHunter: faster and more sensitive homology search. Bioinformatics **18**(3), 440–445 (2002). DOI 10.1093/bioinformatics/18.3.440. URL http://www.ncbi.nlm.nih.gov/pubmed/11934743https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/18.3.440
8. Mak, D.Y., Benson, G.: All hits all the time: parameter-free calculation of spaced seed sensitivity. Bioinformatics **25**(3), 302–308 (2009). DOI 10.1093/bioinformatics/btn643. URL https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btn643
9. Marcais, G., Deblasio, D., Pandey, P., Kingsford, C.: Locality-sensitive hashing for the edit distance. Bioinformatics **35**(14), i127–i135 (2019). DOI 10.1093/bioinformatics/btz354
10. Noé, L.: Best hits of 11110110111: model-free selection and parameter-free sensitivity calculation of spaced seeds. Algorithms for molecular biology : AMB **12**, 1 (2017). DOI 10.1186/s13015-017-0092-1. URL http://www.ncbi.nlm.nih.gov/pubmed/28289437http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC5310094
11. Noe, L., Martin, D.E.: A coverage criterion for spaced seeds and its applications to support vector machine string kernels and k-mer distances. Journal of Computational Biology **21**(12), 947–963 (2014). DOI 10.1089/cmb.2014.0173. URL http://www.ncbi.nlm.nih.gov/pubmed/25393923http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4253314
12. Ondov, B.D., Treangen, T.J., Melsted, P., Mallonee, A.B., Bergman, N.H., Koren, S., Phillippy, A.M.: Mash: Fast genome and metagenome distance estimation using MinHash. Genome Biology **17**(1), 1–14 (2016). DOI 10.1186/s13059-016-0997-x. URL http://dx.doi.org/10.1186/s13059-016-0997-x
13. Onodera, T., Shibuya, T.: The Gapped Spectrum Kernel for Support Vector Machines. In: International Workshop on Machine Learning and Data Mining in Pattern Recognition, pp. 1–15. Springer, Berlin, Heidelberg (2013). DOI 10.1007/978-3-642-39712-7{_}1. URL http://link.springer.com/10.1007/978-3-642-39712-7_1
14. Pham, N., Pagh, R.: Fast and scalable polynomial kernels via explicit feature maps. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 239–247 (2013)
15. Röhling, S., Dencker, T., Morgenstern, B.: The number of k-mer matches between two DNA sequences as a function of k. bioRxiv p. 527515 (2019). DOI 10.1101/527515. URL https://www.biorxiv.org/content/10.1101/527515v2
16. Sims, G.E., Jun, S.R., Wu, G.A., Kim, S.H.: Whole-genome phylogeny of mammals: Evolutionary information in genic and nongenic regions. Proceedings of the National Academy of Sciences of the United States of America **106**(40), 17,077–17,082 (2009). DOI 10.1073/pnas.0909377106. URL http://www.ncbi.nlm.nih.gov/pubmed/19805074http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC2761373

17. Sims, G.E., Kim, S.H.: Whole-genome phylogeny of Escherichia coli/Shigella group by feature frequency profiles (FFPs). Proceedings of the National Academy of Sciences of the United States of America **108**(20), 8329–34 (2011). DOI 10.1073/pnas.1105168108. URL `http://www.ncbi.nlm.nih.gov/pubmed/21536867http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC3100984`
18. Ulitsky, I., Burstein, D., Tuller, T., Chor, B.: The Average Common Substring Approach to Phylogenomic Reconstruction. Journal of Computational Biology **13**(2), 336–350 (2006). DOI 10.1089/cmb.2006.13.336. URL `http://www.liebertpub.com/doi/10.1089/cmb.2006.13.336`

Appendix A Tensor Slide Sketch Algorithm

For all $1 \le p \le q \le t$, let $H_{p:q}\colon \mathcal{A}^{q-p+1} \to [D]$ and $S_{p:q}\colon \mathcal{A}^{q-p+1} \to \{-1,1\}$ be partial hash sum and products from p up to q, respectively, and define $T_{\boldsymbol{x}}^{(p:q)}(r,b)$ be the probability mass function of $(H_{p:q}(\boldsymbol{x}_{[\mathbf{i}]}), S_{p:q}(\boldsymbol{x}_{[\mathbf{i}]}))$, when $\mathbf{i}$ is a uniformly drawn ordered $(q-p+1)$-tuple.

