Sequence similarity estimation by random subsequence sketching

Ke Chen[1,*], Vinamratha Pattar[2], and Mingfu Shao[1,3,*]

[1]Department of Computer Science and Engineering, The Pennsylvania State University, PA 16801

[2]Department of Computer Science and Engineering, Amrita School of Computing, Amrita Vishwa Vidyapeetham, Bengaluru, India

[3]Huck Institutes of the Life Sciences, The Pennsylvania State University, PA 16801

[*]Corresponding authors. kxc5915,mxs2589@psu.edu

Abstract

Sequence similarity estimation is essential for many bioinformatics tasks, including functional annotation, phylogenetic analysis, and overlap graph construction. Alignment-free methods aim to solve large-scale sequence similarity estimation by mapping sequences to more easily comparable features that can approximate edit distances efficiently. Substrings or kmers, as the dominant choice of features, face an unavoidable compromise between sensitivity and specificity when selecting the proper k-value. Recently, subsequence-based features have shown improved performance, but they are computationally demanding, and determining the ideal subsequence length remains an intricate art. In this work, we introduce SubseqSketch, a novel alignment-free scheme that maps a sequence to an integer vector, where the entries correspond to dynamic, rather than fixed, lengths of random subsequences. The cosine similarity between these vectors exhibits a strong correlation with the edit similarity between the original sequences. Through experiments on benchmark datasets, we demonstrate that SubseqSketch is both efficient and effective across various alignment-free tasks, including nearest neighbor search and phylogenetic clustering. A C++ implementation of SubseqSketch is openly available at `https://github.com/Shao-Group/SubseqSketch`.

Keywords: Alignment-free sequence comparison, Phylogenetic clustering, Nearest neighbor search, Edit distance embedding

1 Introduction

In a recent paper [7], the authors motivated their sequence sampling method with an interesting puzzle (paraphrased): is the number of DNA 5-mers containing the substring ACGT the same as that for the substring AAAA? Astute readers will immediately answer "no" because it is impossible for a 5-mer to both start and end with ACGT – taking the union of the two disjoint groups gives the correct number – which is not the case for AAAA whose symmetry would cause the same strategy to double-count the 5-mer AAAAA.

As a curious extension, the same question can be asked, replacing substring with *subsequence*, namely, we do not require the containment to be consecutive. This seemingly more complicated version turns out to have a counterintuitively nicer answer: the number of n-mers containing a given k-mer as a subsequence is a function of n and k, independent of the choice of the k-mer. We shall explore this property a bit more in section 2 to facilitate the basic, parameter-free version of our method.

The above is just one of the many nice structural properties of subsequences that, if leveraged properly, can make them more pleasant to work with than substrings. Nonetheless, substring-based methods are still dominant in large-scale sequence comparison, while subsequence-based ones are yet underinvestigated. To compare a large number of sequences by alignment, the common approach is seed-and-extend, where the simplest seeds are k-mers: fixed-length (k) consecutive exact matches in the sequences. More advanced k-mer selection schemes exist, such as minimizer [23, 17], syncmer [5] and k-min-mer [6]. Seeds sampled

from subsequences, either with limited patterns such as spaced seed [2, 14] and strobemer [21, 15, 22], or unrestricted such as SubseqHash [13, 12], usually deliver better performance but are more expensive to compute.

To keep up with the ever-increasing amount of data, sequence comparison methods that do not rely on the expensive alignments gained increased popularity. Among such alignment-free methods, sketching is known for its scalability, efficiency, and versatility. The main idea of sketching is to summarize large genomic sequences into small, fixed-size fingerprints that can be rapidly compared in place of the original sequences. Together with its variants, the most widely used sketching method is MinHash (MH) [1]. In its simplest form, MH utilizes a hash function that maps each k-mer of a sequence to a number and only keeps the k-mer with the lowest hash value as the representative of that sequence. It is easy to see that the probability for two sequences to be represented by the same k-mer is proportional to the Jaccard similarity of the two sequences (viewed as sets of k-mers), namely, the number of shared k-mers between the sequences normalized by the total number of distinct k-mers among them. Hence, by repeatedly choosing min-k-mers with different hash functions and keeping track of the number of occurrences that the picked k-mers are the same between the two sequences, the Jaccard similarity can be estimated. In this process, the list of all representative k-mers of a sequence is called the MH sketch of this sequence. Two MH sketches are compared by the Hamming similarity – number of identical k-mers at the same indices. Order Min Hash (OMH) [18] extends this idea by estimating the weighted Jaccard similarity. Instead of picking one representative k-mer at a time, each entry of an OMH sketch is generated by picking several k-mers and putting them together following the original order in the sequence. By doing so, OMH has been proved to be a locality-sensitive hashing family for the edit distance. A more comprehensive review of sketching algorithms for genomic data can be found at [20]. Note that both MH and OMH can be considered substring-based sketching methods as they pick substrings as the representatives. Though OMH put several k-mers together that effectively forms a subsequence, we note that the kinds of subsequences can be picked is limited. In particular, OMH always picks the subsequences that consist of a fixed number of chunks. Based on the observation in the seeding approaches that picking unrestricted subsequences (ref. SubseqHash) yields better performance than restricting the type of subsequences (ref. spaced seeds, strobmer), we propose SubseqSketch, an unrestricted subsequence-based sketching approach.

Recently, a new subsequence-based sketching method named Tensor Slide Sketch (TSS) [10] has been developed. Instead of picking k-mers from the input sequence, TSS aims at producing a sketch by counting all subsequences. Since there are an exponential number of them, TSS has to group Subsequences in a smart way to facilitate efficient counting. TSS is certainly a sketching method that utilizes unrestricted subsequences (it uses all of them!). However, we argue that picking a representative is fundamentally different from counting. In particular, we observe that the grouping operation appears to have a negative effect for the alignment-free tasks we tested. There are other applications that call for different types of alignment-free methods such as k-mer counting or k-mer spectrum. We suspect that TSS as the subsequence-based counting approach may outperform the substring-based counting approaches in those applications.