$$H_{p:q}(\boldsymbol{a}) := \sum_{j=p}^{q} h_j(a_j) \pmod D, \quad S_{p:q}(\boldsymbol{a}) := \prod_{j=p}^{q} s_j(a_j), \qquad \boldsymbol{a} \in \mathcal{A}^{q-p+1} \tag{27}$$

$$T_{\boldsymbol{x}}^{(p:q)}(r,b) := \mathbb{P}\big(H_{p:q}(\boldsymbol{x}_{[\mathbf{i}]}) = r, S_{p:q}(\boldsymbol{x}_{[\mathbf{i}]}) = b \,\big|\, \mathbf{i} \sim \mathcal{I}_{|\boldsymbol{x}|}^{q-p+1}\big), \qquad b \in \{-1,1\} \tag{28}$$

The recursive insertion is analogous to (19), as we can derive $T_{\boldsymbol{x}_{[1:w]}}^{(p:q)}(r,b)$ based on the previous layer $T_{\boldsymbol{x}_{[1:w-1]}}^{(p:q)}(\cdot,\cdot)$ and smaller problems $T_{\boldsymbol{x}_{[1:w-1]}}^{(p:q-1)}(\cdot,\cdot)$. Recursive deletion is equivalent to rolling back insertion of the first character. We can derive $T_{\boldsymbol{x}_{[1:w+1]}}^{(p:q)}(r,b)$ based on $T_{\boldsymbol{x}_{[2:w+1]}}^{(p+1:q)}(r',b')$ and $T_{\boldsymbol{x}_{[2:w+1]}}^{(p:q)}(r,b)$, by separating it into two cases when the random tuple starts at the first index being in the tuple

$$T_{\boldsymbol{x}_{[1:w+1]}}^{(p:q)}(r,b) = (1-t)T_{\boldsymbol{x}_{[2:w]}}^{(p:q)}(r,b) + t\, T_{\boldsymbol{x}_{[2:w]}}^{(p+1:q)}(r - r' \pmod D, b'), \quad t := \mathbb{P}(\mathbf{i}_1 = 1 \mid \mathbf{i} \sim \mathcal{I}_{w+1}^{q-p+1}) \tag{29}$$

where $t = \frac{p-q+1}{w+1}$, analogous to the insertion only case, and $r' := r - h_p(\boldsymbol{x}_1)$ and $b' := bs_p(\boldsymbol{x}_1)$ are recursive hashes as before. We can rearrange this identity to compute $T_{\boldsymbol{x}_{[2:w]}}^{(p:q)}(r,b)$ based on other terms

$$T_{\boldsymbol{x}_{[2:w+1]}}^{(p:q)}(r,b) = \frac{1}{1-t} T_{\boldsymbol{x}_{[1:w+1]}}^{(p:q)}(r,b) - \frac{t}{1-t}\, T_{\boldsymbol{x}_{[2:w+1]}}^{(p+1:q)}(r - r' \pmod D, bb'), \tag{30}$$

Defining probability vector $\mathbf{T}_{\boldsymbol{x}}^{(p:q)}(b) := (T_{\boldsymbol{x}}^{(p:q)}(r,b))_{r\in[D]}$, and calculating $\frac{t}{1-t} = \frac{q-p+1}{w-q+p}$, we can write the recursion in the vector form more concisely as:

◊ *Recursive delete ($p \le q$):*

$$\begin{cases} \mathbf{T}_{\boldsymbol{x}_{[2:w+1]}}^{(p:q)}(\pm 1) = (1+z)\mathbf{T}_{\boldsymbol{x}_{[1:w]}}^{(p:q)}(\pm 1) - z\, \mathrm{shift}_{r'}\left(\mathbf{T}_{\boldsymbol{x}_{[2:w]}}^{(p+1:q)}(\pm b')\right), \\ b' := s_p(\boldsymbol{x}_1), \quad r' := h_p(\boldsymbol{x}_1), \quad z := \frac{q-p+1}{w} \end{cases} \tag{31}$$

◊ *Recursive insert ($p \le q$):*

$$\begin{cases} \mathbf{T}_{\boldsymbol{x}_{[1:w]}}^{(p:q)}(\pm 1) = (1-z)\mathbf{T}_{\boldsymbol{x}_{[1:w-1]}}^{(p:q)}(\pm 1) + z\, \mathrm{shift}_{r'}\left(\mathbf{T}_{\boldsymbol{x}_{[1:w]}}^{(p:q-1)}(\pm b')\right), \\ b' := s_q(x_N), \quad r' := h_q(x_N), \quad z := \frac{q-p+1}{w-q+p} \end{cases} \tag{32}$$

◊ *Basis ($q < p$):*

$$\mathbf{T}_{\boldsymbol{x}}^{(p:q)}(b) := \left(\mathbb{1}\left\{r \overset{D}{\equiv} 0\right\}\right)_{r\in[D]} \tag{33}$$

In order to implement an iterative algorithm, one can roll back the recursive calls and execute them in the reverse order. Namely for deletion, recursive relation for $\mathbf{T}_{\boldsymbol{x}_{[2:w+1]}}^{(p:q)}(\cdot)$ depends on $\mathbf{T}_{\boldsymbol{x}_{[1:w+1]}}^{(p:q)}(\cdot)$, and $\mathbf{T}_{\boldsymbol{x}_{[2:w]}}^{(p+1:q)}(\cdot)$, implying that updates for $\mathbf{T}_{\boldsymbol{x}_{[2:w+1]}}^{(p+1:q)}(\cdot)$ must precede updates for $\mathbf{T}_{\boldsymbol{x}_{[2:w+1]}}^{(p:q)}(\cdot)$, imposing a partial order corresponding to tuple length $(q-p+1)$. This in turn corresponds with the random subsequence length. For insertion recursion (19), sub-problems correspond to shorter subsequence lengths, while for deletion recursion (32), sub-problems correspond to longer subsequence lengths, which impose an ascending and descending dependency on subsequence length, respectively.