Since alignment provides the most comprehensive information about (and often defines) the similarity between sequences, alignment-free methods aim to approximate the underlying unknown alignment efficiently. One way to understand k-mer-based sketching approaches is to view them as trying to estimate the alignment by sampling k-mers strategically. Sequences with higher similarity have more matching base pairs, so it is more likely for a sampled k-mer to be part of an alignment which would then result in a match in the sketches. But gaps in an alignment limit the lengths of matching k-mers. This leads to the commonly observed difficulty in choosing a proper k: larger k is desirable to eliminate spurious matches but there are very few shared long k-mers even between closely related sequences. Sampling subsequences, instead of substrings, is a natural choice for tackling this issue. SubseqSketch builds on this intuition.

In order to compare sketches from different sequences, they need to be generated using some "shared randomness". For example, two MH sketches are comparable only if they used the same random permutation of k-mers. For SubseqSketch, the shared randomness comes from a list L of testing subsequences. Each element in L is a random sequence of some fixed length k and is used to test if the input sequences contain it as a subsequence, hence the name. The number of sequences in L determines the size of the sketches.

2 SubseqSketch

The idea of SubseqSketch is to identify long common subsequences between two sequences through random sampling. Computing the sketch of a sequence s can be figuratively thought of as answering a survey in which each question asks if s contains a random sequence as subsequence. By comparing the answers of two sequences, their similarity can be estimated. We note that this idea does not work well with substrings (k-mers). As the number of k-mers in a sequence is negligible comparing to the number of length-k subsequences, the chance of successfully sampling a reasonably sized common substring is low, even between highly similar sequences.

While being able to sample long subsequences is beneficial for similarity estimation, it becomes computationally expensive for long inputs. In the following section the idea of tokenization is introduced to successfully generalize the above strategy to genome-sized sequences. Along with an "enhanced survey" where the yes/no questions are upgraded to interval scale questions, we present the fully-powered SubseqSketch as an effective and efficient sketching method.

2.1 Tokenized subsequence

A sequence x of length kt over an alphabet Σ can be viewed as a sequence of k "tokens" each of which is a string of length t. We say x is a *tokenized subsequence* of a length-n sequence s if there is a list of indices $1 \leq i_1 < i_2 < \cdots < i_k \leq n - t + 1$ such that the length-t substring of s starting at i_j matches the j-th token of x. Note that when $t = 1$, a tokenized subsequence is a regular subsequence; it is not necessarily the case when $t > 1$, as the tokens are allowed to overlap, see Figure 1 for an example.

s: CTACCCGATTCTAGTAAAA

x: CT CC GA AT AG TA

Figure 1: An example of tokenized subsequence. The bottom sequence x is tokenized with token size 2. It is a tokenized subsequence of the top sequence s, on which the corresponding tokens are underlined. Observe that x is not a regular subsequence of s.

2.2 Construction of SubseqSketch

To construct SubseqSketch for input sequence(s), we first generate a list L of random sequences of length kt, where k and t are predefined parameters. We call L the list of testing subsequences. Two SubseqSketches are only comparable if they were generated with the same list L, i.e., L serves as the shared random information in the sketching. Given an input sequence s, SubseqSketch takes a testing subsequence in L and determines the maximum number of its prefix tokens that form a tokenized subsequence of s. The resulting vector of $|L|$ integers, one for each testing subsequence, is the sketch of s, denoted as SubseqSketch(s). See Figure 2 for an example.

In order to compute the $|L|$ integers efficiently, we can preprocess the input sequence s to build and index that facilitates rapid lookup for the position of the next token of a testing subsequence. For example, for token size $t = 1$, we can build an automata on s in $O(|s||\Sigma|)$ time and space. In the automata, each character s_i stores $|\Sigma|$ pointers. The pointer corresponds to $c \in \Sigma$ points to the next appearence of c after s_i (null if there is no c after s_i). Then for a testing subsequence x, we can simply follow the pointers according to the characters of x, until either a null pointer is encountered or x is exhausted. This takes $O(|x|)$ time for each testing subsequence so the total sketching time is $O(|s||\Sigma| + |x||L|)$.

For larger token size, a similar idea can be applied: we can preprocess s to build a lookup table of size $|\Sigma|^t$ where each entry records the occurring positions of that token on s, either in a sorted array or some other data structures that supports quick search. Each testing subsequence can then be processed by following this lookup table until all tokens are used or the end of a position array is reached. We provide this preprocessing approach as an option in our implementation. However, through experiments we found

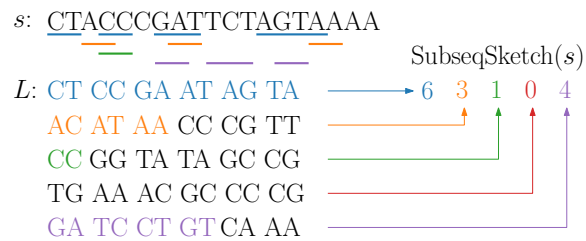

Figure 2: An illustration of SubseqSketch construction with $t = 2$, $k = 6$, and $|L| = 5$. For each testing subsequence in L, its maximum prefix tokens that form a tokenized subsequence of s are colored. Their matching tokens in s are underlined.

that the linear search std::string::find is almost always faster. The overhead of preprocessing may only be justified for a large number of very long testing subsequences with a small token size, which is not a realistic setting for our sketching algorithm.

SubseqSketch tolerates edits while allowing for a fast algorithm. The sketch of a sequence s is a highly informative characterization of s. To see the intuition, consider two sequences s and t. If both sketches show large numbers at the same dimension, then s and t must share a long tokenized subsequence and hence likely similar in terms of the edit distance. Conversely, if one sketch has a large value and the other a small value in the same dimension, then s and t are likely dissimilar. Sketches that are uniformly small provide less information in deciding the similarity of s and t; this is a key reason for using cosine similarity on sketches (see below). Furthermore, we optimize parameters specifically to prevent situations where most entries are small (see below).

Similar to other sketching methods such as MH and OMH, we use a similarity measure on the sketches to estimate the similarity of the input sequences. Unlike MH and OMH which use the Hamming similarity on sketches, the SubseqSketch of two sequences a and b are compared by their cosine similarity, namely

$$\frac{\text{SubseqSketch}(a) \cdot \text{SubseqSketch}(b)}{\|\text{SubseqSketch}(a)\|_2 \, \|\text{SubseqSketch}(b)\|_2},$$

where $\cdot$ is the vector dot product. In Section Results, we show experimentally that the cosine similarity of SubseqSketch exhibits a strong correlation with the normalized edit similarity of the input sequences.