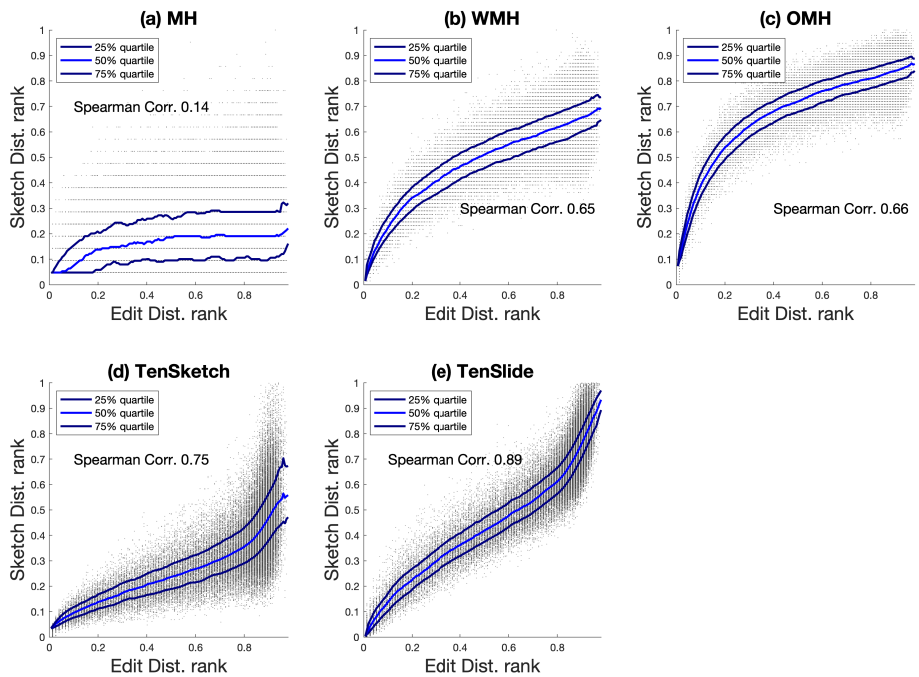


Fig. S1: Normalized sketch distance ranks versus normalized edit distance ranks. 50000 sequence pairs of length $N = 256$ were generated over $\#\mathcal{A} = 4$ alphabets. One sequence was generated randomly, and the second was mutated randomly, to generate a spectrum of edit distances. Subplot (a-e) show rank of sketch-based distances vs. rank of edit distances, both normalized by the total number of pairs. The embedding dimension for all models, and parameters for each model are $D = 100$, and parameters models are (a) MinHash $k = 3$, (b) Weighted MinHash $k = 3$, (c) Ordered MinHash $k = 3, t = 3$, (d) Tensor Sketch $t = 3$, (e) Tensor Slide Sketch $w = 32, t = 3$.

Appendix B More results

Figure S1 shows individual pair of sequences as a scatter plot, while the solid lines indicate median and quartiles. As the Spearman's rank correlation indicates, that rank of edit distances are better preserved by the two tensor sketching methods, and more so by tensor slide sketch. This difference is due to the fact that hash-based sketches are sensitive to small edit distances, namely below 0.1 (normalized to length), they plateau more quickly.

We can evaluate these methods more quantitatively, by requiring them to solve a decision edit distance problem. In the decision version of the problem, the task is to decide which points are bellow a certain threshold and which ones are not. Therefore, the performance of each method can be depicted by ROC curve, which are plotted for four different thresholds in Figure S2, with the dataset remains the same as Figure S1. The main takeaway of ROC plots is that while hash-based methods are effective at very low thresholds, they lose their sensitivity for higher thresholds, and their sensitivity declines as task moves away from comparing very similar sequence pairs. Interestingly, tensor slide sketch seems to perform nearly as optimal in the lower threshold, but outperforms other methods on most lower thresholds significantly.

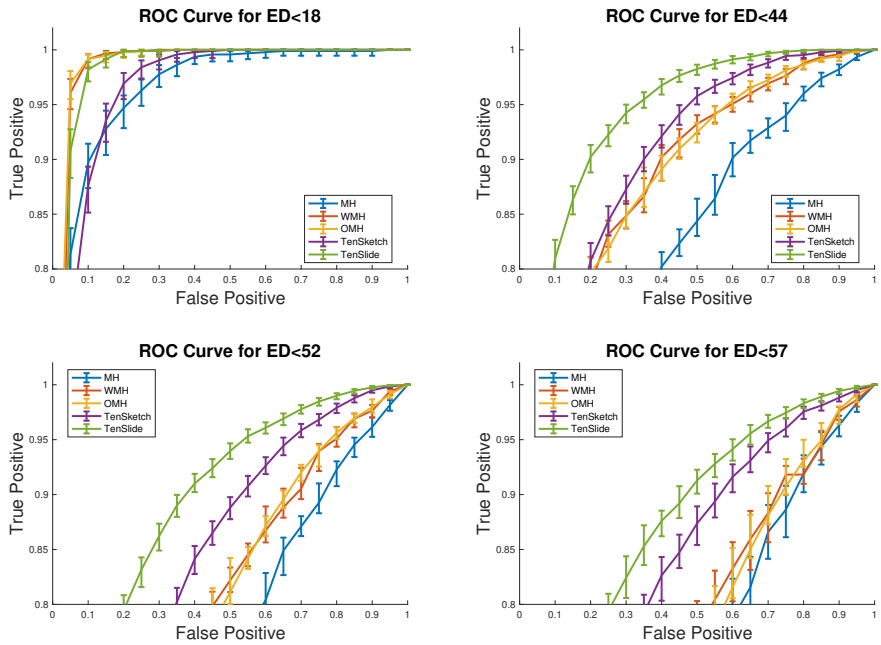


Fig. S2: 10000 sequence pairs of length $n = 100$ were generated over an alphabet of size 4. with the number of random edit operations, uniformy drawn from $\{0, 1\}$