2.3 Choice of parameters

SubseqSketch has three parameters: the token size t, the number of tokens k in each testing subsequence, and the size $|L|$ of the testing list. The parameter $|L|$ controls the size of the sketches. In particular, a SubseqSketch takes $|L| \log k$ bits space to store. As with other sketching methods, a larger sketch gives a better estimation at the cost of increased time and storage. In the experimental sections, we compare the sketching methods at the same sketch size.

The parameters t and k are related. In the resulting sketches, each entry is an integer between 0 and k. If t is too large (for example, close to the input length n), most entries would be 0; on the other hand, if both t and k are small, most entries would max out at k, regardless of the input sequence s. Neither case is desirable as the sketches cannot provide a strong distinction between similar and dissimilar input sequences.

We now try to derive an optimal choice of k. For $t = 1$, as hinted in the introduction, we can compute exactly the number of length-n sequences that contains a given length-k sequence as subsequence. Consider a length-k sequence x, we count the number of length-n sequences s whose subsequence $1 \le i_1 < i_2 < \cdots < i_k \le n$ is x. To avoid overcounting, we only count s if $(i_1, \ldots, i_k)$ is the first occurrence of x in s. It means the characters in s before i_1 cannot be x_1, leaving them $|\Sigma| - 1$ choices each. The same holds for regions in between i_j and i_{j+1}, and finally all characters after i_k are free to be anything in Σ. This leads to $(|\Sigma|-1)^{i_k-k}|\Sigma|^{n-i_k}$ choices. Note that the expression only depends on i_k (i.e., any combination of $i_1, \ldots i_{k-1}$

yields the same number), so we can group the terms and sum over choices of i_k to get the answer

$$\sum_{i_k=k}^{n} \binom{i_k - 1}{k - 1} (|\Sigma| - 1)^{i_k - k} |\Sigma|^{n - i_k}.$$

We emphasize that the calculation is independent of the chosen subsequence. An example is shown in Figure 3. In this example, we can choose $k = 35$ to achieve the goal that most of the entries in the sketches

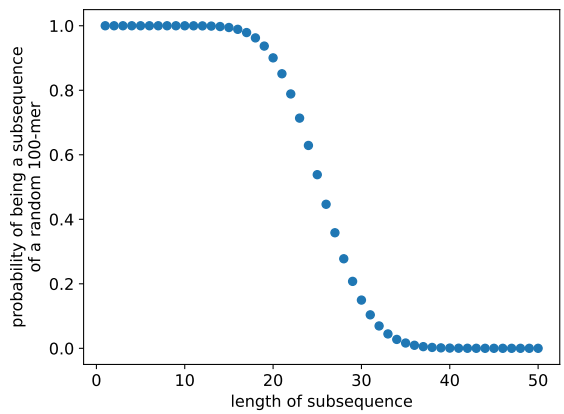

Figure 3: The fraction of length-100 sequences with $|\Sigma| = 4$ that contain a given subsequence as a function of the length of the subsequence.

do not reach the maximum value k.

For larger t, the derivation is not as neat. Since using a small k makes the sketching faster to compute and smaller to store, we recommend $k < 16$ (namely, each entry in the sketch fits in 4 bits). Numerical calculations suggest the empirical formula $t = \lceil (\log_2(n) - 5) \rceil$ works well in ensuring the entries are neither too small nor maxed out. Table 1 lists the recommended values of t for some common input sizes n.

Table 1: Recommended t values for different input sizes n according to the empirical formula.

n	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9
t	2	5	9	12	15	19	22	25

2.4 Sample subsequences from input

Using randomly generated testing sequences is the best one can do in a data-oblivious setting, while better performance can usually be achieved if we can afford to adjust the sketches according to the input data. One idea to introduce data dependency is to sample subsequences from input sequences to form the testing list. This is particularly suitable if the input consists of a small number of sequences, for example, in estimating phylogenetic distances between closely related genomes as shown in Section 3.3. On the other hand, if the sketches are used to build an index of a large database of sequences to handle queries, it may not be practical to re-sketch the entire database with a new testing list for each query. In this situation, we simply use the data-oblivious version with a fixed list of randomly generated testing sequences and demonstrate in Section 3.2 that it already achieves good performance.

3 Experiments

In this section, we first show a strong correlation between the cosine similarity of SubseqSketch with the edit similarity between simulated pairs of sequences. Then the sketch quality of SubseqSketch is tested on two sequence comparison tasks, the nearest neighbor search and phylogeny reconstruction. In each task, we compare SubseqSketch with competing methods on both simulated sequences and published benchmark datasets. For a fair comparison, each method is set to produce sketches of (roughly) the same size. A grid search is performed for each competing method to find the best parameters. Details are reported in each subsection.

3.1 Correlation between sketch similarity and edit similarity

To directly compare the sketch similarity against the desired but much more expensive to compute edit similarity, we generate $100,000$ random DNA sequences of length $1,000$. Each sequence is randomly mutated (an insertion, deletion, or substitution) for a random number of rounds up to $1,000$ to produce a pairing sequence. For each pair, we compute their exact edit similarity, as well as sketch similarities for SubseqSketch, MinHash (MH), Order Min Hash (OMH), and Tensor Slide Sketch (TSS). For each sketching method the Pearson correlation between the exact edit similarity and the sketch similarity over the 100,000 pairs of sequences is reported. We use the implementation of [10] for MH, OMH, and TSS.

Figure 4 shows the scatter plots of all the pairs under different sketching methods. The horizontal axis marks the normalized edit similarity which is computed as one minus the edit distance divided by sequence length. The vertical axis shows the sketch similarities which are normalized to the range $[0, 1]$. Observe that SubseqSketch achieves the best Pearson correlation. Both MH and OMH are good estimators for sequences with high edit similarities but struggle to distinguish dissimilar sequences with edit similarity between 0.5 and 0.8. The TSS similarity exhibits a linear relation with the edit similarity and hence has a higher Pearson correlation than MH and OMH. But it suffers from an extremely large variance, especially for dissimilar sequences, which makes it difficult to interpret the estimation in practical use. SubseqSketch strikes a balance between the ability to estimate the full range of edit similarity and the estimation variance.

As with other sketching methods, the variance of SubseqSketch can be reduced by using a larger sketch. For all the experiments, we measure the size of a sketch as the number of entries in it (sometimes called its dimension), and all methods are configured to produce the same number of entries (except for TSS, which we follow the suggestion in [10] even though it produces a larger sketch). However, in real applications, the actual space needed to store the sketches is a more relevant measure. Recall that each entry of SubseqSketch can be stored in 4 bits (ref. Section 2.3) which is four times smaller than an entry of MH (16 bits for $k = 8$), six times smaller than OMH (24 bits for $k = 6$ and $\ell = 2$), and eight times smaller than TSS (32-bit float). Thus, given a fixed amount of disk space, SubseqSketch can utilize more testing subsequences than the number of k-mers MH or OMH can select, thereby achieving a similar or better variance.

3.2 Nearest neighbor search

Nearest neighbor search asks to find the top-T most similar sequences for a query among a large database. Since computing the exact edit distance of the query with each sequence in the database is too expensive, the general approach is to map the sequences in the database into some well-studied metric space (usually high dimensional Euclidean space) where a nearest neighbor index is readily available (for example, the hierarchical navigable small world index [16]). Then a query can be mapped into the same space, and the nearest neighbors there are reported. In this experiment, we choose to not include any index because the accuracy of the index may affect the final results. Following the pipeline of CNN-ED [4], a tool that performs nearest neighbor search using a learned embedding for edit distance, we compute the sketch distances between a query and all sequences in the database and report the top-T nearest neighbors. It is worth noting that computing sketch distances is much more scalable than computing edit distances.

We show results on two widely used datasets GEN50kS and GEN20kL from [25]. The GEN50kS dataset contains $50,000$ sequences with an average length $5,000$. The GEN20kL dataset contains $20,000$ sequences

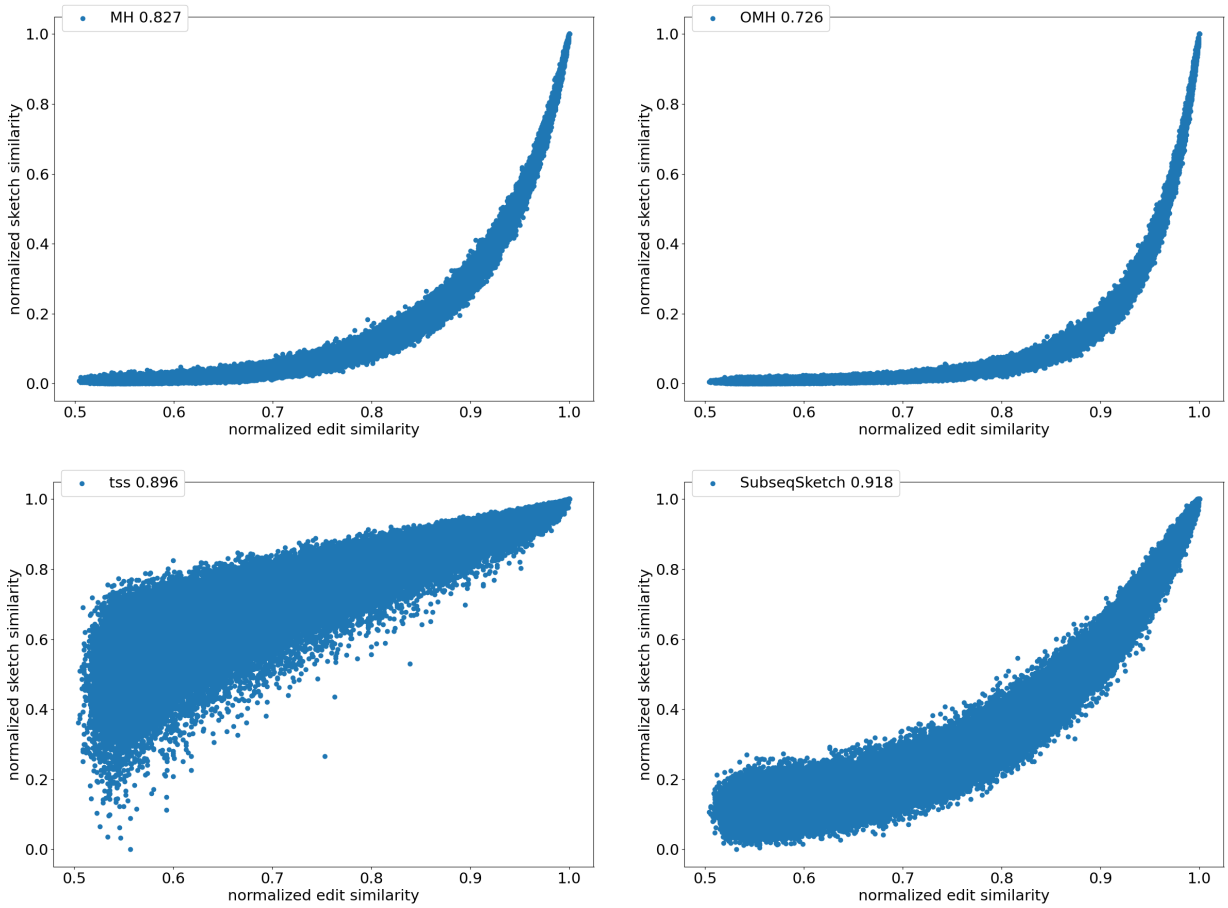

Figure 4: Correlation between normalized sketch similarities and normalized edit similarity on length $n = 1000$ sequences. The legend marks the name of the method and the Pearson correlation.All methods use sketch size 1000. MH uses k-mer size 8. OMH uses k-mer size 6 and $\ell = 2$. TSS uses $t = 2$, dimension 32, window size $0.1n = 100$, stride size $0.01n = 10$, as suggested in [10]. SubseqSketch uses token size 6.

with an average length $20,000$. The CNN-ED pipeline splits each dataset into three disjoint sets: a training set with $1,000$ sequences, a query set with $1,000$ sequences, and a base set containing the remaining sequences. It then computes the all-vs-all edit distances between the query set and the base set to form the ground truth for the nearest neighbor search. For the sketching methods, the training set is not used.

To evaluate the performance of different methods, we plot the widely used recall-item curves in Figure 5 and Figure 6. For a figure labeled top-T, the T nearest neighbors of a query in the base set according to the edit distances are considered true neighbors. The horizontal axis represents the number of neighbors (items) each method is allowed to report (according to their respective sketch/embedding distances) and the vertical axis marks the fraction of true neighbors being reported (recall).

In this experiment we restricted our comparison to CNN-ED, which was shown to outperform other methods such as the CGK embedding [3]. The CNN-ED results are obtained by the implementation of [4]. It is a deep convolutional neural network model which we trained for 50 epochs as reported in the original paper. For a fair comparison, SubseqSketch is configured to produce vectors of the same dimension as the outputs of CNN-ED. Observe that SubseqSketch consistently outperforms CNN-ED by a large margin. This is a surprising result. It is commonly believed (which is often, though not always, justified) that machine learning models should have a better performance than traditional algorithmic methods because the models

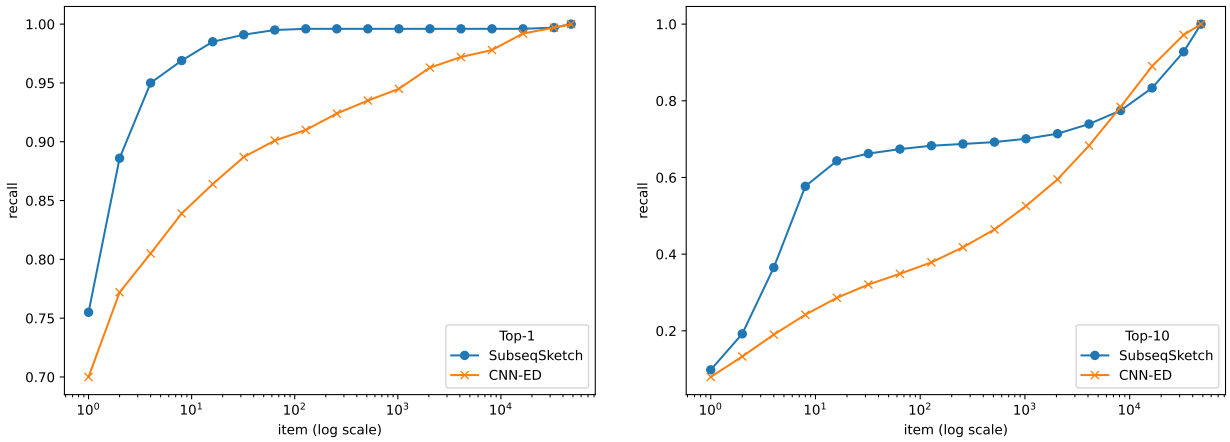

Figure 5: Recall-item curves of different methods on the GEN50kS dataset. All methods output dimension 200. SubseqSketch uses token size 6. Left: ground truth is the top-1 nearest neighbor by edit distance. Right: ground truth contains the top-10 nearest neighbors by edit distance.

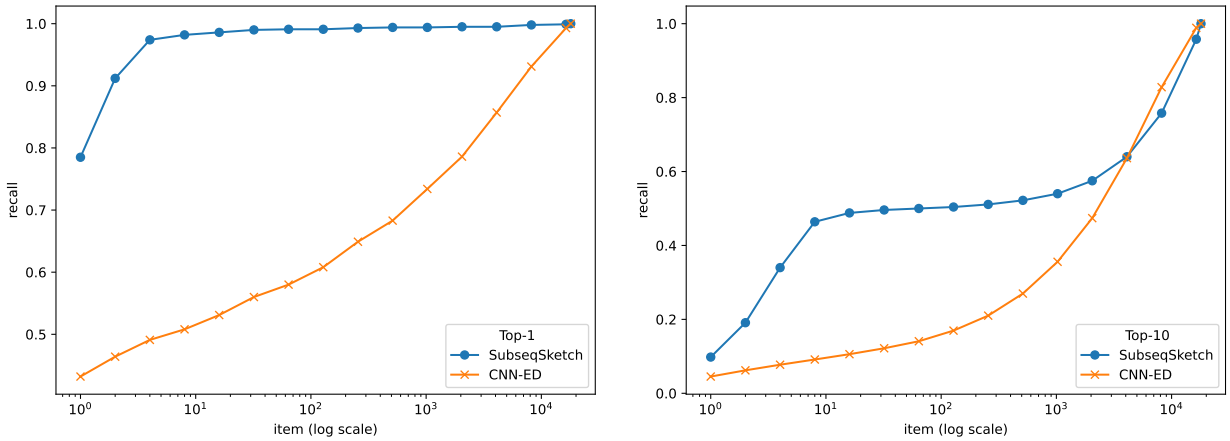

Figure 6: Recall-item curves of different methods on the GEN20kL dataset. All methods output dimension 128. SubseqSketch uses token size 7. Left: ground truth is the top-1 nearest neighbor by edit distance. Right: ground truth contains the top-10 nearest neighbors by edit distance.

can learn data-dependent features that the data-oblivious algorithms cannot take advantage of. In [4], the CGK embedding [3] was shown to produce a worse result than CNN-ED on this task, even though it is an edit distance embedding with theoretical guarantees. Our result here demonstrates that there is a gap between theoretical bounds and practical performance which warrants further investigation. In particular, we conjecture that SubseqSketch can also provide some guarantees on the distortion as a randomized embedding function for the edit distance, though a theoretical proof seems difficult.

3.3 Phylogeny reconstruction

Phylogeny reconstruction is another common task that can be used to evaluate the performance of alignment-free methods. Given a set of biologically related genomes, the goal is to build a phylogeny on them based on pairwise similarities/distances estimated by the sketches. The result can then be compared with a ground

truth tree constructed from some biological model or multiple sequence alignment. We test on two datasets for this task: one is a simulation of a simple mutation model similar to that used in [18]; the other is a set of 29 assembled *E. coli* genome sequences collected in [24].

For both datasets, an all-vs-all distance matrix is computed for each method. For the simulated dataset, the matrices are used to build the phylogenies with the neighbor-joining algorithm implemented in the biotite package [11]. The normalized Robinson-Foulds (nRF) distances between the constructed trees and the ground truth tree are then calculated with the ETE toolkit [8]. The nRF distance measures the dissimilarity of branching patterns between two trees and ignores branch lengths. A value of 0 means the two phylogenies have the identical tree topology, whereas a value of 1 indicates the two trees are maximally dissimilar. For the real *E. coli* genome sequences, the AFproject [26] (a benchmark project for alignment-free sequence analysis tools) provides a web interface where the phylogenies can be computed from the uploaded distance matrices. The nRF distances are then reported by comparing the resulting trees against a ground truth tree built from multiple sequence alignment. It also provides the normalized Quartet Distance (nQD) as an additional measure for topological disagreement. On the website, many alignment-free phylogeny reconstruction tools are ranked based on the nRF distances achieved.

Following the experiment in [18], we simulate a family of sequences using a simple mutation model that includes both point mutations and mobile genomic elements, commonly found in bacterial genome rearrangements, known as insertion sequences (IS). The simulated sequences form a perfect binary tree. The root of the tree is a random sequence of length $10,000$; it is considered as the 0-th generation genome. To obtain the i-th generation, each sequence in the $(i-1)$-th generation produces two children genomes by independent and random point mutations with mutation rate 0.01%. Then a random IS of length 500 is inserted at a random position for each newly generated i-th generation genomes. Note that the IS is shared among all sequences in the same generation, but the inserting positions can be different. See Figure 7 for an illustration. Although simple and somewhat unrealistic, this model produces a solid ground truth phylogeny and allows us to investigate the effectiveness of different sketching methods to recover the mixed history of point mutations and large insertion events.

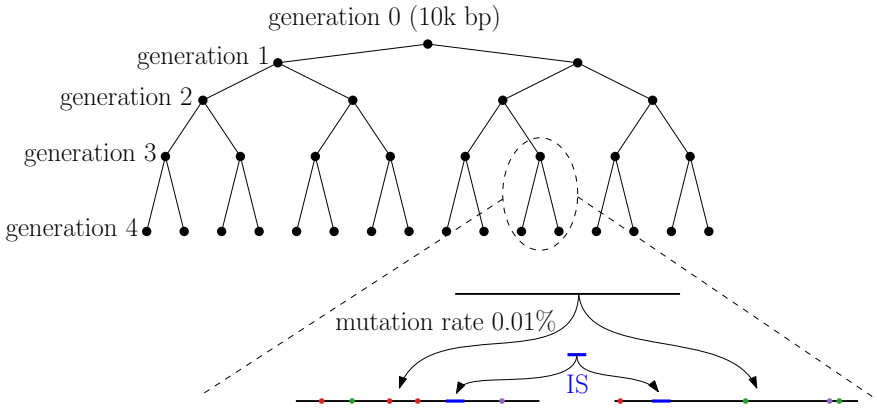

Figure 7: An illustration of the simulated phylogeny. In the zoomed-in view at the bottom, the top segment represents a sequence from the 3-rd generation. Its two children in the 4-th generation are obtained by random point mutations represented by colored dots. The blue segment represents the common IS inserted into each sequence in the 4-th generation.

Figure 8 shows the nRF distances achieved by each method on progressively larger inputs from the simulated dataset. The horizontal label i means all the 2^i sequences from the i-th generation are used as input sequences. Not surprisingly, pairwise edit distance (ED) most accurately captures the mutation history, at the cost of significantly longer computation time (see Figure 9). Among the sketching methods, SubseqSketch constructs the best phylogeny for generation 6 and larger inputs. Furthermore, the nRF distances obtained by SubseqSketch exhibits a strong correlation with those achieved by the exact edit

distances, indicating it can be used as a faithful approximation of the expensive edit calculation. In contrast, although MH and OMH produce trees with smaller nRF distances for the smaller input sets, they both show some inverse relation with the nRF using edit distances (from generation 3 to 4 for MH and from generation 4 to 5 for OMH).

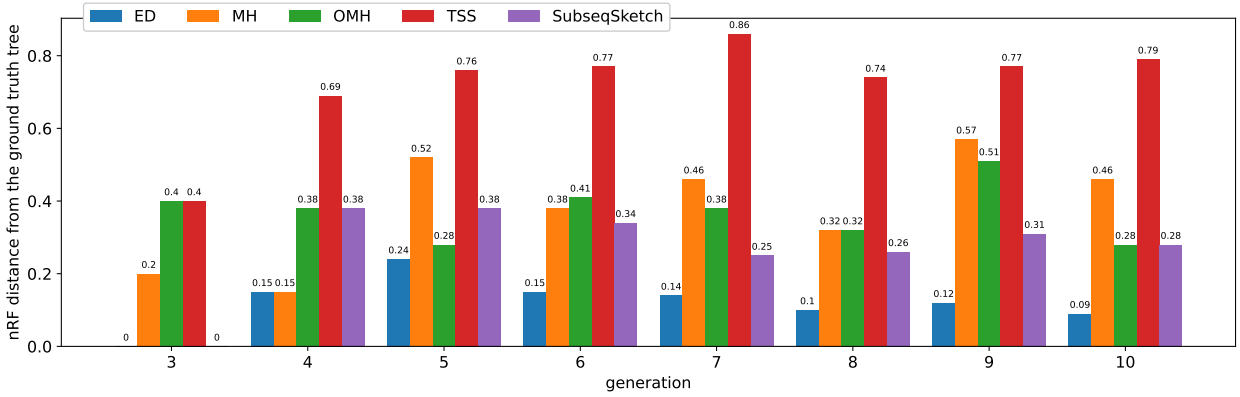

Figure 8: Normalized RF distances achieved by each method on the simulated dataset. A lower nRF distance indicates the constructed phylogeny is more similar to the ground truth tree. All methods use sketch size 256. MH uses k-mer size 8. OMH uses k-mer size 6 and $\ell = 2$. TSS uses $t = 4$, dimension 16, window size $1,000$, and stride size 100. SubseqSketch uses token size 5.

We also plot the running time of each sketching method in Figure 9 to demonstrate the efficiency of SubseqSketch. As expected, all the sketching methods are much faster than computing the all-vs-all exact edit distances. Among them, SubseqSketch is consistently the fastest, regardless of the input size. More specifically, SubseqSketch achieves a 6× speedup compared to the second fastest method (MH).

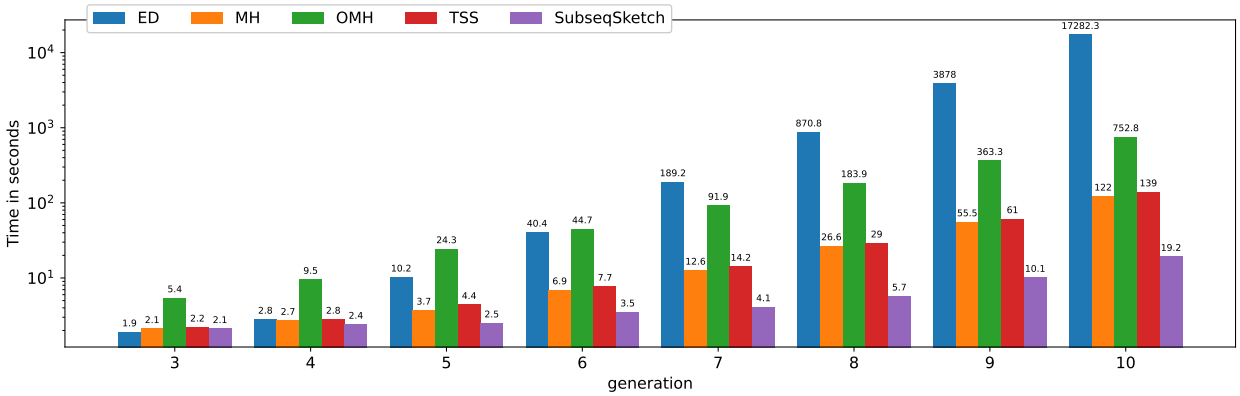

Figure 9: Time spent by each method in seconds (log scale). All experiments run on a server with an Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz. Edit distance is computed with the Python package Levenshtein. MH, OMH, and TSS are computed using the implementation of [10].

Results for the real *E. coli* dataset are summarized in Table 2. On the AFproject website, nearly 100 tools (include different configurations for the same tool) are ranked based on the nRF distance. SubseqSketch is ranked 7th and there are 12 tools that achieve smaller nRF distances due to ties. It is worth pointing out that the higher ranked ones are tools designed for the task of phylogeny reconstruction, which are often based on some sketching method but also apply biological and algorithmic heuristics to adjust the

distance matrix. Since SubseqSketch is a sketching method rather than a complete tool for phylogeny, here we aim to evaluate the sketch quality without the adjustments. By using the raw sketch distance matrices, SubseqSketch constructs the best phylogeny (closest to the ground truth) among MH, OMH, and TSS.

In this task, since there are only 29 genomes, we can afford to sample the testing subsequences from the input to further improve the quality of SubseqSketch. Because the inputs are all closely related, this sampling strategy also enables us to use a much larger token size $t = 40$ to achieve an even better result than the recommended $t = 15$. From Table 2, it is evident that setting $t = 40$ significantly improves accuracy.

Table 2: Phylogeny reconstruction results on 29 *E. coli* genomes. The RF, nRF, and nQD distances all measure topological disagreement between the reconstructed tree and the ground truth tree. A lower value indicates a more accurate reconstruction of the phylogeny. The Rank is based on the nRF distances among many tools tested by the AFproject. All methods use sketch size $10,000$. MH uses k-mer size 10. OMH uses k-mer size 22 and $\ell = 3$. TSS uses $t = 5$, dimension 100, window size $500,000$, stride size $100,000$. The token size used by SubseqSketch is marked in parentheses.

Method	RF	nRF	nQD	Rank
MH	30	0.58	0.3307	13
OMH	30	0.58	0.3645	13
TSS	40	0.77	0.4806	17
SubseqSketch ($t = 15$)	22	0.42	0.1377	9
SubseqSketch ($t = 40$)	18	0.35	0.1679	7

4 Discussion

We presented SubseqSketch, a subsequence-based sketching method that is both effective and efficient at sequence similarity estimation. Comparing to the widely used MH, OMH, and TSS sketches, SubseqSketch requires smaller space, is faster to compute, and achieves a stronger correlation with the edit similarity. It delivers strong performance in two alignment-free tasks: nearest neighbor search and phylogeny reconstruction. In particular, it outperforms a machine learning edit distance embedding model by a large margin which suggests our method indeed captures critical features of the sequences being sketched.

There are numerous interesting directions that call for further investigations. From the theoretical perspective, a deeper understanding of SubseqSketch, and subsequence-based features in general, can be beneficial for better algorithmic designs as well as guiding practical applications. Many methods compared in the experiments come with theoretical guarantees: MH is an unbiased estimator for the Jaccard similarity; OMH is a locality-sensitive hashing (LSH) family for the edit distance; and CGK is an embedding for the edit distance with a quadratic distortion. Given the superior performance of SubseqSketch against these methods, it is natural to consider what bounds can be proved on it. More specifically, we are curious if SubseqSketch is an LSH, and if so, does it offer better hash collision probabilities? Or is it an embedding for the edit distance? In that case, study the relation between its parameters and the achieved distortion can help to make informed decisions in practical use.

On the application side, there are several potential approaches to enhancing SubseqSketch. For example, Mash [19] is a popular tool for genome distance estimation. It is based on MH whose estimation does not exhibit a strong correlation with edit distance. However, by applying a simple Poisson model to adjust the MH estimation, Mash produces a distance that closely approximates the mutation rate on real datasets. Since SubseqSketch starts with a more accurate estimation, it is reasonable to believe that similar techniques can be applied to further improve its performance.

A related question concerns the similarity function used by SubseqSketch. The cosine similarity was chosen because it matches our intuition that sketches of similar sequences should have near identical corresponding entries and therefore should be roughly pointing to the same direction in the sketch vector space.

Furthermore, the cosine similarities between large number of sketches can readily be computed using matrix multiplication which contributes to the efficiency of SubseqSketch. On the other hand, the cosine similarity measures the angles between sketches and explicitly ignores their magnitude. In the extreme case, a sketch full of 1's is considered to have the maximum similarity with another sketch full of 10's. This greatly diverges from the designed meaning of the SubseqSketch entries – the first sequence barely contains any testing subsequences whereas the second contains large portions of each testing subsequence – they must be very different! Exploring different similarity functions that can better incorporate the expected interpretation of the entries can therefore potentially make SubseqSketch more accurate.

Yet another observation is that SubseqSketch is sensitive for global alignment but can struggle with sequences that only share meaningful local alignments. For example, we cannot expect a 1 million base pair sequence to produce a similar SubseqSketch as a 100 base pair sequence. Other sketching methods such as MH also suffer from these situations and special variants such as FracMinHash [9] are designed to handle them differently. As another example, in building overlap graphs for genome assembly, one needs to identify overlapping pairs of sequences that contain additional unaligned prefixes and suffixes. Suppose that the tail of sequence a overlaps with the head of sequence b. Since SubseqSketch tests for subsequences from left to right and stops immediately when the next token cannot be found, the sketches will be disproportionally skewed: because b does not have the beginning part of a, a testing subsequence fully lives inside a can produce a 0 for b, even if b contains a long suffix of that subsequence. We hope to see various versions of SubseqSketch being designed to address these diverse challenges.

Acknowledgments

This work is supported by the US National Science Foundation (2145171 to M.S.) and by the US National Institutes of Health (R01HG011065 to M.S.).

References

[1] Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997.

[2] Andrea Califano and Isidore Rigoutsos. FLASH: A fast look-up algorithm for string homology. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'93)*, pages 353–359. IEEE, 1993.

[3] Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *Proceedings of the 48th ACM Symposium on Theory of Computing (STOC'16)*, pages 712–725, 2016.

[4] Xinyan Dai, Xiao Yan, Kaiwen Zhou, Yuxuan Wang, Han Yang, and James Cheng. Convolutional embedding for edit distance. In *Proceedings of the 43rd international ACM SIGIR conference on Research and Development in information retrieval*, pages 599–608, 2020.

[5] Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, 2021.

[6] Barış Ekim, Bonnie Berger, and Rayan Chikhi. Minimizer-space de Bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer. *Cell Systems*, 12(10):958–968, 2021.

[7] Martin C Frith, Jim Shaw, and John L Spouge. How to optimally sample a sequence for rapid analysis. *Bioinformatics*, 39(2):btad057, 2023.

[8] Jaime Huerta-Cepas, François Serra, and Peer Bork. Ete 3: Reconstruction, analysis, and visualization of phylogenomic data. *Molecular Biology and Evolution*, 33(6):1635–1638, 02 2016.

[9] Luiz Irber, Phillip T Brooks, Taylor Reiter, N Tessa Pierce-Ward, Mahmudur Rahman Hera, David Koslicki, and C Titus Brown. Lightweight compositional analysis of metagenomes with fracminhash and minimum metagenome covers. *BioRxiv*, pages 2022–01, 2022.

[10] Amir Joudaki, Gunnar Ratsch, and André Kahles. Fast alignment-free similarity estimation by tensor sketching. *bioRxiv*, 2020.

[11] Patrick Kunzmann, Tom David Müller, Maximilian Greil, Jan Hendrik Krumbach, Jacob Marcel Anter, Daniel Bauer, Faisal Islam, and Kay Hamacher. Biotite: new tools for a versatile python bioinformatics library. *BMC bioinformatics*, 24(1):236, 2023.

[12] Xiang Li, Ke Chen, and Mingfu Shao. Efficient seeding for error-prone sequences with subseqhash2. *bioRxiv*, pages 2024–05, 2024.

[13] Xiang Li, Qian Shi, Ke Chen, and Mingfu Shao. Seeding with minimized subsequence. *Bioinformatics*, 39(Supplement_1):i232–i241, 06 2023.

[14] Bin Ma, John Tromp, and Ming Li. Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002.

[15] Benjamin Dominik Maier and Kristoffer Sahlin. Entropy predicts fuzzy-seed sensitivity. *bioRxiv*, page 2022.10.13.512198, 2022.

[16] Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

[17] Guillaume Marçais, Dan DeBlasio, and Carl Kingsford. Asymptotically optimal minimizers schemes. *Bioinformatics*, 34(13):i13–i22, 2018.

[18] Guillaume Marçais, Dan DeBlasio, Prashant Pandey, and Carl Kingsford. Locality-sensitive hashing for the edit distance. *Bioinformatics*, 35(14):i127–i135, 2019.

[19] Brian D Ondov, Todd J Treangen, Páll Melsted, Adam B Mallonee, Nicholas H Bergman, Sergey Koren, and Adam M Phillippy. Mash: fast genome and metagenome distance estimation using minhash. *Genome biology*, 17:1–14, 2016.

[20] Will PM Rowe. When the levee breaks: a practical guide to sketching algorithms for processing the flood of genomic data. *Genome biology*, 20:1–12, 2019.

[21] Kristoffer Sahlin. Effective sequence similarity detection with strobemers. *Genome Research*, 31(11):2080–2094, 2021.

[22] Kristoffer Sahlin. Strobealign: flexible seed size enables ultra-fast and accurate read alignment. *Genome Biology*, 23(1):1–27, 2022.

[23] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD/PODS'03)*, pages 76–85, 2003.

[24] Huiguang Yi and Li Jin. Co-phylog: an assembly-free phylogenomic approach for closely related organisms. *Nucleic Acids Research*, 41(7):e75–e75, 01 2013.

[25] Haoyu Zhang and Qin Zhang. Embedjoin: Efficient edit similarity joins via embeddings. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 585–594, 2017.

[26] Andrzej Zielezinski, Hani Z Girgis, Guillaume Bernard, Chris-Andre Leimeister, Kujin Tang, Thomas Dencker, Anna Katharina Lau, Sophie Röhling, Jae Jin Choi, Michael S Waterman, et al. Benchmarking of alignment-free sequence comparison methods. *Genome biology*, 20:1–18, 2019.